SAND2019-2445C

Exceptional service in the national interest

Sandia National Laboratories

Communication-avoiding & pipelined Krylov solvers in Trilinos

Ichitaro Yamazaki (UTK) & Mark Hoemmen (SNL)

SIAM CSE, 28 Feb 2019

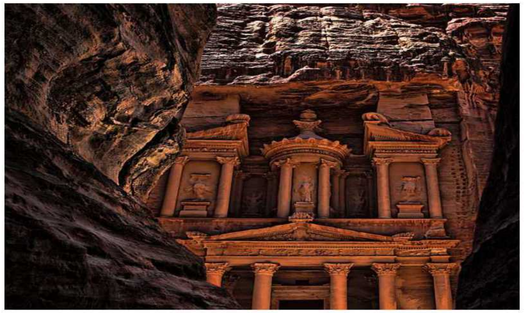
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Communication-avoiding & pipelined Krylov solvers
  - Like other Krylov methods, solve linear systems Ax=b iteratively
  - Avoid (do less) or hide (overlap) communication
  - Algorithms are (mostly) prior work, some our own
- We implemented these solvers in Trilinos
  - Trilinos: Big C++ production math library
  - Parallel: MPI + threads (e.g., OpenMP, CUDA)
  - I'll talk about 2 software engineering challenges today
- Deployed solvers in ECP ExaWind application
  - ExaWind: Simulate multiple wind turbines & wakes in terrain
  - See talk by our NREL collaborators in this minisymposium
  - 1.5x faster on Cori Haswell; results soon on other architectures

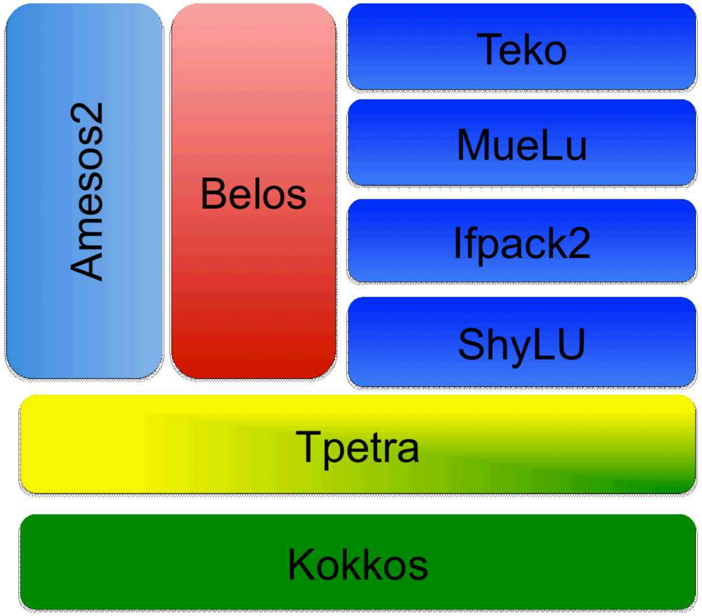
What is Trilinos?

- Parallel math libraries for science & engineering applications
  - Sparse matrices & parallel distributions
  - Linear solvers & preconditioners
  - Nonlinear solvers, optimization, …
- ~ 20 years' continuous development
- Mostly C++11, some C & Fortran
- Supports many different platforms
  - CPUs: x86, KNL, POWER, ARM, …
  - GPUs: NVIDIA, AMD in progress, …
- github.com/trilinos/Trilinos
- Users inside & outside Sandia



Trilinos' linear solvers

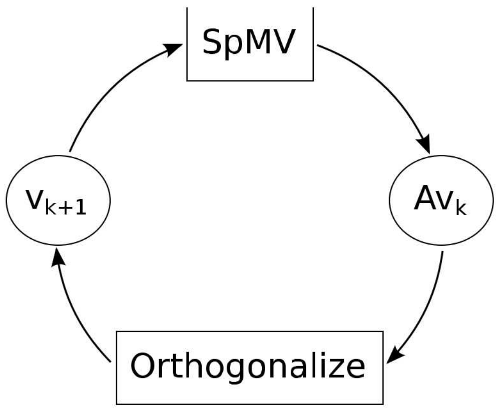
- Iterative linear solvers (Belos)
- Parallel linear algebra (Tpetra)
- Thread parallelism (Kokkos)
- Sparse direct solvers (Amesos2)
- Direct+iterative solvers (ShyLU)
- Algebraic preconditioners (Ifpack2)
- Algebraic multigrid (MueLu)
- Segregated block solvers (Teko)

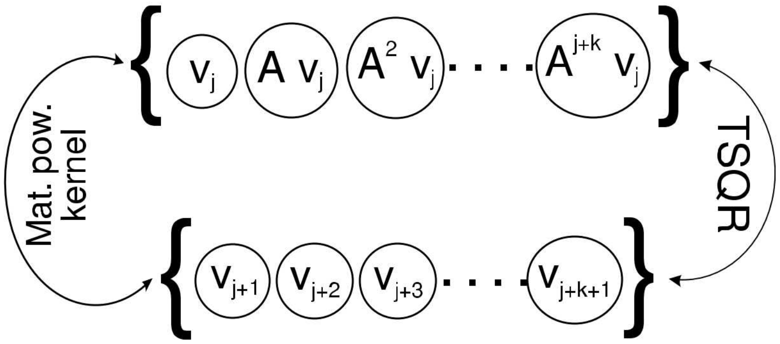


Belos only uses underlying linear algebra implementation through abstract interface (note for later)

Green: Programming model
Yellow: Provide data & kernels
Blue: Use data & kernels directly
Red: Use kernels abstractly

Communication-avoiding Krylov





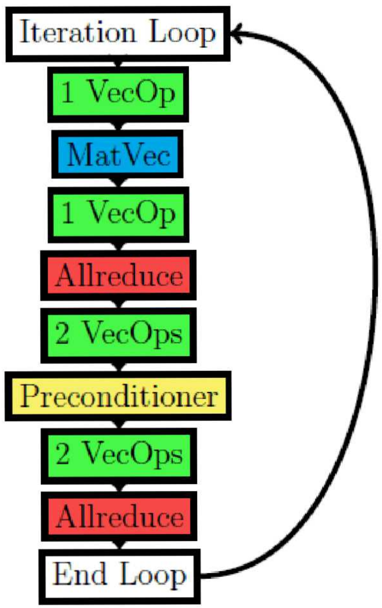
Regular Krylov: data dependency forces >= 2 communication rounds per iteration

Communication-avoiding a.k.a. s-step Krylov: Reorganize algorithm to break dependency

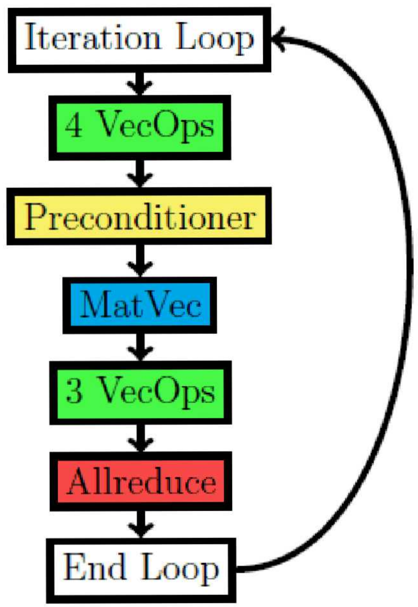
Details: Hoemmen 2010

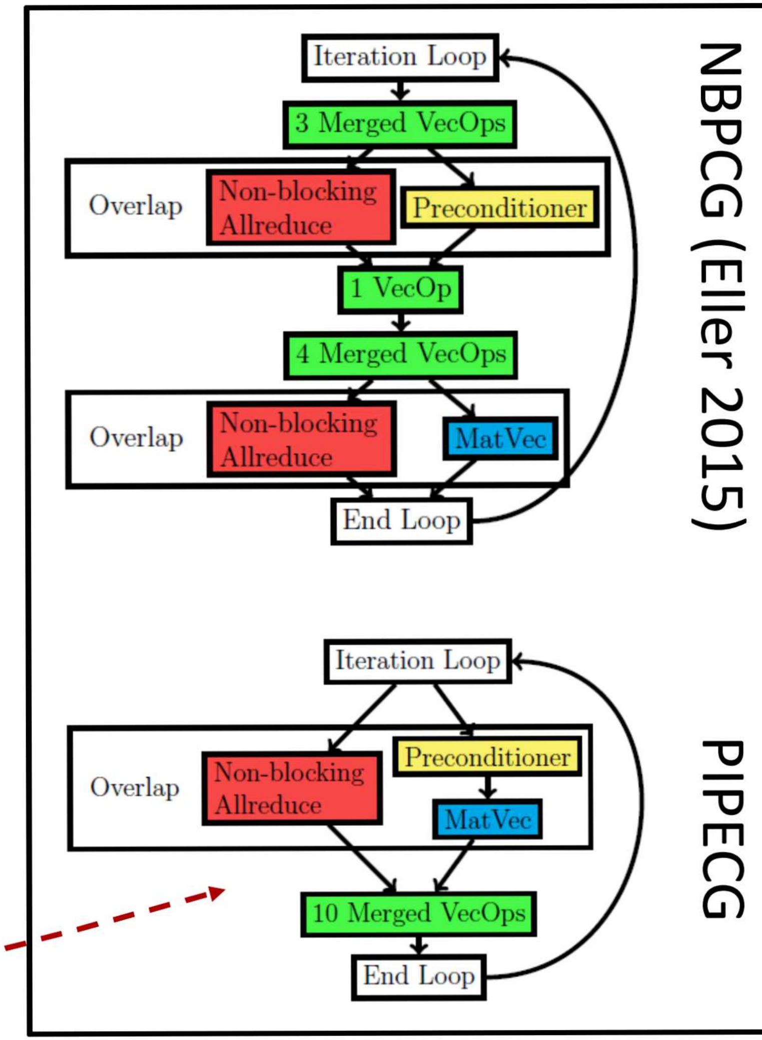
Pipelined Krylov (e.g., CG)

Regular CG



1 all-reduce CG

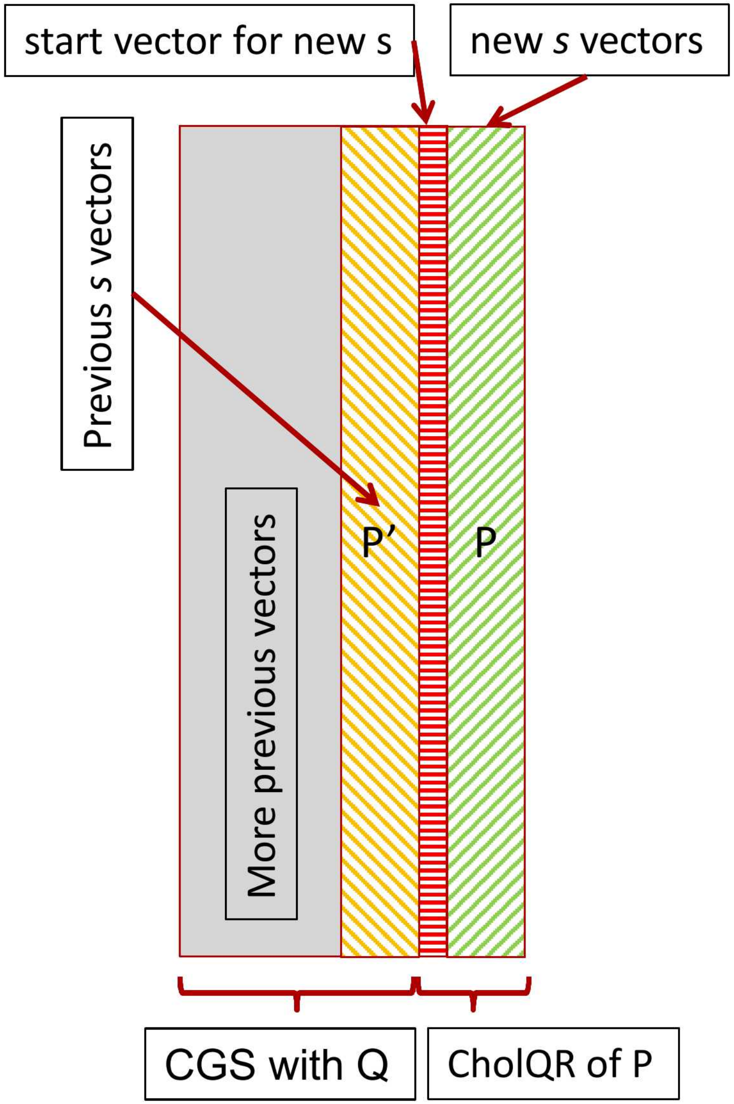




2 pipelined CG variants

We added these to Trilinos

Pipelined CA-GMRES (re)orthog.



- CGS: Classical Gram-Schmidt
- 1 reduce CGS + CholQR
  1. [C; G] = [Q, P]^T * P
  2. Q = Q – C * P, G = G – C^T * C
  3. R = chol(G), Q = Q / R
  - (Next iteration orthog’s P)
- Above + reorthogonalize P
  1. [T, C; G1, G] = [Q, P]^T * [P’, P]
  2. R’ = chol(G1), P’ = P / R’
  3. Update C & G, then 2-3 above
- MGS, CGS2, …: NREL talk today!
- PDSEC’17; adding to Trilinos soon

dot

dot

Krylov methods we implemented

- Available now in Trilinos
  - Pipelined CG (Ghysels & Vanroose 2012)
  - 1 all-reduce CG (Saad ‘85, D’Azevedo ‘93)
  - Pipelined GMRES (Ghysels et al. 2016)
  - 1 all-reduce GMRES (Ghysels et al. 2016)
  - CA-GMRES (Hoemmen 2010)
- Prototypes to be deployed soon
  - Pipelined CA-GMRES (Yamazaki)
    - See our PDSEC 2017 paper
    - Results later in this talk
  - Cool ideas from NREL folks

Nalu Wind performance results

- Nalu Wind (CFD)
  - github.com/exawind/nalu-wind
  - Low Mach, unstructured, C++
  - Trilinos & Hypre linear solves
  - Sierra Tool Kit (STK) meshes
  - Can handle >> 10^9 dofs
- Problem: Simulate air flow around wind turbine(s)
  - Hybrid RANS-LES (RANS near blade, LES in wake)
  - 95 M dofs / linear system
  - Segregated physics
  - NERSC Cori: Haswell, 32 c/n

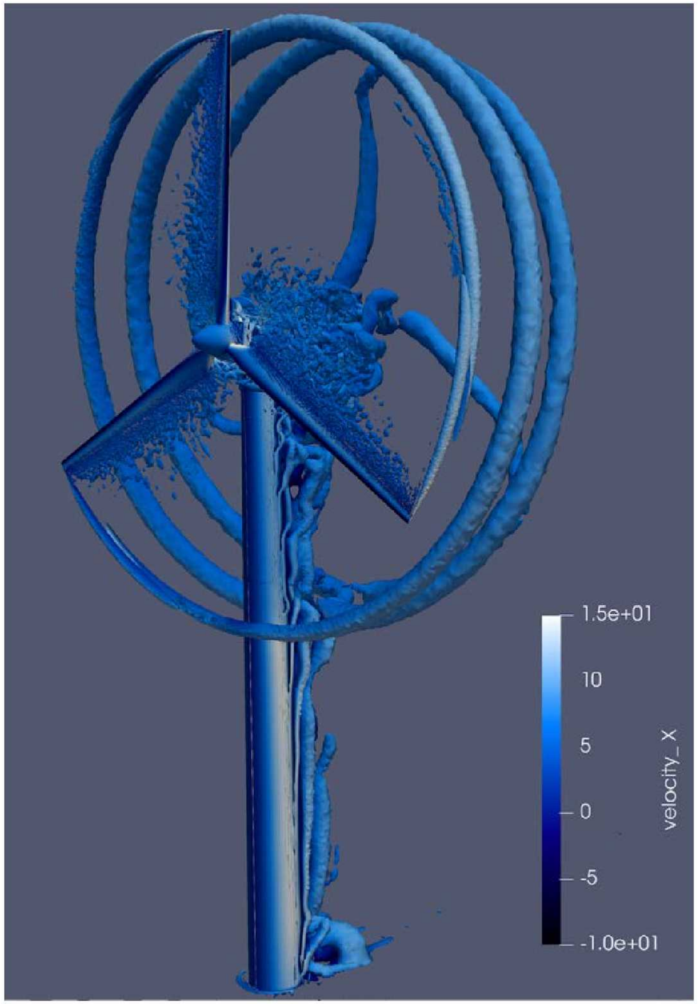
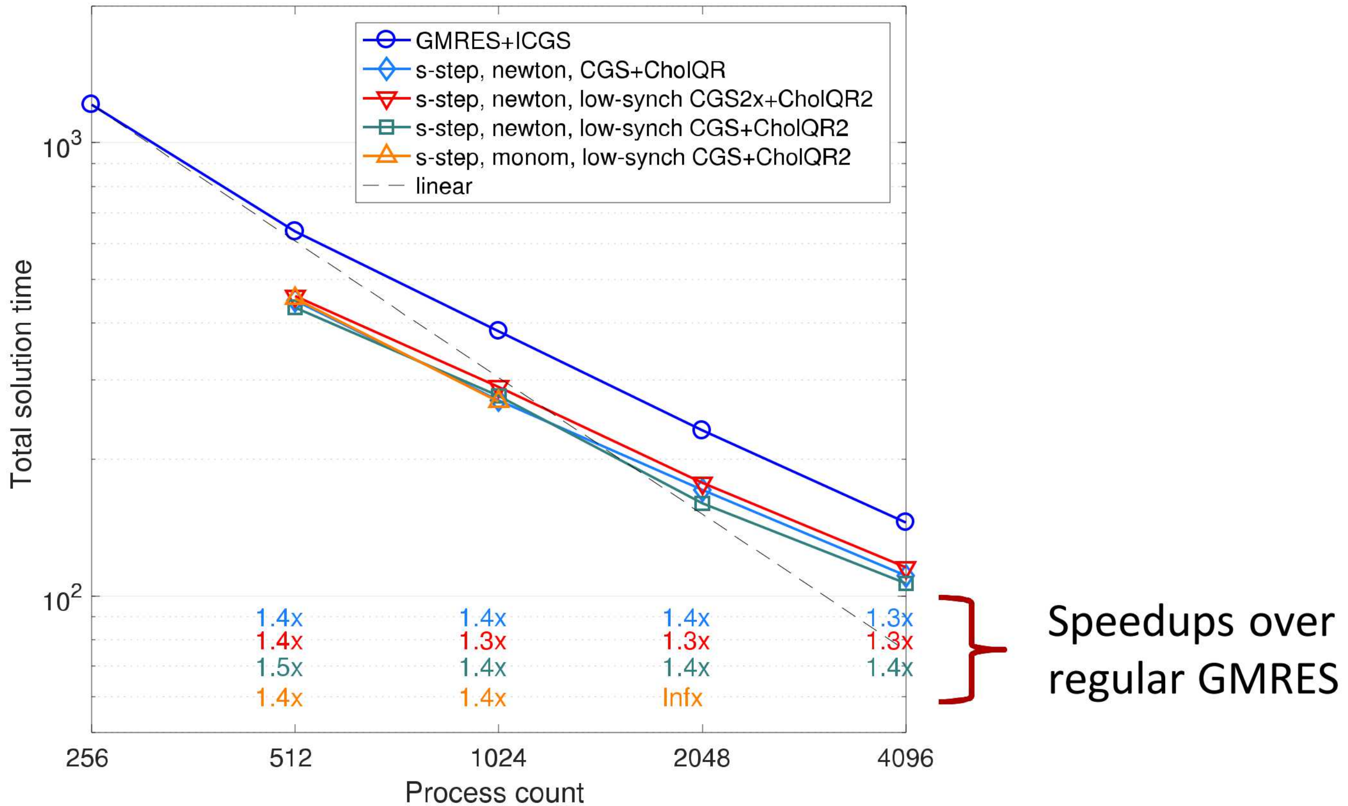


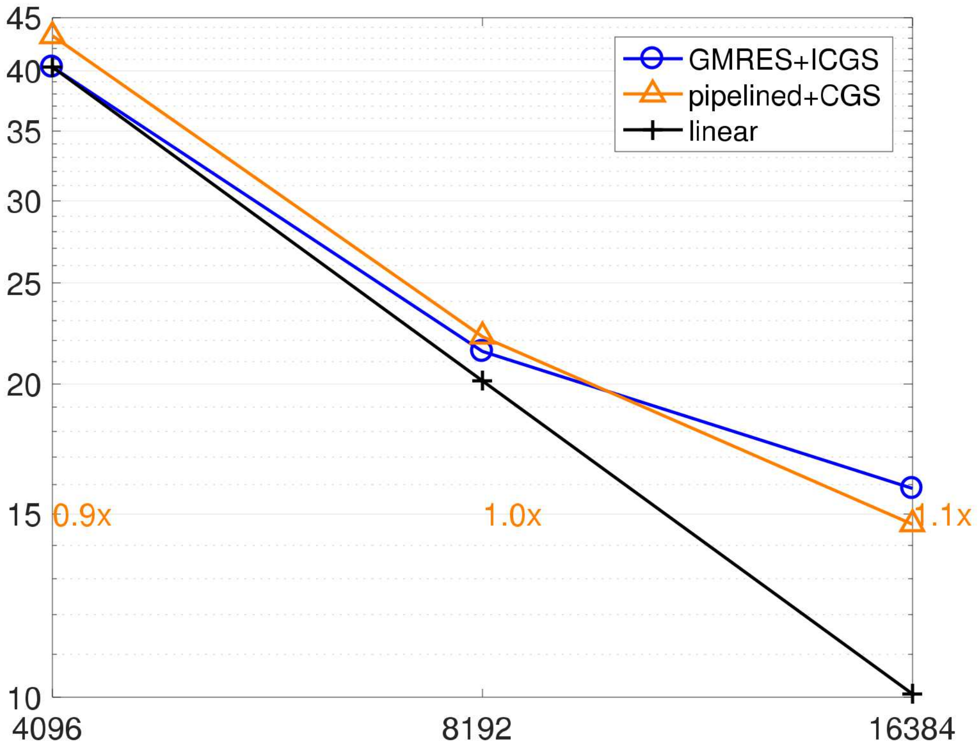
Image credit: Domino, Barone, & Bruner, 2018

Time to solution: Pressure system



- MueLu algebraic multigrid + (GMRES or Pipelined CA-GMRES w/ s=5)
- Newton (Ritz values from first s iterations as shifts) or monomial basis
- CholQR: Cholesky to implement TSQR; CholQR2: iterative refinement
- CGS2x: full reorthogonalization, 2 all-reduces / s steps; else 1 / s

Time to solution: Momentum system



- Symmetric Gauss-Seidel preconditioner + (GMRES, Pipelined CA-GMRES)
- Newton basis (Ritz values from first s iterations as shifts), s=5
- Pipelined (depth = 1)
- No reorthogonalization here (just happens to be what we measured)

Software engineering challenges

- Goal: Make new solvers available for users
  - Production-ready software, not research-ware
  - “Users”: App users, engineers, not solver experts
  - “Available”: via run-time choice (input deck)
- New solvers need new linear algebra ops
  - Esp. nonblocking dot products (using e.g., MPI_Iallreduce)
  - (Belos already designed for block orthogonalization (TSQR))
- Challenges
  - Belos must work for ANY linear algebra library, including users’
  - Trilinos must work for MPI_VERSION < 3 (no MPI_Iallreduce)

Belos' premature optimization

- Trilinos' iterative linear solvers live in the Belos package
- Belos works for any linear algebra (LA) implementation
  - Via polymorphism on Vectors & Linear Operators (matrix, prec)
  - Belos ignorant of LA details: knows only dot, norm, mat-vec, etc.
  - Users can give Belos their own LA types
- Belos uses compile-time polymorphism
  - Template parameters: Vector, Linear Operator
  - (C++) traits classes define fixed set of LA ops for Belos' solvers
  - Users w/ custom LA types must specialize traits classes
- Premature optimization; hinders adding solvers
  - Adding new ops to traits would break users' specializations
  - LA ops take much longer than virtual method call overhead
  - Run-time polymorphism → could add new ops w/ default impls

Linear algebra - specific solvers

- Belos' solvers historically had 1 implementation for all LA
- Now we want solvers that only work for specific LA (Tpetra)
- Problem: Access new solvers, w/out user code changes
  - Must plug solvers into Belos::SolverFactory (name → instance)
  - But SolverFactory is (was) agnostic of LA, just like (most) solvers!
- Solution: Inject custom LA-specific factory at run time (DII)
  - Specializations of SolverFactory can take run-time "custom factories"
  - Write new solvers to be "their own factories"
  - Tpetra also templated, but we fix set of allowed args at config time
  - → can write opaque "register ${SOLVER} w/ factory" function
  - Tpetra specialization of SolverFactory calls registration function
- Side benefit: No extra build time cost for new solvers

Nonblocking dot products

- MPI 3 (2012) added support for nonblocking collectives
  - MPI_Iallreduce: nonblocking version of MPI_Allreduce
- Trilinos' interface to nonblocking dot product:
  - auto request = idot(&result, x, y); // 🡐 MUST NOT BLOCK
  - /* … do other stuff ... Then */ request->wait();
- What if Trilinos was built with MPI < 3?
  - Capture (&result, x, y) in a closure (C++11 lambda)
  - Closure does blocking dot product; don't invoke closure yet
  - request->wait() just invokes the closure as std::function
- We write the solver once; it works for all MPI versions

Conclusions

- Deployed communication-avoiding & pipelined Krylov methods in Trilinos
- Improved solve performance in Nalu Wind by up to 1.5x
- Did so without breaking software backwards compatibility

Thank you!!

- Our NREL collaborators
- Chris Luchini (SNL) & other Nalu developers
- ECP PEEKS, for funding
  - This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nations exascale computing imperative.