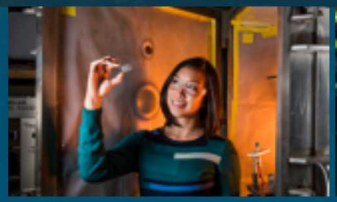
SAND2019-3471PE

Sandia National Laboratories

STK NGP Test: a platform portable unit testing framework

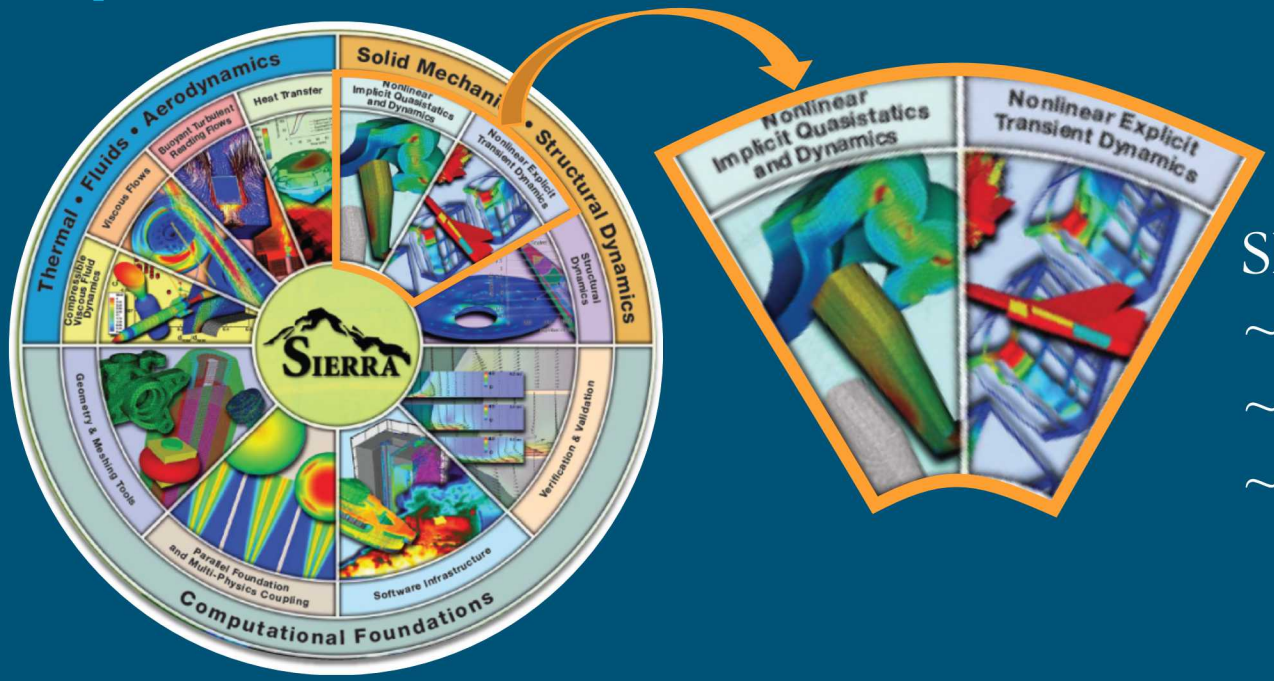


PRESENTED BY

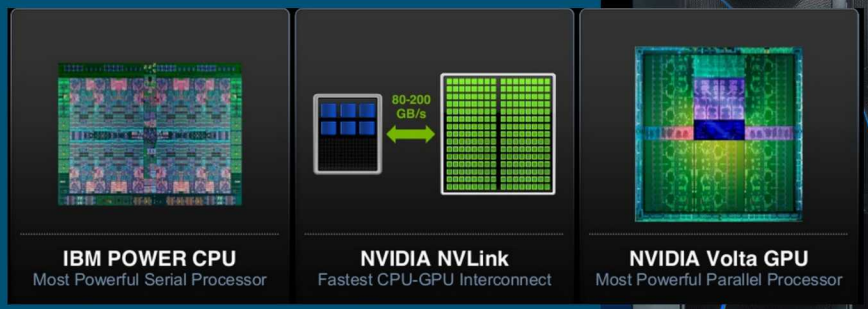
Matthew Mosby, SIERRA/SolidMechanics

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.

SIERRA Multi-Physics Simulation on Sierra!

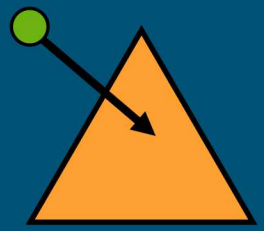


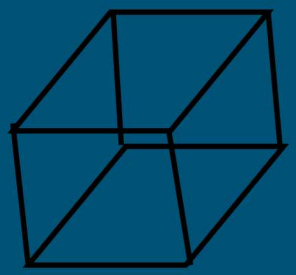
SIERRA/SolidMechanics
~500k lines of C++
~15 current developers
~15+ year dev history

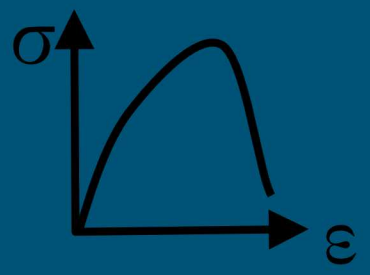


SIERRA/SolidMechanics Path Towards GPU Capability

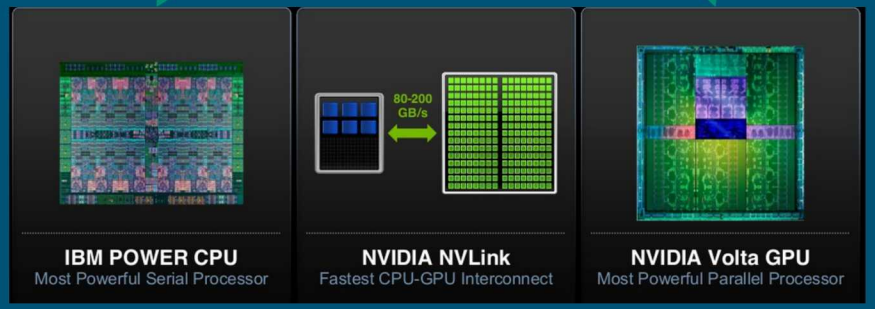
$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}.$$







Unit Tests from TDD on all hardware



- Implement and maintain CPU/GPU execution of core algorithms
  - Support library of vector/tensor and other math functions (~25k lines of C++)
  - Support library of geometry/search algorithms (~30k lines of C++)
  - Both of these libraries have legacy unit test suites that maintain ~95% coverage
  - 863 existing unit tests
- New development of Element/Material libraries
  - Strict adherence to the practice of Test Driven Development (TDD)
  - Implement using Kokkos with GPU/CPU execution as part of the TDD process

Outline

Overview of Test Driven Development (TDD)

My experience developing GPU code using Kokkos

Unit testing on the GPU

The legacy unit test suite problem

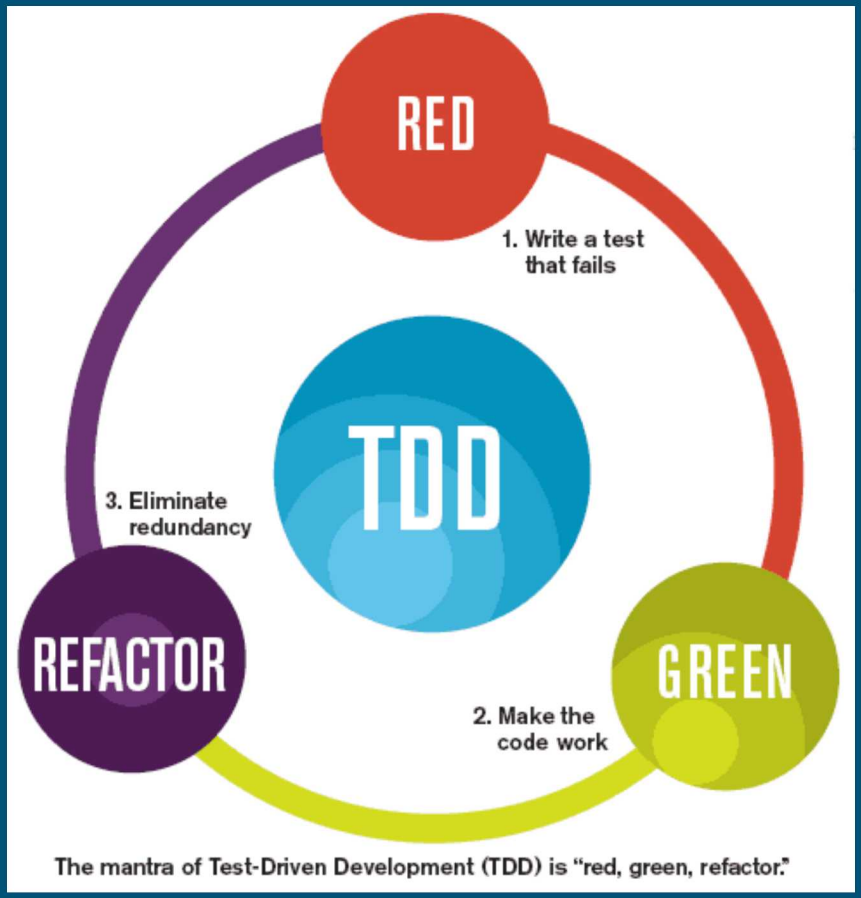
STK NGP Test – Unified unit testing on CPU and GPU

- Migrating existing tests
- How it works

Conclusions and next steps

Test Driven Development – "Red-Green-Refactor"

- Write a failing test – failure to compile is a failing test
- Write as little code as possible to make the test pass
- When the test captures the smallest increment of behavior and passes:
  `git commit`
- Refactor to remove duplication of intent, then:
  `git commit –amend`
- Write the next failing test & repeat

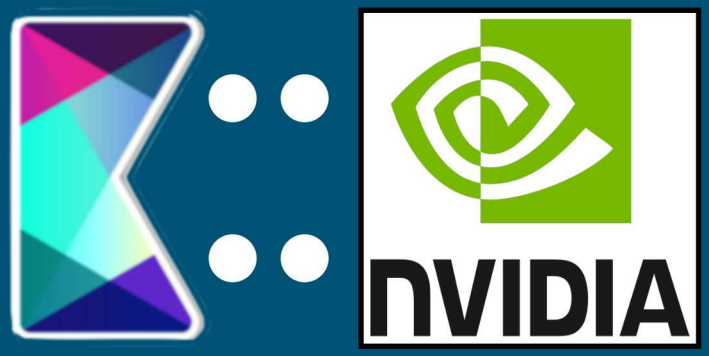


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

You are always a `'git reset'` away from all tests passing!!!

My Experience Using Kokkos for GPU Development

- Things work… unless they don't
- Lots of build warnings that sometimes are runtime errors
- Limited success using debuggers
  - May be related to our build system
  - May be developer inexperience with cuda debuggers
  - Most robust method is good ol' `printf`




```
terminate called after throwing an instance of 'std::runtime_error'
  what():  cudaDeviceSynchronize() error( cudaErrorLaunchFailure):
  unspecified launch failure …/Kokkos_Cuda_Impl.cpp:119
  Traceback functionality not available
Aborted (core dumped)
```

Unit tests can isolate issues to tiny regions of code, making identifying and fixing errors easier

Unit Testing on the GPU

- Some restrictions for execution on the GPU
  - Cannot call `kokkos::parallel_for` from private/protected functions
  - All functions called within kernels must be marked `__device__`
- Use GoogleTest as our unit testing framework
  - Test body is a private function
  - Test `EXPECT/ASSERT_*` macros are not marked for GPU execution
- Best practice is to check conditions after kernel execution rather than within kernel

```
using namespace kokkos;
using View = dual_view<…>;
#define FUNC KOKKOS_FUNCTION

struct functor {
  functor() { … }
  FUNC void operator()() { … }
  View get_result() { … }
  …
};

void execute(functor& f) {
  parallel_for(KOKKOS_LAMBDA(int i)
   { f(); }
}

void verify_results(View out) {
  // EXPECT/ASSERT_* …
}

TEST(GPU, do_something) {
 functor f;
 execute(f);
 verify_results(f.get_results());
}
```

"What about my old unit tests that weren't written for the GPU?"

The Legacy Unit Test Suite Problem

- Legacy unit tests often mix the execution and test/assert portions of the code
- Most legacy unit tests have the main execution directly in the body of the test
- Even well-structured unit tests written for the CPU may have structural impediments to migrating to the GPU

```
TEST(CPU, typical_example) {
 auto obj = do_some_setup();

 // some operation with obj
 …
 // some verification with obj
 // EXPECT/ASSERT_*
 …

 auto obj2 = do_more_setup(obj);

 // some operation with obj2
 …
 // some verification with obj2
 // EXPECT/ASSERT_*
 …
}
```

Migrating existing unit tests to execute on the GPU can be a large effort

STK NGP Test Enables Simplified Migration of Existing Tests

- Include STK NGP Test
- Extract existing test body to free or public function and mark `KOKKOS_FUNCTION`
- Ensure all functions called within this new function are marked `KOKKOS_FUNCTION`
- Prepend GoogleTest macros with `NGP_`
- Execute the new test function in a `'execute'` function wrapping a `kokkos::parallel_for`

Now have a suite of tests executing on the GPU and can confidently refactor

```
#include <stk_ngp_test/ngp_test.hpp>
#define FUNC KOKKOS_FUNCTION

FUNC void old_test_body() {
 auto obj = do_some_setup();

 // some operation with obj
 …
 // some verification with obj
 // NGP_EXPECT/ASSERT_*
 …

 auto obj2 = do_more_setup(obj);

 // some operation with obj2
 …
 // some verification with obj2
 // NGP_EXPECT/ASSERT_*
 …
}

void execute_old_test_body() { … }

NGP_TEST(CPU, migrated_example) {
 execute_old_test_body();
}
```

STK NGP Test – How it Works

- Based on GoogleTest framework
- Reporting managed by instances of `kokkos::experimental::ErrorReproter`
  - Collects error information from `NGP_EXPECT/ASSERT_*` macros on the GPU for reporting outside of the kernel
- Implement `::ngp_testing::Test` that manages clearing/reporting the instances of `ErrorReporter` during setup and tear down
  - Instances of this class are created by the `NGP_TEST` macros
- Implement `NGP_EXPECT/ASSERT_*` macros that behave similarly to the GoogleTest macros

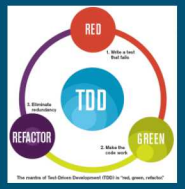
The code is open-source and in Trilinos, feel free to poke around!

Supported STK NGP Test Macros
`NGP_EXPECT/ASSERT_TRUE(bool)`
`NGP_EXPECT/ASSERT_FALSE(bool)`
`NGP_EXPECT/ASSERT_EQ(a, b)`
`NGP_EXPECT/ASSERT_NE(a, b)`
`NGP_EXPECT/ASSERT_LT(less, more)`
`NGP_EXPECT/ASSERT_LE(less, more)`
`NGP_EXPECT/ASSERT_GT(more, less)`
`NGP_EXPECT/ASSERT_GE(more, less)`
`NGP_EXPECT/ASSERT_NEAR(a, b, tol)`
`NGP_TEST(TestSuite, TestName)`
`NGP_TEST_F(TestFixture, TestName)`

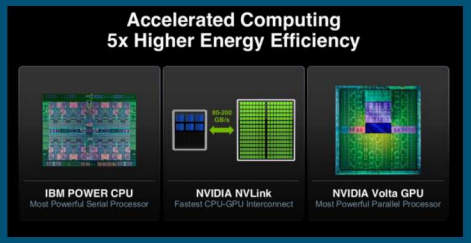
Conclusions and Next Steps

- Robust suites of unit tests are critical to effectively maintaining GPU code
- TDD is a proven method for developing code on the GPU and automatically results in robust unit test suites
- GPU unit tests are structurally different from those targeting CPUs
- STK NGP Test (part of Trilinos) facilitates migration of CPU unit tests to GPU executable unit tests
- Improve error reporting
  - GoogleTest prints values as well as literals passed to `EXPECT/ASSERT_*` macros
- Allow customization of reporters
  - Collect reports across MPI ranks before writing out
  - Specify output stream
- Verify and/or implement advanced usage available in GoogleTest
  - Parameterization of tests
  - Other uses

Image Credits



https://medium.com/mobility/why-developers-scared-to-refactor-code-47efd1b854e7



https://wccftech.com/nvidia-volta-gpus-ibm-power9-cpus-deliver-300-petaflops-performance-2017-summit-sierra-supercomputers/

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}.$$

https://en.wikipedia.org/wiki/Outer_product