Present and Future Computing Requirements for PETSc

Jed Brown jedbrown@mcs.anl.gov

Mathematics and Computer Science Division, Argonne National Laboratory
Department of Computer Science, University of Colorado Boulder

NERSC ASCR Requirements for 2017
2014-01-15

Extending PETSc's Hierarchically Nested Solvers

ANL	**Lois C. McInnes**, **Barry Smith**, Jed Brown, Satish Balay
UChicago	Matt Knepley
IIT	Hong Zhang
LBL	Mark Adams

- Linear solvers, nonlinear solvers, time integrators, optimization methods (merged TAO)
- Maximize versatility and efficiency for existing and new applications
- Performance portability from laptops to Top 10 systems
- Algorithm R&D for fundamental bottlenecks (e.g., memory bandwidth)

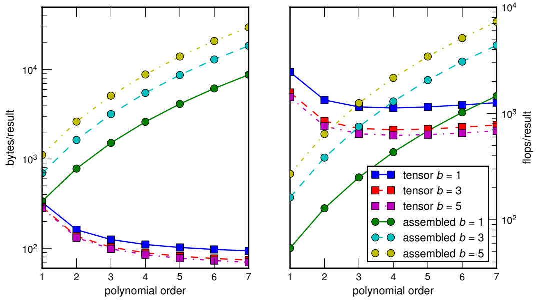
Library-oriented workflow

- Partner with specific applications, but provide features to all
  - Recognize commonality, simple and versatile abstractions, reusable implementation
  - We don’t know most of our users, avg 50 emails/day
  - We hope to hear about problems (algorithmic/convergence, performance, portability). Respond quickly.
- Erratic use of supercomputing
  - New development on laptops, workstations, and small servers
  - Need to test new algorithm or new implementation
  - Validate our expected performance models, find bottlenecks
  - User reports scalability problem and we need to reproduce
- Very short duration jobs across a range of sizes
  - Strong and weak scaling studies
  - Large jobs rarely run for more than 10 minutes
  - Small runs take longer for strong scaling (up to a couple hours)
  - Debug our code, user code (occasionally), and system implementation (e.g., MPI)
  - 1-2 million hours at NERSC and ALCF/OLCF

Computational characteristics

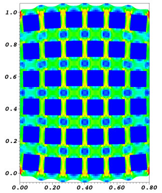
- Assembled sparse linear algebra
    - Arithmetic Intensity $<$ 0.25 flops/byte (cf. hardware at 8 flops/byte)
    - Comfortable abstraction
    - Adaptive coarsening, problems with poor geometric/multilevel structure
    - Research: find higher level structure (UQ, implicit Runge-Kutta)
    - Research: matrix-free, nonlinear methods, exotic multigrid/ephemeral data
- Communication
    - Neighbor “halo exchange” (bounded number of neighbors)
    - Long-range communication to coarse process sets
    - Reductions (blocking and non-blocking) where mathematically necessary (orthogonality)

Research exemplars

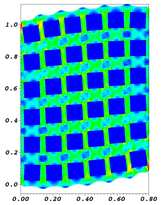


Implicit Runge-Kutta

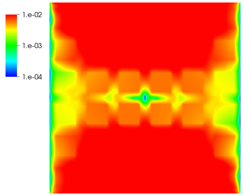
$$G = I \otimes S + J \otimes I$$



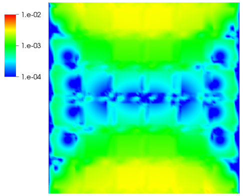
(a) Initial solution.



(b) Increment.



(c) Smoothed error without τ.



(d) Smoothed error with τ.

What is “scalability”?

- **Transient simulation does not weak scale.**
    - Fixed turn-around needed: policy, manufacturing/supply-chain, active control, real-time guidance (field work, surgery, etc.)
    - d-dimensional problem, increase resolution by $2\times$.
    - Data increases by 2^d, but we need $2\times$ more time steps (hyperbolic).
    - With perfect scaling, we use 2^{d+1} more cores.
    - Local data changes by $2^d/2^{d+1} = \frac{1}{2}$
- More applications feeling this
    - Asymptotics are relentless
    - New analysis requires more solves in sequence
        - From forward simulation to optimization with uncertainty . . .
    - New physics and higher fidelity observation requires more calibration/validation
- Other applications are safe for now
    - Steady-state solves with scalable methods
    - Transient with a small number of time steps
    - Maximize resolution/problem size – memory-constrained
- PETSc emphasizes **versatility**

What is “scalability”?

- **Transient simulation does not weak scale.**
  - Fixed turn-around needed: policy, manufacturing/supply-chain, active control, real-time guidance (field work, surgery, etc.)
  - d-dimensional problem, increase resolution by $2\times$.
  - Data increases by 2^d, but we need $2\times$ more time steps (hyperbolic).
  - With perfect scaling, we use 2^{d+1} more cores.
  - Local data changes by $2^d/2^{d+1} = \frac{1}{2}$
- More applications feeling this
  - Asymptotics are relentless
  - New analysis requires more solves in sequence
    - From forward simulation to optimization with uncertainty . . .
  - New physics and higher fidelity observation requires more calibration/validation
- Other applications are safe for now
  - Steady-state solves with scalable methods
  - Transient with a small number of time steps
  - Maximize resolution/problem size – memory-constrained
- PETSc emphasizes **versatility**

Indirect Challenges

- Irresponsible library dependencies
  - Difficult to install, non-portable, bad error-reporting
  - Lack of 64-bit integers, `__float128`
  - Not scalable in *P* (e.g., ParMETIS), assume non-empty subdomains
- Misbehaving system software
  - Broken features that we cannot test for
    - `getpwuid` on BG/Q exists, but calling it rolls the dice between returning NULL, returning a valid pointer with junk, returning an invalid pointer, and crashing the program without returning.
    - `MPI_Bcast` and `MPI_Comm_split` deadlocking for large core count
    - Users of various skill levels spend time tracking down issues and talking to us.
  - Useless performance and fragile run-time configuration
    - No asynchronous progress for `MPI_Iallreduce` in MPT-5.6+
    - `MPICH_ASYNC_PROGRESS` helps a little (few percent)
    - Affinity for async progress thread
- User knowledge and discipline
  - Module environment changes between configure and make
  - Portability limited by poorly-written code or build system
  - Don't know about the barbed wire and broken glass

Strategies for New Architectures

- Choose the right tool for the job.
- Typical GPU users are the least competent and have the least realistic expectations
- CUDA (CUSP & CUSPARSE) and OpenCL (ViennaCL) Mat & Vec
- Problem for matrix and vector assembly
    - Bad abstractions for calling from thread block. New kernel launches imply moving bulky intermediate to global memory.
- User must write GPU code if they wish to have nonlinear residuals and matrix assembly use GPUs (Amdahl)
- Coupling apps/libs with different threading (OpenMP, TBB, pthreads)
- Defer choice of threading model to run-time (threadcomm)
- Less consistent performance, fragile run-time configuration
- Recommend MPI-only for most users
- Intel MIC is an abject failure of hardware and software. Less efficient than Xeon for dense QR (all sizes), advertise useless OpenCL stack. Maybe the next generation will be acceptable.

Some unpopular opinions

- DSL — informally-specified language with immature compiler
  - Syntax or *semantics*?
  - Ability to use legacy (mature) code? Debuggability?
- Where is the fundamental complexity?
  - Math and CS researchers often self-select a distorted perspective
  - For long-term success, most code is written by domain scientists
  - Material models, MD force models, ML feature extraction
  - “Kernels” can have sprawling dependencies and $> 100 kLOC$
  - Legacy and new experiments, written by non-experts
- Granularity, static versus dynamic, versatility
  - Small subdomains: surface area big compared to volume
  - Over-decomposition lengthens the critical path
  - We hear about 32^3 patches/subdomains
    - How efficient is one 8^3 patch/node? 4^6? 100 fields/cell?
  - Frequency of performance variation compared to latency to redistribute or steal
  - Period of interruption/OS jitter
  - Time between algorithmically-required data dependency?

Summary

- New algorithmic modifications become useful (scaling and hardware balance)
  - Exotic low-communication multigrid
  - Nonlinear and matrix-free methods
  - Tensor product solvers (implicit Runge-Kutta, stochastic Galerkin)
  - Efficient solver support for new discretizations
- PETSc runs at NERSC need to keep pace with diverse user group
- Recommendations
  - Emphasize versatility
  - Identify “cooked” performance experiments
    - Unrealistic problem sizes/turn-around time
    - Artifical configurations
    - Normalize by energy efficiency/acquisition cost/TCO, not shrink-wrap
  - Performance reproducibility, diagnostics, debugging
  - Complicated execution environments will require a lot of education and a lot of support by third parties.