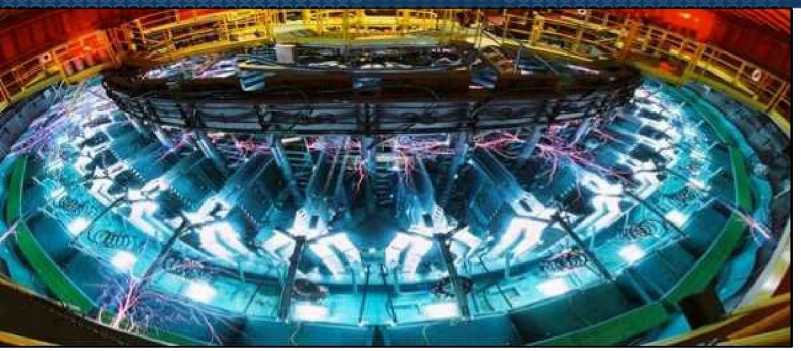
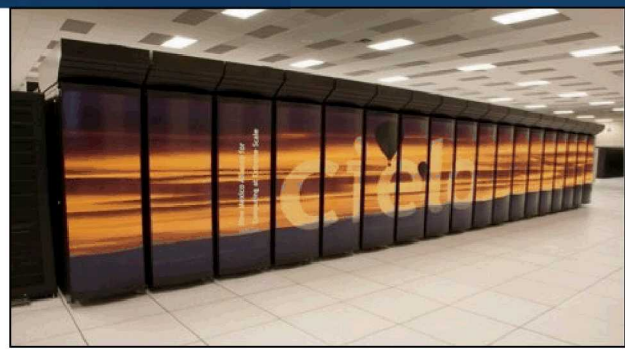
SAND2019-5113PE





Build Systems and Package Management for Modern C++

Jeremiah Wilke
Scalable Modeling and Analysis, Sandia National Labs, Livermore CA
Kokkos User's Group Meeting

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Harvard Business Review: Behavioral Economists #1 Reasons People Make Bad Decisions

- ***Not anticipating unexpected events***
  - Trilinos is great. Everyone wants to use it – but often single packages as a TPL
- ***Indecisiveness***
  - The DOE – even with ECP – has not required uniformity in best practices
- ***Remaining locked in the past***
  - We have modern CMake now
- ***Having no strategic alignment***
  - Strategy is make the code as easy to use and develop as possible
- ***Over-dependence (and cyclic dependence)***
- ***Isolation:*** Components separated so long they can't be brought together
- ***Lack of technical depth***
  - Is modern CMake widely understood?

Modern CMake wants a clean separation of 'building' and 'using' libraries

Sandia National Laboratories

- All options should be applied specifically to TARGETS (libs, exes)
  - No more directly modifying CXXFLAGS
  - No more global setting include directories and compiler flags
- All include directories and compiler flags should be clearly defined as:
  - PUBLIC: Flag needed to build Kokkos and needed downstream to use Kokkos
    - Kokkos headers
    - Flags like –fopenmp or CUDA flags needed for the backend
  - PRIVATE: Flag only needed to build Kokkos (not needed to use)
    - Certain warning flags
    - Optimization flags
    - C++ standard flags???
  - INTERFACE: Header-only libraries
    - Ignore for now

Modern CMake wants a clean separation of 'building' and 'using' libraries

- All options should be applied specifically to TARGETS (libs, exes)
- Compiler *features* are not activated with explicit flags
  - e.g. TARGET_COMPILE_FEATURES(target, std_cxx_14)
- All TPLS are CMake targets – even if imported from autotools
- Should work for transitive dependencies
  - Kokkos -> KokkosKernels -> MyMathLib
  - MyMathLib should get Kokkos flags *without* Kokkos Kernels doing any work
  - MyMathLib should get Kokkos flags without knowing Kokkos exists
- A few things missing in CMake
  - CUDA/CUDACXX needs a feature list just like C++
  - Needs intermediate C++ standards as a feature
  - *TARGET_COMPILE_FEATURES needs to detect ABI problems*
- ***Thought experiment:*** Should C++ standard flags be PUBLIC? PRIVATE? Something new like ABI_PRIVATE?

Complexity Measured in Lines of Code

- Kokkos supports 6 different ways of building and installing (or not installing)
  - Trilinos (Tribits)
  - Standalone – separate libs or single lib
  - In-tree build – separate libs or single lib
  - Raw Makefile

	Today	No More Trilinos! TPL Only!	And No Makefile! Must use Cmake!	And Install Single Lib Only
Tribits Wrappers	500 LOC			
Trilinos Cmake Options	600 LOC			
Makefile-CMake Handoff	500 LOC	500		
Makefile.kokkos	1200 LOC	1200		
Find TPLs CMake	400 LOC	400	400	400
Actual core CMake	1200 LOC	1200	1200	1000
Total	4400 LOC	3300 LOC	1600 LOC	1400 LOC

What should your CMake file look like for ***using*** Kokkos?

Sandia National Laboratories

```
FIND_PACKAGE(Kokkos REQUIRED)
ADD_LIBRARY(target ${SOURCES})
TARGET_LINK_LIBRARIES(target PUBLIC Kokkos::kokkos)
```

I need Kokkos to build – and anyone using my API needs Kokkos

```
FIND_PACKAGE(Kokkos REQUIRED)
ADD_LIBRARY(target ${SOURCES})
TARGET_LINK_LIBRARIES(target PRIVATE Kokkos::kokkos)
```

I need Kokkos to build – but using my API does not require Kokkos

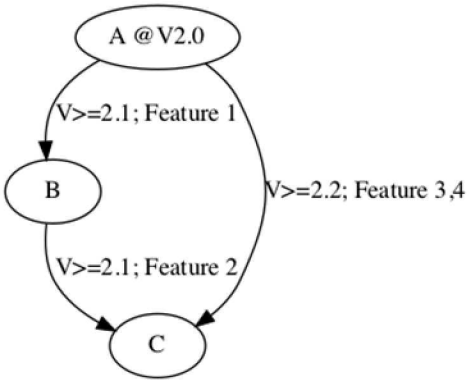
```
FIND_PACKAGE(Kokkos REQUIRED)
ADD_LIBRARY(target ${SOURCES})
TARGET_LINK_KOKKOS(target)
```

The likely near-term solution to minimize complexity…

TARGET_LINK_KOKKOS could in the future take arbitrary assertions

Make easy problems slightly less easy if it makes really hard problems tractable

- What are the hard problems for software stacks with dozens of dependent packages?
  - Developing against *release* versions of packages: EASY
  - Developing against *feature* branches of packages: HARD
- Versioning
  - I export a version number and compatibility model
  - I depend on packages with a particular version (=, >=, <=)
  - Develop features depend on develop branches or commits
- Concretization:
  - Agreeing on a dependence (i.e. Kokkos) that satisfies all dependent packages
  - Package C concretizes to @2.2 with Features 2,3,4
- Patching
  - I want new features, but the new version breaks my code
  - I want new features, but need to cherry-pick commits



Spack: Package Management for HPC

- Packages define a 'package.py' that says:
  - What packages will I use?
  - How do I build?
  - These will depend on variants (features)
- Example: Ifpack2 if it were a standalone library
  - spack install ifpack2@develop +openmp %gcc@7.2.0

```
Variants:
    build_type [RelWithDebInfo]    Debug, Release,          CMake build type
                                   RelWithDebInfo,
                                   MinSizeRel
    complex_double [off]           True, False              ETI complex double
    cuda [off]                     True, False              enable Cuda backend
    double [on]                    True, False              ETI doubles
    float [off]                    True, False              ETI float
    openmp [off]                   True, False              enable OpenMP backend
    serial [on]                    True, False              enable Serial backend
 Installation Phases:
    cmake    build    install
Build Dependencies:
    cmake  kokkos-kernels
```

Spack Ifpack2 Example Detail

- Spack concretizes a dependency graph
  - Dependency hash is unique ID for version/variants/compiler

- Spack builds packages in dependency order
- At each step, CMake used to build and install
  - What flags do I need to build?
  - What flags do dependencies need to use me?
- I can fine-tune dependencies by:
  - Creating ”versions” of a project by cherry-picking feature commits
  - Patching dependencies
- Dependencies can be “resources” for in-tree builds

Using nvcc_wrapper with Spack

- Install nvcc_wrapper for your underlying compiler
  - spack install nvcc_wrapper %gcc@7.2.0-kokkos
- Add nvcc_wrapper to the Spack compilers.yaml with meaningful name
  - Suggested style is calling it gcc@7.2.0-kokkos
- Proceed with workflow with as before
  - spack install ifpack@develop +cuda %gcc@7.2.0-kokkos

"Alpha" version of new Cmake and Spack available today for the brave

Hope On The Horizon For Building Kokkos as TPL with CMake and Spack?

Sandia National Laboratories

"The struggle itself towards the heights is enough to fill a man's heart. One must imagine **Sisyphus happy**."

-Albert Camus, The Myth of Sisyphus

We still have to support all 6 modes for now… How do we do it?

Sandia National Laboratories

- No calls to raw Tribits functions, single set of CMake files
  - Wrapper functions used that support standalone or Tribits
  - Too many IF(…) statements
  - Too many test configurations
  - Fragile is good when broken things look broken
  - Fragile is bad (i.e. parallel CMake's) if silent errors
- Don't trust Tribits to propagate flags (transitive dependencies) when installed
  - Clang-CUDA doesn't work today
- Makefile.kokkos logic only used for config header
  - All the bad things with CXXFLAGS will only affect Makefile users

Ultimate goal is maximize Kokkos usability: is this possible supporting 6 build methods?

Sandia National Laboratories

"A man devoid of hope and conscious of being so has ceased to belong to the future."

-Albert Camus, The Myth of Sisyphus

"I have achieved the summit only to watch the ball roll down the other side, with all the work still remaining before me."

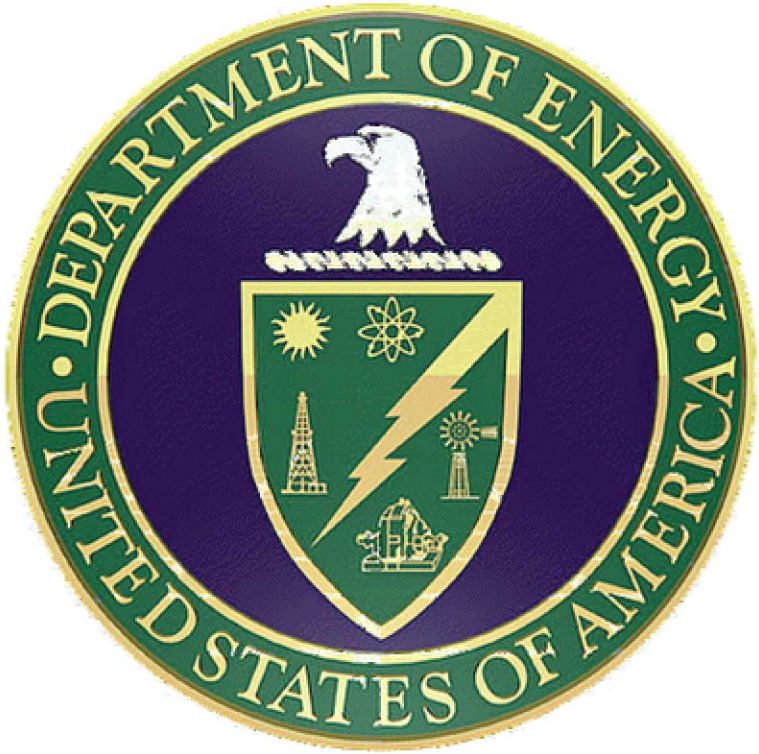
-Jeremiah Wilke

Harvard Business Review: Behavioral Economists #1 Reasons People Make Bad Decisions

- ***Not anticipating unexpected events***
  - Small, agile projects can more rapidly adapt to customers
- ***Indecisiveness***
  - If a little pain now makes long-term future better, we need to act
  - I would rather spend my time forcing, ahem, teaching people to use CMake than maintain 6 different build methods
- ***Remaining locked in the past***
  - Keep Makefiles for 2.X. Should a version 3.0 shed the baggage?
  - The only acceptable context going forward for Makefiles is *mini-apps!*
- ***Having no strategic alignment***
  - Strategy is make the code as easy to use as possible. 6 different build methods all with their own documentation isn't good for developers, users, or collaboration
- ***Isolation:***
  - Hard to bring tools from Kokkos ecosystem together if different build worlds
- ***Lack of technical depth***
  - SIMD types/layouts/executors are hard. CMake is not (when it works)

Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.



National Nuclear Security Administration

Sandia National Laboratories