SAND2018-2310C

Production Implementations of Pipelined & Communication-Avoiding Iterative Linear Solvers

Ichitaro Yamazaki (UTK) & Mark Hoemmen (SNL)

SIAM Parallel Processing, March 09, 2018

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

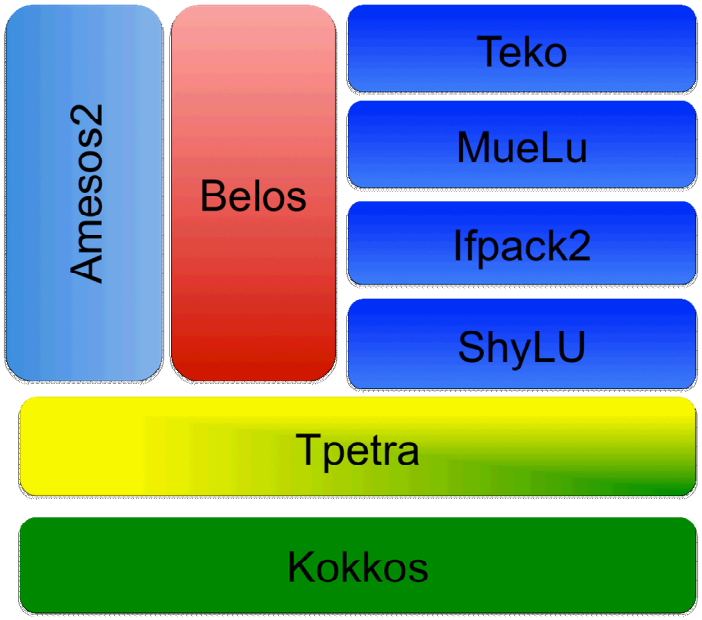
- Pipelined & communication-avoiding Krylov solvers can perform better, especially at large scales
- We want to make these solvers available in Trilinos
- This talk is about resulting software development challenges
- This talk is NOT about algorithms or their performance

What is Trilinos?

Sandia National Laboratories

- Parallel math libraries for science & engineering applications
  - Parallel programming models
  - Sparse linear algebra
  - Linear & nonlinear solvers
  - Optimization algorithms
  - Space & time discretizations
- github.com/trilinos/Trilinos
- Mostly C++ with some C and Fortran
- Many users inside & outside Sandia
- Must work on many different platforms
  - CPUs: x86, KNL, POWER, ARM, …
  - GPUs: NVIDIA, later AMD

Trilinos' linear solvers



- Sparse linear algebra (Tpetra)
  - Sparse graphs, (block) sparse matrices, dense vectors, parallel solve kernels, parallel communication & redistribution
- Iterative (Krylov) solvers (Belos)
  - CG, GMRES, TFQMR, recycling methods
- Sparse direct solvers (Amesos2)
- Algebraic iterative methods (Ifpack2)
  - Jacobi, SOR, polynomial, incomplete factorizations, additive Schwarz
- Algebraic multigrid (MueLu)
- Segregated block solvers (Teko)
- Direct+iterative preconditioners (ShyLU)

Goal: “Productionize” solvers

Sandia National Laboratories

- Make pipelined & communication-avoiding iterative linear solvers available for Trilinos users
- Must build & pass tests on all supported platforms
- Available to users via run-time choice (input deck)
  - Users don’t need to change their code
  - This means plugging solvers into existing “factories”
  - Not just dumping a class or function into the code repository

Software challenges

Sandia National Laboratories

- Trilinos' iterative linear solvers package makes it hard to add new linear algebra operations
- Trilinos must support older MPI versions that lack features needed for pipelined Krylov solvers
- MPI implementations by default may not make progress on nonblocking collectives, thus taking away benefit of pipelined Krylov methods

Trilinos' Belos package

Sandia National Laboratories

- Trilinos' iterative linear solvers live in the Belos package
- Belos was written in the mid-2000's, to support Anasazi (iterative eigensolvers package)
- Belos works for any linear algebra (LA) implementation
  - Belos defines a fixed set of ops on Vectors & LinearOperators (matrices & preconditioners): e.g., dot, norm, axpy, apply(X,Y)
  - Fixed set of LA ops defined via (C++) traits classes
  - Belos' solvers are templated on Vector & LinearOperator, & invoke LA ops by using traits classes directly
  - Belos provides specializations of traits for Trilinos' native LA types
  - Users who want Belos to work for their own LA types must write their own specializations of traits classes

Nonblocking dot breaks build?

Sandia National Laboratories

- Problem: What if we need new linear algebra (LA) ops?
  - Pipelined Krylov: Nonblocking dot product
  - Communication-avoiding Krylov: Matrix powers kernel, TSQR
- Can't add new LA ops to Belos without breaking build!
  - OK for Trilinos' native LA
    - Belos just changes its traits class specializations
  - NOT OK for users' own LA
    - Users have their own Belos traits specializations
    - They would need to change their code to add new LA ops

Traits classes too rigid here

Sandia National Laboratories

- Not a C++ problem, but a design problem
- Belos could have used run-time polymorphism (inheritance)
  - Would let us add new LA ops through “mix-in” classes without breaking backwards compatibility for libraries that lack them
  - Virtual method dispatch overhead tiny vs. MPI communication
- Belos chose compile-time polymorphism for historical reasons
  - Early C++ adopters for math codes worried about run-time overhead
  - C++ templates promised zero overhead
  - Belos in 1st generation of Trilinos packages that could use templates
  - Trilinos developers have more experience w/ templates now
- Applications use Belos, but want to access our new solvers
  - We don’t want to make them rewrite their code
  - Belos has useful features (like custom convergence tests)

Solution: LA-specific solvers

Sandia National Laboratories

- Belos' solvers before just had one implementation for all LA
- Previous slides show: Some solvers need LA-specific ops
- Users or third-party libraries may have optimized entire solvers for specific LA; Trilinos users want to access them
- Solution: extend Belos to support LA-specific solvers
  - Belos::SolverFactory already takes solver name at run time, & returns instance of the desired solver
  - NEW: Add interface to Belos::SolverFactory, that lets us or users inject a "custom solver factory" at run time
    - SolverFactory templated on Vector & LinearOperator ➔ custom factory is specific to those types
    - Custom solvers need only work for one LA, so they can code directly to that LA & use whatever ops they want
  - Solves a more general problem than pipelined & CA Krylov

MPI nonblocking all-reduce support

Sandia National Laboratories

- MPI 3 (2012) added support for nonblocking collectives
  - MPI_Iallreduce: nonblocking version of MPI_Allreduce
- Trilinos must support older MPI implementations
- 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, but doesn't invoke it yet
  - request->wait() just invokes the closure as std::function
  - Effect: idot(&result, x, y) really does not block
  - → We write the solver once; & it works for all MPI versions

MPI progress on nonblocking comm

Sandia National Laboratories

- "Nonblocking" ➔ return immediately after being called
- MPI could just defer all communication until MPI_Wait*
- Common case:
  - By default, MPI only sends/receives inside MPI functions
  - For asynchronous progress, must enable MPI_THREAD_MULTIPLE support & possibly also "progress thread" options at MPI build time
- Problems:
  - Users / sysadmins, not Trilinos, pick build MPI options
  - MPI_THREAD_MULTIPLE & progress thread incur overhead
  - Would we need to poll manually?
    - Paul Eller (UIUC): PETSc's implementation of pipelined Krylov needed manual MPI polling embedded inside the sparse matrix-vector multiply kernel in order for MPI_Iallreduce to be effective (!)

MPI progress a “work in progress”

Sandia National Laboratories

- MPI_THREAD_MULTIPLE overhead
  - Cost: message queue locking for MPI 2-sided (send, recv)
  - Trilinos’ sparse matrix-vector multiply uses 2-sided
  - Vendors recommended switching to MPI 1-sided (MPI_Win), since optimized implementations don’t use MPI message queues
  - Trilinos has plans to explore this, but not this year
- Manual polling? Impossible on GPUs, invasive in code
- Programming model mismatch
  - Pipelined Krylov methods really want a dataflow model
  - MPI historically resisted “active messages” (run a function asynchronously when I receive data from another process)
- Not sure what to do yet about this

Conclusions

- We want to deploy pipelined & communication-avoiding Krylov methods in Trilinos
  - Implementations exist now
  - We will put them in Trilinos this year
- Software challenges, because we want it to work for production users, instead of just hacking it in there
- We addressed some Belos & MPI – related challenges
- We need a better approach to asynchronous progress for nonblocking MPI operations