Argonne NATIONAL LABORATORY

Progress with PETSc on Manycore and GPU-based Systems on the Path to Exascale

Richard Tran Mills

(with contributions from Hannah Morgan, Karl Rupp, Jed Brown, Matthew Knepley and Barry Smith

PETSc 2019 User Meeting, Atlanta, GA, USA
June 6, 2019

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.

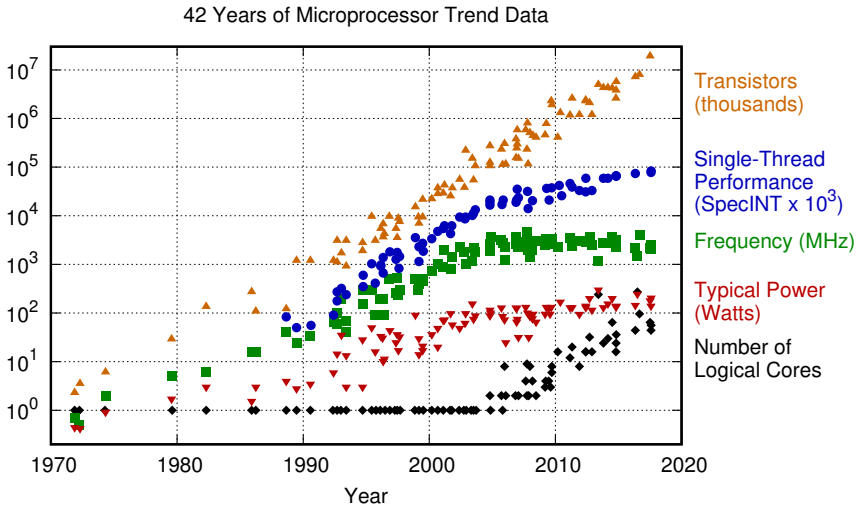
Dennard Scaling (1974)

- 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...

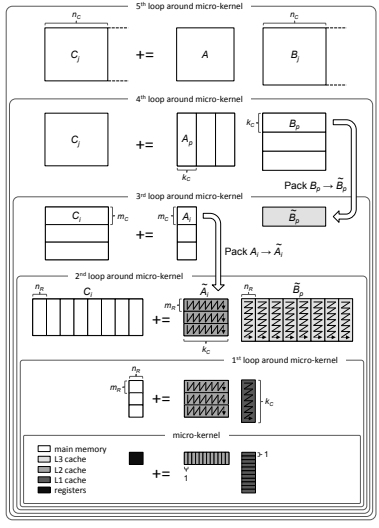
- Lots of hardware cores, hardware threads
- Wide SIMD registers
- Increasing reliance on fused-multiply-add (FMA), with multiple execution ports, proposed quad FMA instructions
- Multiple memories to manage (multiple NUMA nodes, GPU vs. host, normal vs. high-bandwidth RAM, byte-addressable NVRAM being 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.

Some principles guiding our development work

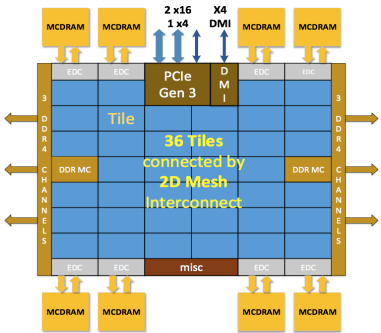
- Defer algorithmic choices until execution time, and enable complex composition of multi-layered solvers via runtime options
- Strive to separate *control logic* from *computational kernels*
  - Allow injecting new hardware-specific computational kernels without having to rewrite the entire solver software 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



[F. Van Zee, T. Smith, ACM TOMS 2017]

Manycore Computing Architectures

- In recent years, the number of compute cores and hardware threads has been dramatically increasing.
- Seen in GPGPUS, “manycore” processors such as the Intel Xeon Phi, and even on standard server processors (e.g., Intel Xeon Skylake).
- There is also increasing reliance on data parallelism/fine-grained parallelism.
  - Current Intel consumer-grade processors have 256-bit vector registers and support AVX2 instructions.
  - Second-generation Intel Xeon Phi processors and Intel Xeon (Skylake and beyond) server processors have 512-bit vectors/AVX512 instructions.

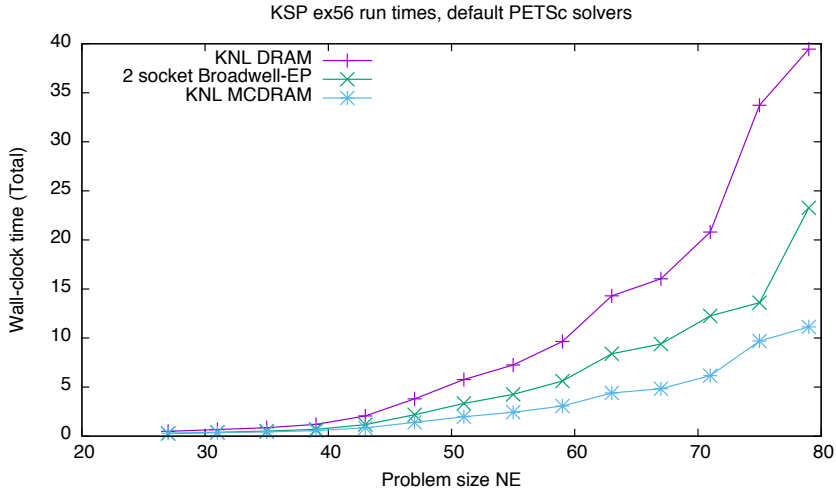


At left, “Knights Landing” (KNL) Xeon Phi processor:

- 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 (16 `float`s wide) SIMD lanes, AVX512 vector instructions
- High bandwidth memory (MCDRAM) on package: 490+ GB/s bandwidth on STREAM triad[2]
- Powers the NERSC Cori and ALCF Theta supercomputers
- Similarities to announced post-K computer (512 bit SIMD, high core counts, high-bandwidth memory)

KSP ex56: Linear Elasticity on KNL

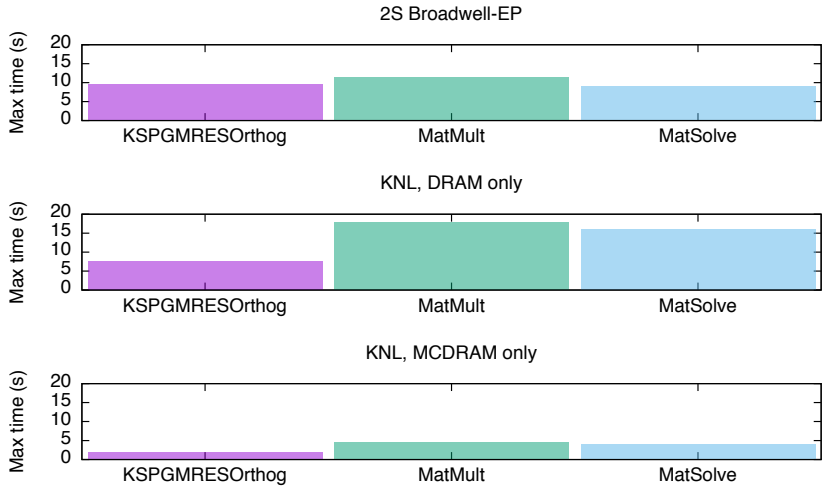
Using PETSc default solvers: GMRES(30), block-Jacobi, ILU(0) on blocks




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

KSP ex56: Linear Elasticity on KNL

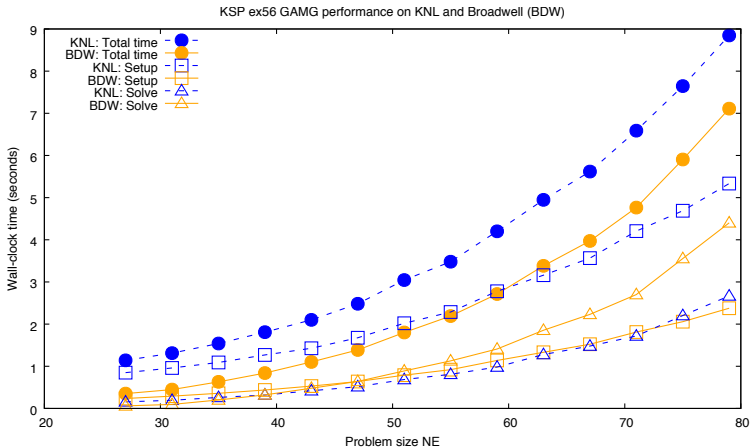
Using PETSc default solvers: GMRES(30), block-Jacobi, ILU(0) on blocks




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

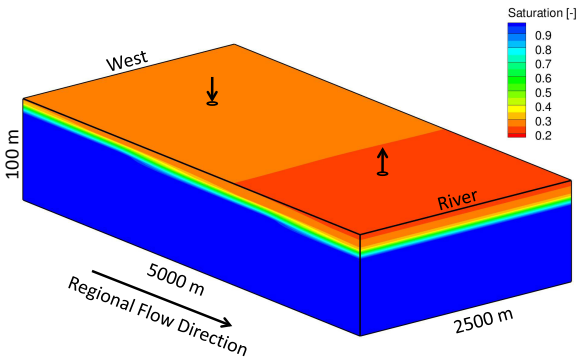
KSP ex56: Using PETSc GAMG Algebraic Multigrid

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



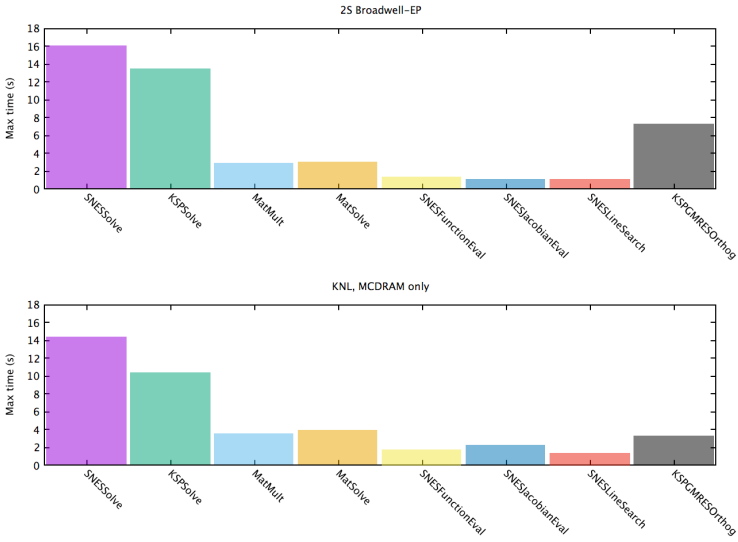
- "Solve" phase is quite fast on KNL.
- Unoptimized setup is comparatively slow on KNL.
- We are working on new AVX-512 and GPU-optimized sparse matrix-matrix multiply.

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 anisotrophy: 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)

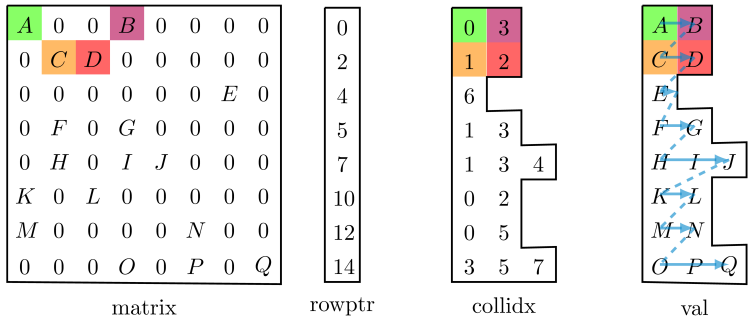
PFLOTRAN Performance on KNL: Out-of-box



- 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.

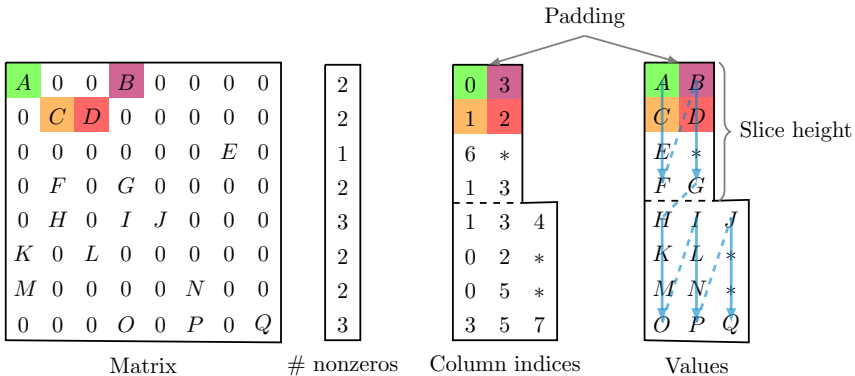


Two main disadvantages with AIJ representation:

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 (with variable slice width).
- Supports sparse matrix-vector multiplication, Jacobi and SOR smoothers.
- Provide AVX and AVX-512 intrinsics implementations.
- Also provide MATAIJSELL subclass of MATAIJ "shadow" copy of SELL matrix with AIJ for fallback operations.
- See Hong Zhang's talk this afternoon!



MATAIJMKL: MKL Sparse BLAS Support

Another matrix class targeting KNL (or other Intel CPUs)

- MATAIJMKL (and block variant MATBAIJMKL) matrices facilitate use of MKL sparse BLAS operations (including new sparse inspector-executor routines).
- Inherits from MATAIJ, but overrides some methods to call MKL routines.
- Usage as simple as

```
mpirun -n $N ./executable -mat_type aijmkl
```

or

```
export PETSC_OPTIONS=-mat_seqaij_type seqaijmkl
```

- Latter option will make all sequential AIJ matrices default to `seqaijmkl`.
- Former option allows different sequential matrix types for diagonal and off-diagonal blocks in MPIAIJ; enables leveraging optimizations in PETSc for zero rows.

Provides another avenue for thread support

- Compile PETSc with OpenMP and link with `-lmkl_intel_thread`.
- Use AIJMKL matrices and set `MKL_NUM_THREADS`.

Overview of GPU Support in PETSc

Transparently use GPUs for common matrix and vector operations, via runtime options — no change of user code required.

CUDA/cuSPARSE:

- CUDA matrix and vector types:
  `-mat_type aijcusparse -vec_type cuda`
- GPU-enabled preconditioners:
  - GPU-based ILU: `-pc_type ilu -pc_factor_mat_solver_type cusparse`
  - Jacobi: `-pc_type jacobi`

ViennaCL:

- ViennaCL matrix and vector types:
  `-mat_type aijviennacl -vec_type viennacl`
- Compute backend selection (CUDA, OpenCL, or OpenMP):
  `-viennacl_backend opencl`
- Switch between CUDA, OpenCL, or OpenMP (CPU) at runtime
- GPU-enabled preconditioners:
  - Fine-grained parallel ILU: `-pc_type chowiluviennacl`
  - Smoothed aggregation AMG: `-pc_type saviennacl`

GPU Support—How Does it Work?

Host and Device Data

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

Possible Flag States

```
  typedef enum {PETSC_CUSP_UNALLOCATED,
                PETSC_CUSP_GPU,
                PETSC_CUSP_CPU,
                PETSC_CUSP_BOTH} PetscCUSPFlag;
```

GPU Support—How Does it Work?

Fallback-Operations on Host

- Data becomes valid on host (`PETSC_CUSP_CPU`)

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

Accelerated Operations on Device

- Data becomes valid on device (`PETSC_CUSP_GPU`)

```
PetscErrorCode VecAYPX_SeqCUSP(..) {
  VecCUSPGetArrayReadWrite(...);
  // some operation on raw handles on device
  VecCUSPRestoreArrayReadWrite(...);
}
```

Example

KSP ex12 on Host

-

```
$> ./ex12
    -pc_type none -m 200 -n 200 -log_summary
```

```
KSPGMRESOrthog      1630 1.0 4.5866e+00
KSPSolve               1 1.0 1.6361e+01
```

KSP ex12 on Device

-

```
$> ./ex12 -vec_type cusp -mat_type aijcusp
    -pc_type none -m 200 -n 200 -log_summary
```

```
MatCUSPCopyTo          1 1.0 5.6108e-02
KSPGMRESOrthog      1630 1.0 5.5989e-01
KSPSolve               1 1.0 1.0202e+00
```

GPU Pitfalls

Pitfall: Repeated Host-Device Copies

- PCI-Express transfers kill performance
- Complete algorithm needs to run on device
- Problematic for explicit time-stepping, etc.

Pitfall: Wrong Data Sizes

- Data set too small: Kernel launch latencies dominate
- Data set too big: Out of memory

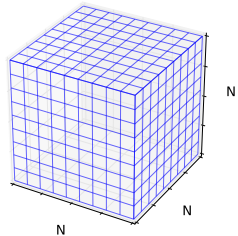
Pitfall: Function Pointers

- Pass CUDA function “pointers” through library boundaries?
- OpenCL: Pass kernel sources, user-data hard to pass
- Composability?

GPU Pitfalls

Pitfall: GPUs are too fast for PCI-Express

- Example system: 720 GB/sec from GPU-RAM, 16 GB/sec for PCI-Express
- 40x imbalance (!)



Compute vs. Communication

- Take $N = 512$, so each field consumes 1 GB of GPU RAM
- Boundary communication: $2 \times 6 \times N^2$: 31 MB
- Time to load field: 1.4 ms
- Time to load ghost data: **1.9 ms (!!)**

Recap: GPU Support in PETSc

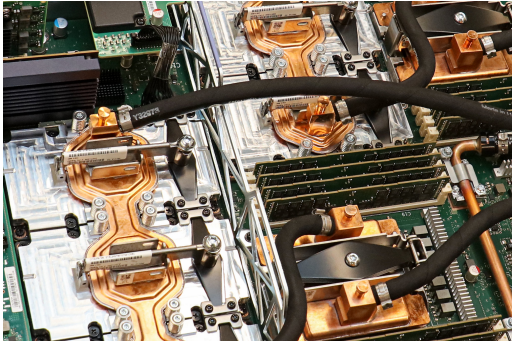
Current GPU-Functionality in PETSc

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

Current Work

- Optimized sparse matrix-matrix multiply (CPUs and GPUs)
- GPU-acceleration for GAMG algebraic multigrid
- Expand use of cuBLAS and cuSPARSE
- Plugin for NVIDIA AmgX algebraic multigrid preconditioner
- Better support for $n > 1$ processes
  (smarter gather/scatter between GPU and host)

OLCF Summit Supercomputer



System totals

- $\sim$ 200 PFlop/s theoretical peak
  143 PFlop/s LINPACK—#1 in TOP500
- 4,608 compute nodes

Node configuration

- Compute:
  - Two IBM Power9 CPUs, each 22 with cores, 0.5 DP TFlop/s
  - Six NVIDIA Volta V100 GPUs, each with 80 SMs–32 FP64 cores/SM, 7.8 DP TFlop/s
- Memory:
  - 512 GB DDR4 memory
  - 96 (6×16) GB high-bandwidth GPU memory
  - 1.6 TB nonvolatile RAM (I/O burst buffer)

Almost all compute power is in GPUs!

Early Summit Results: SNES ex19 with multigrid

Running SNES ex19 (velocity-vorticity formulation for nonlinear driven cavity) with 37.8 million total degrees of freedom on single Summit node.

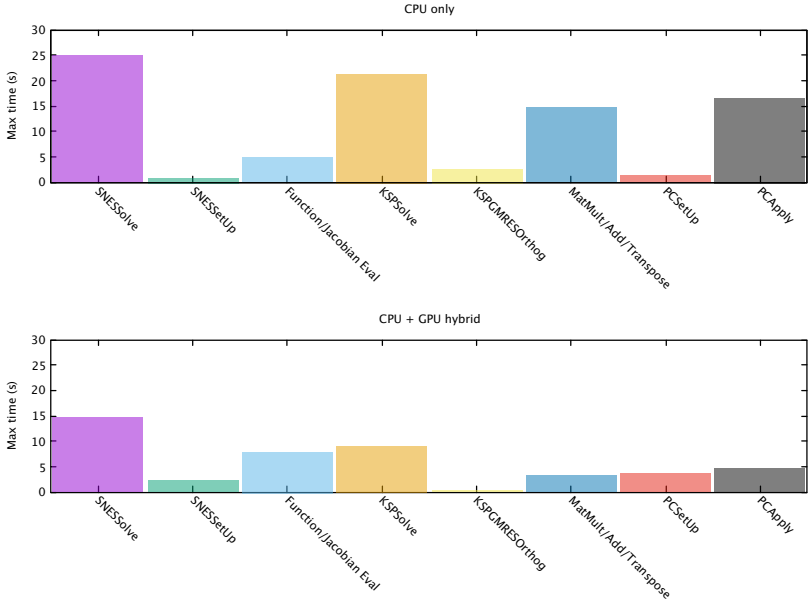
CPU only command line:

```
jsrun -n 6 -a 7 -c 7 -g 1 ./ex19 -cuda_view -snes_monitor -pc_type mg -
    da_refine 10 -snes_view -pc_mg_levels 9 -mg_levels_ksp_type chebyshev -
    mg_levels_pc_type jacobi -log_view
```

CPU + GPU hybrid command line:

```
jsrun -n 6 -a 4 -c 4 -g 1 ./ex19 -cuda_view -snes_monitor -pc_type mg -
    dm_mat_type aijcusparse -dm_vec_type cuda -da_refine 10 -snes_view -
    pc_mg_levels 9 -mg_levels_ksp_type chebyshev -mg_levels_pc_type jacobi -
    log_view
```

Early Summit Results: SNES ex19 with multigrid



Summary: Using Manycore CPUs and GPUs with PETSc

KNL can provide good “out-of-box” performance for some PETSc problems

- **If we stay in MCDRAM!**
- If the problems are solver dominated; “physics”, i.e., residual and Jacobian formulation routines may need more work for efficient vectorization.
- MATSELL, MAIAIJSELL, MATAIJMKL matrix classes may improve performance.
- Hand-optimized AVX-512 routines in PETSc can also help on new Intel Xeon CPUs.

Significant work on GPGPU support in PETSc is also ongoing

- E.g., recently added GPU-friendly ILU and smoothed aggregation algebraic multigrid through ViennaCL.
- Working on optimizing multigrid (geometric and algebraic) on GPUs.
- NVLINK interconnect may significantly reduce offload bottlenecks, making GPGPU use more practical.

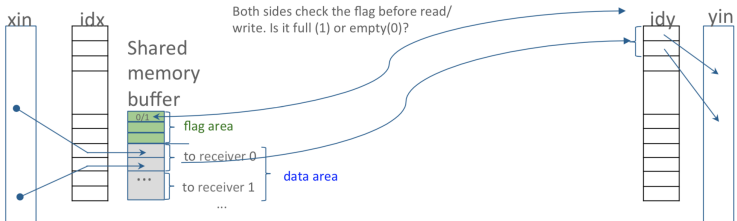
Complementary work: Scalable communication within and across nodes

The focus of this presentation — writing code to effectively use vector units and thread warps present in systems approaching exascale-class systems — is a key challenge.

With increasing core and node counts, effectively communicating/coordinating within nodes and across the entire supercomputer is another key challenge the PETSc team is working to address.

MPI-3 Shared Memory Windows

- Three algorithms implemented for vector scatters using direct load/store within MPI-3 shared memory windows



- Sender allocates a shared memory buffer padded with a header (the flag area)
- Sender assigns each of its receiver a flag. Each flag takes a cache line to avoid false sharing
- Receiver knows addresses of flag & data of each of its senders
- Sender: if flag is 0, safely write data in buffer, and then set flag to 1
- Receiver: if flag is 1, safely read data from buffer, and then set flag to 0

- New `VECNODE` vector type (stored in distributed shared memory) and `VECSCATTERMPI3NODE` scatter type.
  - Choose at runtime: `mpirun -n <np> ./ex2 -vec_type node -vecscatter_type mpi3node`

Pipelined Krylov Methods

- Reductions needed for inner products and norms in Krylov methods require global synchronizations that become very expensive at high node counts.
- PETSc implements several pipelined Krylov methods that hide latency of global reductions: 3 variants of CG, flexible CG, flexible GMRES, BiCGStab, conjugate residuals

Figure: Schematics of the main loop of the Flexible Conjugate Gradient (FCG; left) and the Modified Pipelined Flexible Conjugate Gradient (PIPEFCG; right) methods. Data dependencies in FCG preclude overlapping reductions with the application of the operator or the preconditioner. At the price of increased local work and storage, PIPEFCG reductions can overlap operator and preconditioner application.