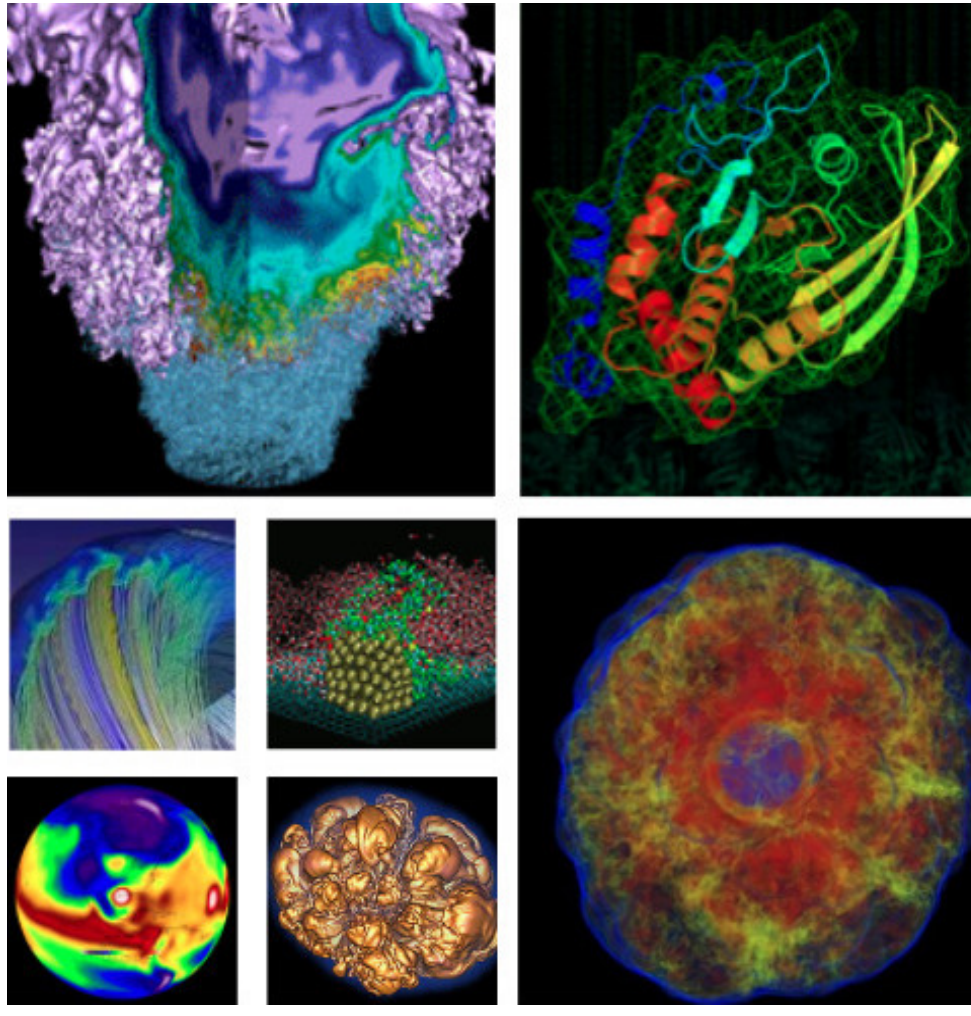
Build Systems
GPU Offloading



National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Jonathan R. Madsen, PhD

✉ jrmadsen@lbl.gov

National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

July 3, 2019

1. Manually Building Code for the Different Offloading Models (Makefiles)
   - OpenMP
   - OpenACC
   - CUDA
   - Kokkos
2. Using CMake to Generate Build System

OpenMP

- Availability: Clang, GCC
- Relevant Compiler Flags:
  - `-fopenmp -fopenmp-targets=nvptx64-nvidia-cuda`
  - `--cuda-path=$CUDA_HOME`

OpenACC

- Availability: PGI
- Relevant Compiler Flags:
  - `-acc -ta=tesla:cc70` ⇒ OpenACC flag and build for Volta architecture
  - `-ta=tesla:managed` ⇒ use managed memory
  - `-Minfo=accel` ⇒ print acceleration report
  - `-Mlarge_arrays` ⇒ allow using large arrays (¿2GB)

CUDA

NVCC

- NVIDIA provided CUDA compiler
- Essentially a C++ compiler that recognizes CUDA syntax and implicitly includes linking to CUDA libraries and header paths

Clang

- Current release (7.0.0) supports CUDA 7.0 through 9.2
- If you need CUDA 10 support, use clang built from r342924 or newer
- Recognizes files with extension `.cu` as CUDA
- Relevant compiler flags:
  - `-x cuda` ⇒ compile as CUDA (*e.g.*, `.cpp` as CUDA)
  - `--cuda-gpu-arch=<GPU arch>` ⇒ compute capability, *e.g.*, `sm_35`
  - `--cuda-path=/path/to/cuda` ⇒ if non-standard install
  - `-L/path/to/cuda/<lib64 or lib>` ⇒ path to CUDA SDK libs
  - `-lcudart_static -ldl -lrt -pthread` ⇒ req. libraries for linking

Kokkos

nvcc_wrapper

- Kokkos provided wrapper to NVCC and a host compiler (default is GNU)
- Run `nvcc_wrapper --help`

CMake Support

- Kokkos documentation claims "Kokkos supports being build as part of a CMake applications. An example can be found in example/cmake_build"
- I found the existing build system is buggy once I started toggling options, *e.g.*, `-DKOKKOS_ENABLE_CUDA=ON` ...
  - After speaking with the Kokkos developers, this is a known issue and support will be improved in Kokkos 3.0 release

CMake

- CMake (Cross-Platform Make) does not replace `Makefile`
- CMake is a build-system generator and the default settings *generates Makefiles* $\Rightarrow$ running `cmake` is analogous to running `configure`
  - Supports more than Makefiles $\rightarrow$ can generate Xcode projects, Visual Studio projects, Ninja, NMake, integrate with IDEs
- CMake is part of a larger set of tools provided by Kitware
- When reporting the competition results, you will be implicitly using with two of them: CTest and CDash
  - CTest will execute the build commands and run the benchmark problem and push the logs to CDash
  - NERSC hosts a CDash dashboard at cdash.nersc.gov (Project: gpu-for-science-day-july-2019)
- NERSC users are free to utilize this dashboard. If interested, contact me: jrmadsen@lbl.gov to create a project dashboard

Benefits

- Supports C, C++ , Fortran, and CUDA as "first-class languages"
- "Knows" optimization flags for a huge variety of compilers
- Provides several scripts for determining the required include directories, compile flags, libraries, link flags, etc. for a large number of community packages, *e.g.,* OpenMP, OpenACC, MPI, BLAS, Boost, OpenCL, OpenGL, Python, Qt, Git, Threads, GTest, HDF5, SWIG, X11, Gnuplot, LATEX, Matlab, etc.
  - `find_package(OpenMP)`
  - See full list of build-in package discovery at `<install-path>/share/cmake-X.YY/Modules`
- Significant benefit for large projects that optionally support GPU off-loading
- Provides concept of "INTERFACE" libraries that store compiler flags, include directories, link libraries, definitions, etc. in a single entity that can just be "linked" to

```
1 cmake_minimum_required(VERSION 3.10 FATAL_ERROR)
2
3 project(Gpu4Science LANGUAGES CXX CUDA VERSION 0.0.1)
4
5 # create "gpu4sci-host" executable
6 add_executable(gpu-host gpu4sci.cpp)
7 # create "gpu4sci-cuda" executable
8 add_executable(gpu4sci-cuda gpu4sci.cu)
```

```
$ ls
CMakeLists.txt  gpu4sci.cpp  gpu4sci.cu
$ mkdir build && cd build
$ cmake .. -G Ninja
-- The CXX compiler identification is GNU 8.3.0
-- The CUDA compiler identification is NVIDIA 10.1.168
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc -- works
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gpu4sci/build
```

```
$ ninja -v
[1/5] /usr/bin/c++     -MD -MT CMakeFiles/gpu-host.dir/gpu4sci.cpp.o -MF
    CMakeFiles/gpu-host.dir/gpu4sci.cpp.o.d -o CMakeFiles/gpu-host.dir/
    gpu4sci.cpp.o -c ../gpu4sci.cpp
[2/5] : && /usr/bin/c++     CMakeFiles/gpu-host.dir/gpu4sci.cpp.o  -o gpu-host
       && :
[3/5] /usr/local/cuda/bin/nvcc     -x cu -c ../gpu4sci.cu -o CMakeFiles/
    gpu4sci-cuda.dir/gpu4sci.cu.o && /usr/local/cuda/bin/nvcc     -x cu -M
    ../gpu4sci.cu -MT CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o -o CMakeFiles/
    gpu4sci-cuda.dir/gpu4sci.cu.o.d
[4/5] /usr/local/cuda/bin/nvcc   -Xcompiler=-fPIC -Wno-deprecated-gpu-targets
    -shared -dlink CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o -o CMakeFiles/
    gpu4sci-cuda.dir/cmake_device_link.o
[5/5] : && /usr/bin/g++  CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o CMakeFiles/
    gpu4sci-cuda.dir/cmake_device_link.o -o gpu4sci-cuda   -L"/usr/local/cuda
    /targets/x86_64-linux/lib/stubs" -L"/usr/local/cuda/targets/x86_64-linux/
    lib" -lcudadevrt -lcudart_static -lrt -lpthread -ldl && :
```

```
$ rm -rf *
$ export CXX=clang++
$ cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_STANDARD=17 -
    DCMAKE_CUDA_STANDARD=11 ..
-- The CXX compiler identification is Clang 6.0.0
-- The CUDA compiler identification is NVIDIA 10.1.168
-- Check for working CXX compiler: /usr/bin/clang++
-- Check for working CXX compiler: /usr/bin/clang++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc -- works
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gpu4sci/build
```

```
$ make VERBOSE=1
[ 20%] Building CXX object CMakeFiles/gpu-host.dir/gpu4sci.cpp.o
/usr/bin/clang++    -O3 -DNDEBUG   -std=gnu++17 -o CMakeFiles/gpu-host.dir/gpu4sci.cpp.o
      -c /home/gpu4sci/gpu4sci.cpp

[ 40%] Linking CXX executable gpu-host
/usr/bin/clang++  -O3 -DNDEBUG   CMakeFiles/gpu-host.dir/gpu4sci.cpp.o  -o gpu-host

[ 60%] Building CUDA object CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o
/usr/local/cuda/bin/nvcc    -O3 -DNDEBUG   -std=c++11 -x cu -c /home/gpu4sci/gpu4sci.cu -
      o CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o

[ 80%] Linking CUDA device code CMakeFiles/gpu4sci-cuda.dir/cmake_device_link.o
/usr/local/cuda/bin/nvcc  -O3 -DNDEBUG -Xcompiler=-fPIC -Wno-deprecated-gpu-targets -
      shared -dlink CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o -o CMakeFiles/gpu4sci-cuda.
      dir/cmake_device_link.o

[100%] Linking CUDA executable gpu4sci-cuda
/usr/bin/g++  CMakeFiles/gpu4sci-cuda.dir/gpu4sci.cu.o CMakeFiles/gpu4sci-cuda.dir/
      cmake_device_link.o -o gpu4sci-cuda  -L"/usr/local/cuda/targets/x86_64-linux/lib/
      stubs" -L"/usr/local/cuda/targets/x86_64-linux/lib" -lcudadevrt -lcudart_static -lrt
       -lpthread -ldl
```

- CMake makes a distinction between the source directory and the build directory
- There are two types of values: local and cache in CMake
  - Local values do not persist between invocations of CMake
  - Cache values DO persist between invocations of CMake and are stored in `CMakeCache.txt`
    - Certain values such as which compilers to use are stored in cache. If you want to change compilers, remove CMakeCache.txt!
- Many projects use CMake poorly $\Rightarrow$ this is mostly due to poor documentation
- Register for CMake training at NERSC to learn more!

- See docs.nersc.gov/programming/build-tools/#cmake
- Three features can greatly simplify your offloading build system
  1. Compiler flag checks for C, C++ , and Fortran ⇒ `check_cxx_compiler_flag("-Wall" CXX_WALL_SUPPORTED)`
  2. INTERFACE libraries
  3. Compile-language generator expressions

```
# define a macro for checking compiler option and adding to a target
macro(ADD_CXX_FLAG_IF_AVAIL     TARG     FLAG         FLAG_NAME)
    if(NOT "${FLAG}" STREQUAL "")
        # runs check to see flag is supported by compiler
        check_cxx_compiler_flag("${FLAG}" ${FLAG_NAME})
        # if the flag is supported...
        if(${FLAG_NAME})
            # add the flag to the interface library when compiling as C++
            target_compile_options(${TARG} INTERFACE
                $<$<COMPILE_LANGUAGE:CXX>:${FLAG}>)
        endif()
    endif()
endmacro()

# provides general compiler flags to targets that "link" to it
add_library(foo-compile-flags INTERFACE)
add_cxx_flag_if_avail(foo-compile-flags "-Wall" CXX_WALL)
add_cxx_flag_if_avail(foo-compile-flags "-fpmodel=precise" CXX_FP_MODEL)
add_cxx_flag_if_avail(foo-compile-flags "-ffp-contract=fast" CXX_FP_CONTRACT)
add_cxx_flag_if_avail(foo-compile-flags "-fstrict-aliasing" CXX_ALIAS_STRICT)
add_cxx_flag_if_avail(foo-compile-flags "-ffast-math" CXX_FAST_MATH)

```

```cmake
find_package(OpenMP REQUIRED)  # find OpenMP

add_library(foo-openmp INTERFACE) # provides OpenMP
target_link_libraries(foo-openmp INTERFACE ${OpenMP_CXX_LIBRARIES})
target_compile_options(foo-openmp INTERFACE
        $<$<COMPILE_LANGUAGE:CXX>:${OpenMP_CXX_FLAGS}>)

add_library(foo-openmp-offload INTERFACE) # provides OpenMP offloading
add_cxx_flag_if_avail(foo-openmp-offload
    "-fopenmp-targets=nvptx64-nvidia-cuda" OMP_OFFLOAD)
# provides standard OpenMP flags PLUS OpenMP offloading flag
target_link_libraries(foo-openmp-offload foo-openmp)

if(OMP_OFFLOAD)
    add_executable(foo-offload foo.cpp)
    target_link_libraries(foo-offload foo-compile-flags foo-openmp-offload)
    set_target_properties(foo-offload PROPERTIES LINKER_LANGUAGE CUDA) # !!!
endif()

add_executable(foo-host foo.cpp)
target_link_libraries(foo-host foo-compile-flags foo-openmp)
```

Future Goals for NERSC

- Hope to provide a repository on NERSC GitHub where users can just copy an INTERFACE library definition into their project, find/replace "NERSC" prefix with their project name, and build seamlessly on their laptop, Jetson Nano, Cori, Permutter, Summit, Frontier, etc.
- For example:
  - NERSC-vectorization $\Rightarrow$ targeted arch vectorization flags
  - NERSC-cuda (and -hip) $\Rightarrow$ compile as X + GPU arch flags
  - NERSC-openmp-offload (and -openacc, etc.)
  - NERSC-*X*-sanitizer $\Rightarrow$ leak, memory, address, thread checkers
  - NERSC-gperftools $\Rightarrow$ lightweight CPU and heap profiler
  - NERSC-ittnotify $\Rightarrow$ hooks to control VTune from code
  - NERSC-code-coverage $\Rightarrow$ generates code coverage
  - NERSC-nvtx $\Rightarrow$ NVTX instrumentation