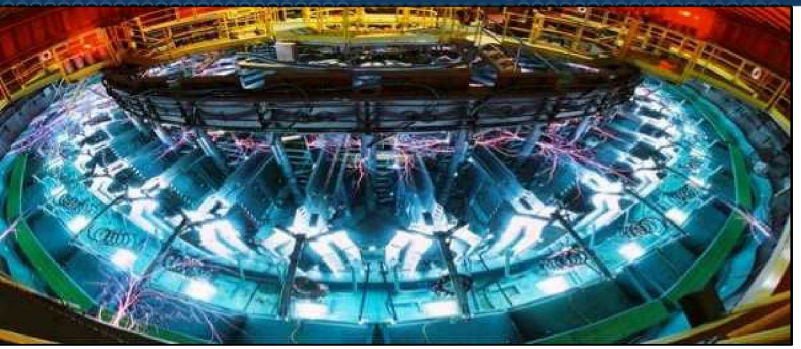
SAND2020-0654PE



Kokkos CMake: Build Systems and for Modern C++

Jeremiah Wilke
Scalable Modeling and Analysis, Sandia National Labs, Livermore CA
Kokkos Bootcamp, Santa Fe, NM
1/17/2019

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***
- ***Indecisiveness***
- ***Remaining locked in the past***
- ***Having no strategic alignment***
- ***Over-dependence (and cyclic dependence)***
- ***Isolation:*** Components separated so long they can't be brought together
- ***Lack of technical depth***

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

Sandia National Laboratories

- CMake 3 (first "modern" version) released June 2014
  - Clean separation of building and using (targets and properties) has been recommended method since release
- All options should be applied specifically to TARGETS (libs, exes)
  - No more directly modifying CMAKE_CXX_FLAGS
  - No more global setting include directories and compiler flags
  - Your compiler/linker flags should be *specific* and *exact* to an individual library
- 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
    - Minimum C++ standards
  - PRIVATE: Flag only needed to build Kokkos (not needed to use)
    - Certain warning flags
    - Certain optimization flags

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

A single CMake function should populate build with all the necessary flags to build *correctly* and all the optimization/architecture flags to improve *performance*

```
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)
```

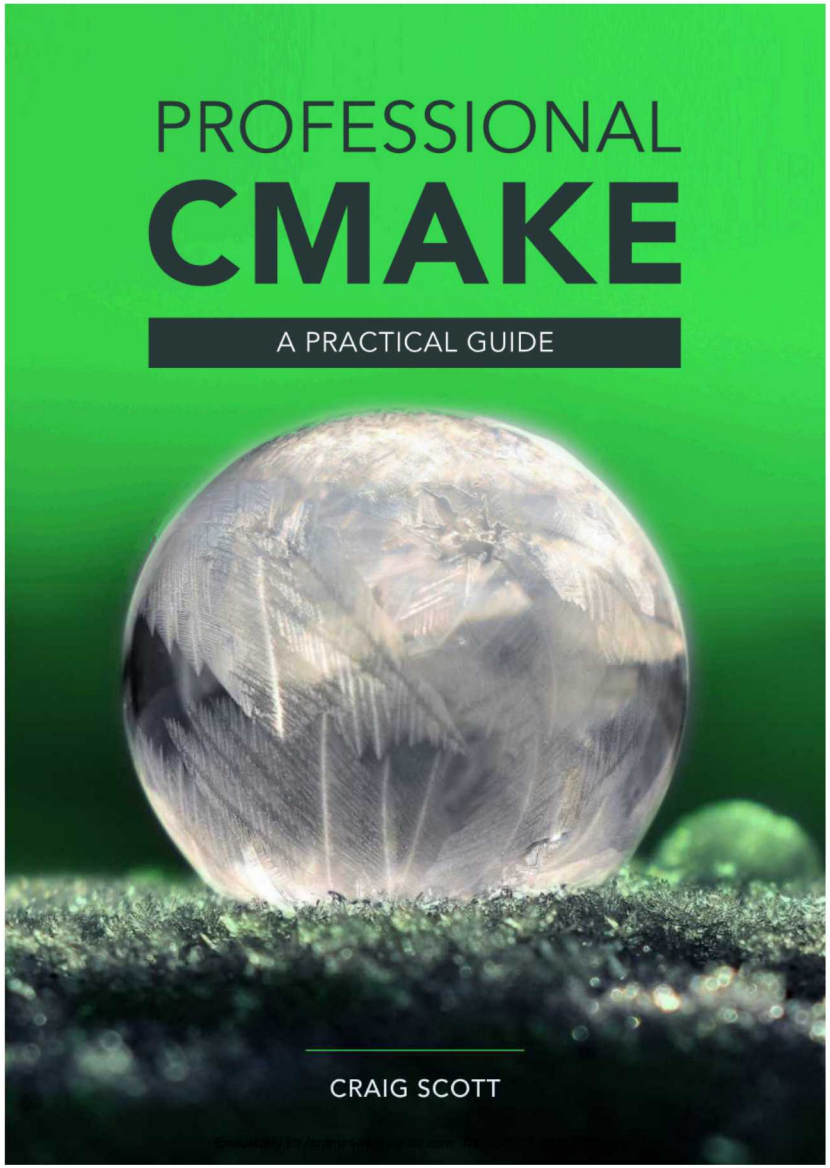
I need Kokkos to build – but using my API does not require Kokkos

```
KOKKOS_CHECK(
  DEVICES CUDA OPENMP
  OPTIONS CUDA_RELOCATABLE_DEVICE_CODE
  ARCH VOLTA70
)
```

Assert that the Kokkos configuration found meets expectations

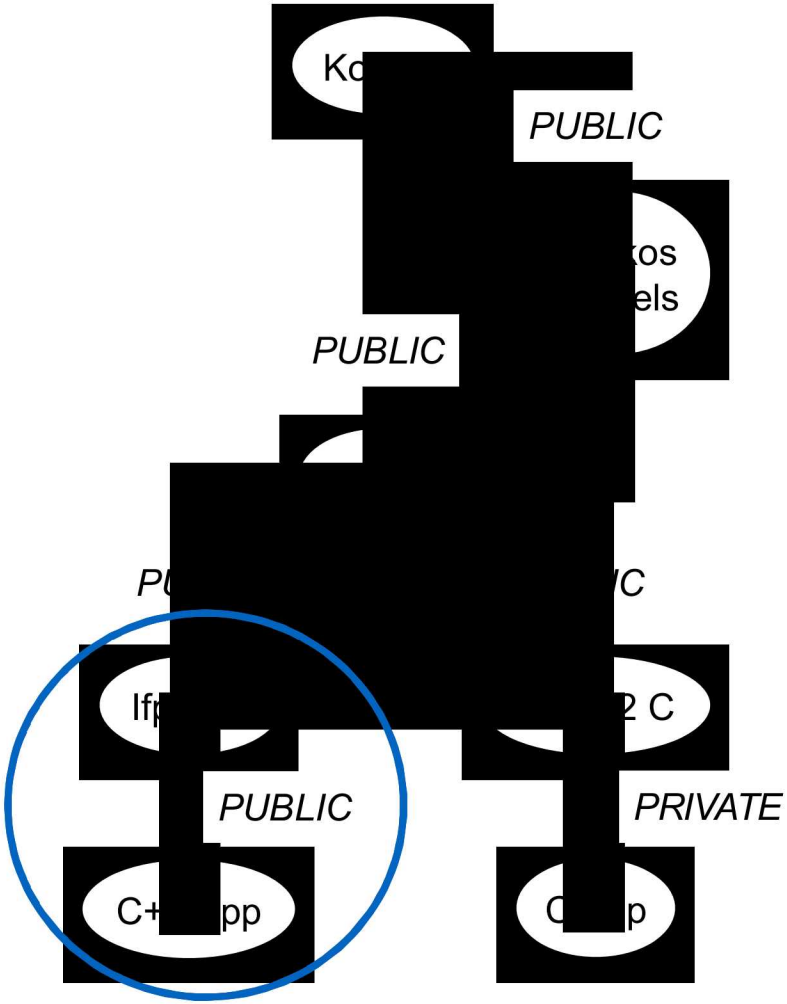
Installed Kokkos: cmake –DKokkos_ROOT=<PREFIX>

In-tree Kokkos: add_subdirectory(kokkos)



Best 40 dollars you will spend

Building and using makes "smaller" interfaces between libraries, solves transitive dependencies



Application should only know about its direct dependencies

TARGET_LINK_LIBRARIES(Ifpack2) makes C++ App depend transitively on Kokkos flags (PUBLIC)

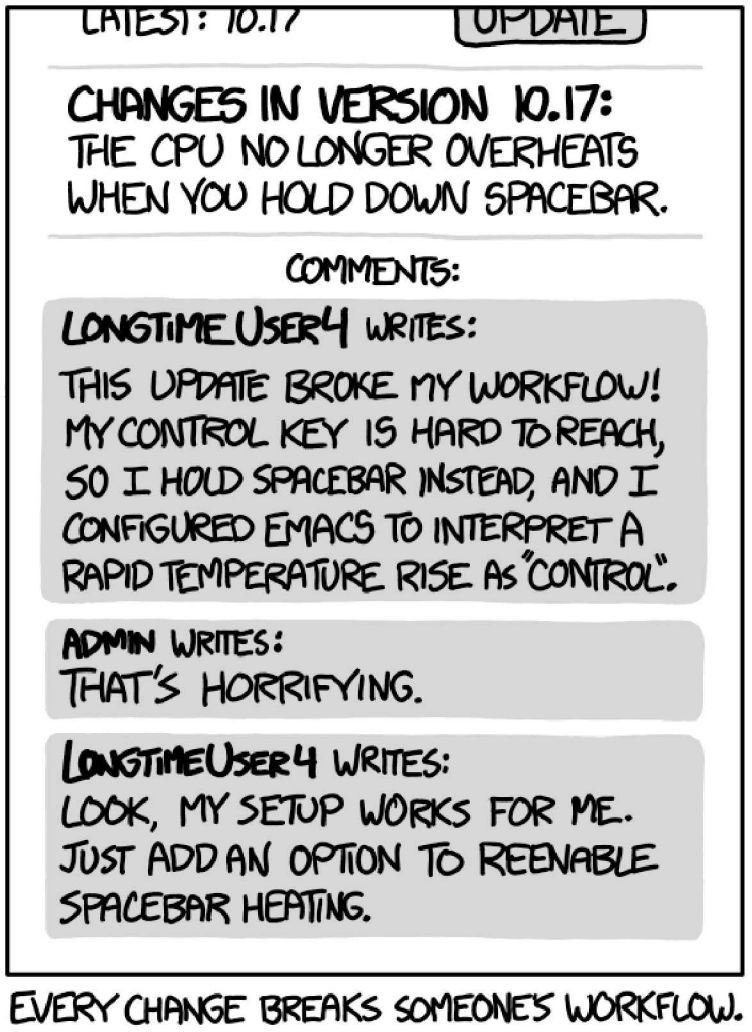
Automake requires collecting and forwarding, e.g.

KokkosKernels_CXX_FLAGS = $(LOCAL_CXX_FLAGS) + $(Kokkos_CXX_FLAGS)

TARGET_LINK_LIBRARIES(Ifpack2_C) does not make C App depend transitively on Kokkos flags (PRIVATE)

Modern CMake (targets and properties) is much more robust than CMake 2

Hyrum's Law:
With a sufficient number of users it does not matter what your public API is.
All observable behaviors will be depended on by somebody!



Modern CMake (targets and properties) is much more robust than CMake 2

- Modern CMake shrinks the interface size between two libraries from many, many CMake variables to a single function call
  - Gives Kokkos the freedom to make interface changes without breaking downstream codes
- Modern CMake enables the use of so-called *generator expressions* that can implement interesting build logic
  - Example: Library mixing C++/C/Fortan only adds Kokkos flags to C++ files
  - Example: Change preprocessor defines based on build type (debug/release) to add extra checks/optimizations flags
- Modern CMake helps resolves conflicting compiler features/options
  - Kokkos C++ standard vs. App C++ standard
  - Optimization/warning flags used to build Kokkos
- Avoid CMake Gotchas: Type-o's are just empty variables
  - TARGET_LINK_LIBRARIES(Kokkos::kokkos) will crash if Kokkos hasn't been correctly found or there are type-o's

Modern CMake will enable best usage of modern C++ going forward to '20 and '23

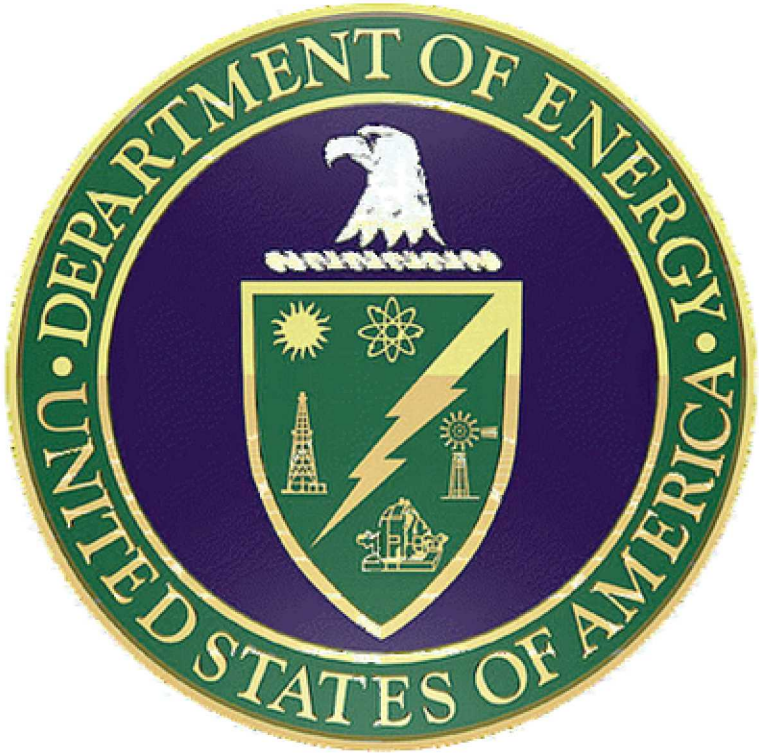
- CMake is de facto "standard" build system for modern C++ going forward
  - If you believe Reddit and CppCon
- Precompiled headers introduced into most recent CMake release
  - No changes to your build system if Kokkos starts using precompiled headers.
  - Single call to TARGET_LINK_LIBRARIES(Kokkos::kokkos) handles all the complexity of the Kokkos interface
- C++ modules will (or won't) be coming soon
  - (Probably) no changes to your project build system if Kokkos starts using modules (code changes, though)
  - Single call to TARGET_LINK_LIBRARIES(Kokkos::kokkos) would handle all the complexity of the Kokkos interface and module dependency graph
- Compatible interface properties
  - CMake allows adding arbitrary interface Boolean/String property checks
  - e.g. POSITION_INDEPENDENT_CODE is builtin

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

- ***Not anticipating unexpected events***
  - TARGET_LINK_LIBRARIES is smallest possible interface that allows Kokkos to be most agile without breaking behaviors users depend on (Hyrum's Law)
- ***Indecisiveness***
  - CMake has well-defined best practices (Professional CMake, Craig Scott)
- ***Remaining locked in the past***
  - Good luck with modules/precompiled headers using Automake
- ***Having no strategic alignment***
  - Does full stack modern C++ require program commitment to modern CMake?
- ***Isolation:***
  - Hard to bring tools from Kokkos ecosystem together if different build worlds
- ***Lack of technical depth***
  - CMake has a learning curve that is steeper than raw Makefiles
  - Make easy problems a bit more difficult if it makes hard problems tractable

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