SAND2018-12003C

Deprecating & removing most of Tpetra's template parameters

Mark Hoemmen, on behalf of the Tpetra Team

Trilinos Users' Group meeting

October 2018

Quotes

“The easiest way to make Trilinos build faster would be to rename all .cpp files to .c, and fix the resulting syntax errors.” – Jed Brown

“**I actually enjoy complexity that’s empowering. If it challenges me, the complexity is very pleasant.** But sometimes I must deal with complexity that’s disempowering. The effort I invest to understand that complexity is tedious work. It doesn't add anything to my abilities.” – Ward Cunningham

“Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.” – Alan Perlis

TL;DR Plan to purge LO, GO, & Node

1. Fix solver packages so they can build with GO=int OFF
   - Epetra & Tpetra support w/ GO=int OFF?
   - Make sure all tests build & run (some disabled now)
2. Forbid CMake options enabling multiple GO or Node types
3. static_assert that template arguments are enabled types
4. Add “TpetraNew” replacement classes w/ desired interface
   - Reimplement existing “TpetraOld” classes using new ones
   - Concurrently add New tests & migrate Old tests
5. Concurrently introduce new interfaces in solver pkgs
6. Deprecate TpetraOld & replace w/ TpetraNew (how: TBD)
   - Make Tpetra{Old & New} interoperable?
   - Trilinos’ release schedule & policy changes? Version macros?

Why bother?

- Most Tpetra template parameters add very little value
  - LO, GO: If we only ever use 1 value of “class T”, just make it a typedef
  - Node: We understand heterogeneous computing better now
- Unneeded template parameters inflate build times & sizes
  - Can’t hide implementations of templated classes
  - Enable another GO type ➔ rebuild all Trilinos thru MueLu & Stokhos
  - Executable & libraries > 4 GB
  - Need 64-bit linkers, else linker crashes
  - 20-min single-file build times (now; in 2015 we crashed laptops)
- Mitigations make build system complex & brittle

What’s the point of templates?

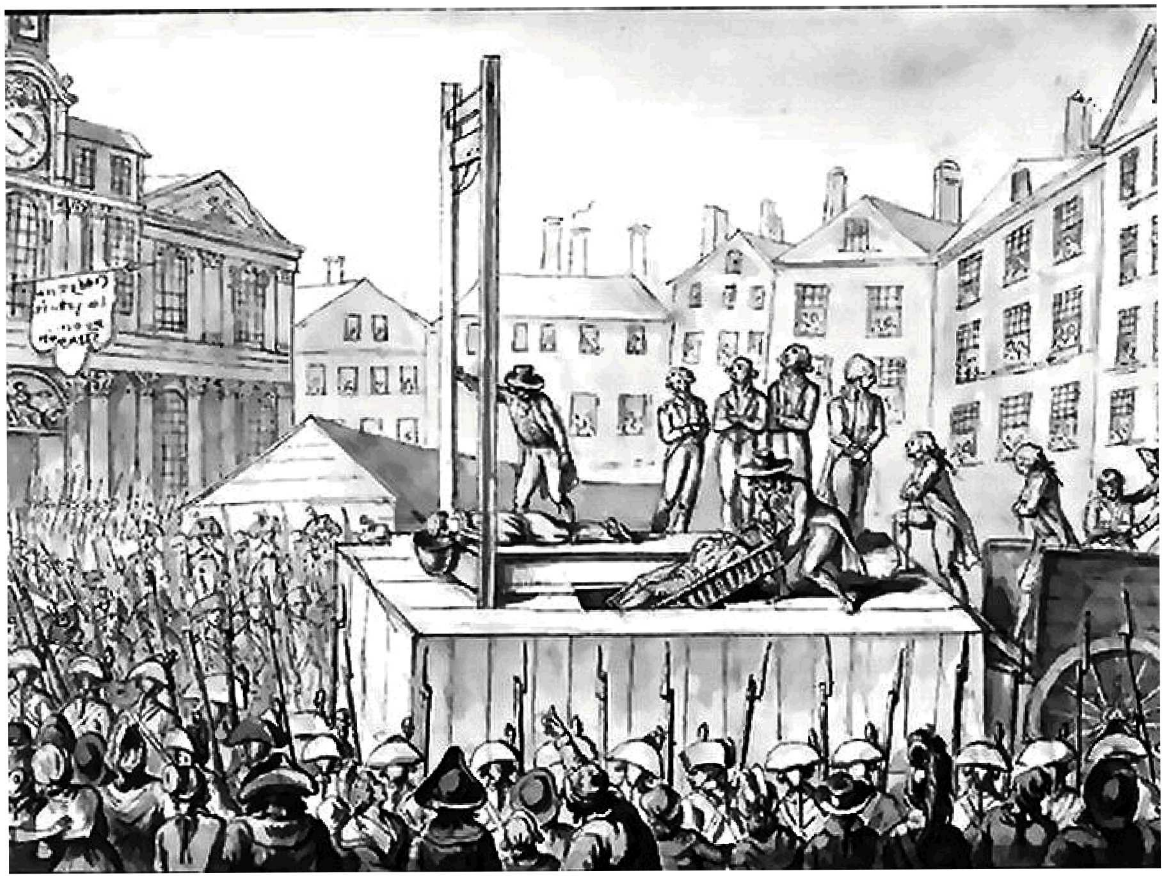
- Compile-time polymorphism
- Compile time: Can’t {use, afford} virtual method dispatch
- Polymorphism: Need “class T” to work for different T
  - Multiple T in the same build, like std::vector<T>
  - Otherwise, make T a configure-time typedef
- Is polymorphism really a requirement?
  - Or only a convenience for testing?
  - Or are we just confusing templates w/ configure-time options?
  - Ask this for each template parameter separately
- Open or closed polymorphism?
  - “Open”: Unbounded (tho SFINAE-constrained) set of valid types
  - “Closed”: Trilinos explicitly allows finite set of types

Templating adds build complexity

- Can’t hide implementations of templated classes
- Partial work-around: Explicit template instantiation (ETI)
  - Speeds up building code that uses Tpetra classes & functions
  - Makes build system obfuscated & brittle
    - e.g., {decl,def}.hpp, auto-gen’d .cpp, macros
    - Some of these needed so .cpp files build in <= 2 GB & < 5 mins
    - Remember when we tried to outsource a major ETI change?
  - Complicates app config & use (bjam)
  - Doesn’t solve “add another GO type, rebuild all of Trilinos” problem; in fact, makes this worse for Trilinos developers
  - Implications for open vs. closed polymorphism…

Trilinos: Closed polymorphism

- Trilinos' ETI requires closed polymorphism
  - Must know set of types at configure time, for which you want to instantiate classes & functions
- Trilinos has been depending more on configure-time knowledge of types, even w/out ETI
  - Packages use "ETI" macros to define tests, even w/ ETI OFF
  - (We've purged hacky tests that include _def &/or require ETI OFF)
  - "Solver factory" dependency injection & inversion (DII)
- "extern template" ETI approach relaxes this, but:
  - Only get build time benefit for closed set; DII too
  - Can't hide implementation (e.g., users see header files)
  - Christian & I first tried this in kokkos-kernels, but k-k & even Kokkos (for ViewCopy & ViewFill) starting to favor an "ETI-only" approach



Put Tpetra's template parameters on trial

Look one-by-one at Scalar, LocalOrdinal, GlobalOrdinal, & Node

Scalar & Packet live: Both add value

- Mixed-precision arithmetic (original Tpetra design goal)
  - Some features removed over time to reduce build complexity
    - CrsMatrix's templated local sparse mat-vec & local triangular solve
    - Ifpack2 Container templated apply
  - Still possible with Operator interface (matrix Scalar != vector Scalar)
- Interesting features like automatic differentiation (AD)
- Large apps use multiple Scalar types in 1 build
  - double & complex<double>
  - double & AD
- DistObject's "Packet" a key part of Petra Object Model
  - Decouples boundary exchange implementation from data structure

LO & GO die: Little value, high cost

- LO: LocalOrdinal (type of local indices)
  - Stored in sparse matrices; used in computational kernels
  - Flexibility could pay: Save memory/time vs. large single-process solve
  - But: 64-bit LO doesn’t build (1 externally contributed PR touched all of Trilinos, except for Kokkos (where it matters), but added no tests)
  - Interface could have hidden storage representation mostly from users, just like Tpetra did once w/ CRS row offset type
- GO: GlobalOrdinal (type of global indices)
  - Why would you want anything other than int64_t ?
  - If you store & use GO a lot, you’re slow anyway
- Mixing multiple (LO,GO) in one build?
  - Old Ifpack2 Container example: Intraprocess domain decomp
  - But: GO causes most build complexity & outstanding issues

Node dies: Wrong model, high cost

- Original heterogeneity model: Per object
  - Node determines memory & execution spaces (anachronism)
  - Rarely used (Stokhos LDRD, CASL VERA), mainly thru clone()
  - HybridPlatform (per-process choice) never used, hard to load balance
  - True concurrency never tested & likely broken due to MPI
  - Node: leftover from Chris Baker's Kokkos 1.0
- Proposed new model: Per kernel, not per object
  - Choose default Kokkos exec. (& memory) space(s) at config time
  - Users may ask to view an object's data in a given memory space
  - Users may give each kernel an optional execution space instance
    - May use enum / wrapper, to avoid exposing implementation

Must break backwards compatibility

- Can’t change template parameters w/out breakage
- We tried using type aliases to hide/ease changes
  - Make e.g., Tpetra::Vector an alias to Tpetra::Classes::Vector
  - Use C++11 parameter pack & fancy deduction to let us deprecate & remove template parameters w/out changes to user code

```
template<class ... Args>
using Vector =
  typename FourArg<Classes::Vector, Args...>::type;
```

~ 200 LOC, fancy

  - Problem: Partial specialization doesn’t take type aliases
    - Breaks Belos::MultiVecTraits & other traits classes
  - Technique thus not a complete solution; can’t hide “Classes::$CLASS"

Remaining questions

- Epetra & Tpetra both enabled in Amesos2 & Xpetra?
  - Amesos2 & Xpetra both define wrappers / adapters for {E,T}petra
  - Wrappers take same template parameters as Tpetra
  - Wrappers do no index conversion: Wrappers' GO == Tpetra's GO
  - Problem: Epetra requires GO=int in the wrapper
  - Do users ever need to enable Epetra, & Tpetra with GO != int?
    - If so, must change wrappers to do index conversion
    - MueLu willing to exclude this
    - 1 app needs Amesos2 w/ both Epetra & Tpetra enabled; still waiting to hear whether they need Tpetra w/ GO != int in this case
- How to manage backwards compatibility?
  - Need Trilinos to state & follow a release & deprecation policy
  - If not, Tpetra will need to imitate Kokkos in defining its own policy & release cycle

Questions?

Concurrent build times/sizes efforts

- New forward declaration headers
  - Help apps get advantages of fwd decls, but maintain backwards compatibility (vs. just fwd-decl’ing themselves – an issue w/ deprecation of the bool “classic” template parameter)
- Purge unneeded header includes
  - Historically has broken downstream apps e.g., Albany
- Distributor
  - Hide underlying implementation
  - Planned anyway as part of comm/comput. overlap & better MPI/{CUDA,OpenMP} interactions
- General code discipline (everyone’s job!)
  - DefaultPlatform deprecation & removal
  - Avoid exposing TPL header includes