Sandia National Laboratories

Exceptional service in the national interest

MEMORY MANAGEMENT AND PROFILING WITH KOKKOS: A TRILINOS CASE STUDY

Christopher Siefert

Kokkos Users Group Meeting, December, 2023.

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-NA0003525.

SAND2023-14515C

OUTLINE

- Trilinos and Kokkos: A Brief History
- WrappedDualView: A Model for Memory Management
- Using Kokkos Tools to Identify H2D/D2H Transfers & Fences
  - Diagnostics
  - Regression
- Note: Even if you're not a Trilinos user, these ideas aren't Trilinos specific. You can feel free to use/modify as you see fit.

KOKKOS & TRILINOS: A BRIEF HISTORY

- Trilinos is a collection of scientific software libraries dating back to the 90s.
  - Originally written for MPI-everywhere use on CPUs.
  - GPU support co-developed with Kokkos.
  - Still a “first user” for Kokkos releases.

- Fun fact: Trilinos was the original home of Kokkos.

- Trilinos objects generally respect the DualView semantic (exist on both Host & Device).
  - Before Kokkos 4.2, there was no portable way to print on device.
  - Trilinos originally used Cuda UVM to provide convenience to applications.
  - Capture of reference counted pointers was problematic.

DEMAND GREW TO REMOVE UVM REQUIREMENT

- UVM support was patchy.
  - Some stuff just didn't work and we had to get fixed.
  - Some things would never be made to work.
  - NVIDIA encouraged us to stop using UVM.
  - Concern over cost (though this was often overblown).

- UVM *did* make it easy to accidently make H2D/D2H transfers. Hard to track with tools.

- Hope was that by getting rid of UVM, apps would identify these transfers and speed up.

HOW TO BEST REMOVE UVM?

- One option would be to expose Kokkos::DualView in all its glory.
- Apps did not like this option. The modify / sync dance was error prone.
- Alternative: Interface which hides the modify/sync and only does them when needed.
- We called this WrappedDualView (WDV); inspired by SYCL's buffer / accessor pattern.
- Note: DualView sync's can do Kokkos::fence() to ensure consistent state.
- Code in: `packages/tpetra/core/src/Tpetra_Details_WrappedDualView.*`

WRAPPEDDUALVIEW (1): CONSTRUCTORS

- Constructor #1: User provides a DualView.
- Constructor #2: User provides a device View. WDV creates host view internally.
- Constructor #3: 1D / 2D subview constructors (WDV + ranges).
- Expert Constructor: 2 DualViews for subview sync management.

- Note: Higher dimensional subviews would work just fine. We just didn't need them.

WRAPPEDDUALVIEW (2): ACCESSORS

- get[Host|Device]View() functions take access tags for sync/modify semantics
  - ReadOnly – Sync yes, modify no.
  - ReadWrite – Sync yes, modify yes.
  - OverwriteAll – Sync no, modify yes.

- Sets the sync/modify flags of underlying DualView.

- Will **throw**: Ask for Host view when a Device view is “checked out” (and vice versa).

- Fun fact: OverwriteAll was originally called “WriteOnly” but the name was misleading, since it would *not* sync your data to the space. OverwriteAll captured the actual behavior better.

WRAPPEDUALVIEW (3): CAVEATS / DETAILS

- OverwriteAll reverts to ReadWrite for SubViews.
- Sync/modify checking gets to be expensive if people are accessing these on Host in loops.
- This matters for us since we have users partially porting CPU-only codes to GPUs.
- Sync/modify checks are only enabled when:
  - Pointers are not aliased OR
  - Execution spaces are GPU spaces.
- This will disable the checking on Serial / OpenMP builds, but leave them on for all GPU builds (including UVM).

WRAPPEDDUALVIEW: FUTURE

- WrappedDualView's design predates wide use of ExecutionSpace instances.
- DualView can generate device-wide syncs, where a stream-sync would have sufficed.
- Will likely allow instance-only sync support, though this can allow for errors.
- Not clear on the interface for this yet.

NOW TO ACTUALLY FIND THE H2D/D2H TRANSFERS

- WrappedDualView got us code safety without UVM!
  - WDV doesn't help us *find* host to device transfers in complicated code.

- Sure you can use a visual profiling tool, but…
  - They're often nearly impossible to read for real apps.
  - They need special tools (e.g. Kokkos tools nvtxConnector) to get the Kokkos goodies.

- Or you could step through it in a debugger
  - Exceptionally labor intensive… when it works at all.

- Many HPC developers (not incorrectly) feel most reliable debugging tool is a printf.

- Our answer: Kokkos Tools.

KOKKOS TOOLS

- SpaceTimeStack is the best-aligned tool to what we want.
  - By providing a stack view of the program, we can localize events to particular chunks of code.
  - Just search the code for the timer label!
  - Controlled via ENV variable KOKKOS_TOOLS_LIBS.

- However it isn't enough
  - Dynamic library loading has to actually work on the platform.
  - Requires you annotate your code with Kokkos::ProfilingRegions.
  - Has a non-configurable floor that drops "small" timers.
  - Doesn't label things, e.g. you have a Kokkos::deep_copy(), but what are you copying, and where?

- Partial solution
  - Modified SpaceTimeStack to add the src/dest Space to the print of Kokkos::deep_copy() calls.

TRILINOS TOOLING (DIAGNOSTIC)

- Trilinos already has a pre-Kokkos Teuchos-based Timer interface
  - Does not create Kokkos::ProfilingRegions necessarily.
  - Apps often print those timers out as part of their regular output.
  - Those timers can be nested (allowing us to track code execution) via StackedTimer.

- Our approach: Use the existing StackedTimer, but use the non-DLL profiling hooks
  - E.g., `Kokkos::Tools::Experimental::set_begin_parallel_for_callback`
  - Have them create Teuchos Timers.
  - Control these via Trilinos ENV variables.

TRILINOS TOOLING (DIAGNOSTIC)

- ENV variables to add Teuchos::Timer objects to various Kokkos things

1. TPETRA_USE_TEUCHOS_TIMERS – Adds Teuchos::Timers to all Tpetra::ProfilingRegions.

2. TPETRA_TIME_KOKKOS_FENCE – Adds Teuchos::Timers to all Kokkos::fence() calls.

3. TPETRA_TIME_KOKKOS_DEEP_COPY – Adds Teuchos::Timers to all Kokkos::deep_copy() calls. This includes space names.
   - TPETRA_TIME_KOKKOS_DEEP_COPY_VERBOSE1 – Adds src/dest View names.
   - TPETRA_TIME_KOKKOS_DEEP_COPY_VERBOSE2 – Adds vector sizes (can generate lots of output in parallel since sizes are different between ranks)

4. TPETRA_TIME_KOKKOS_FUNCTIONS – Adds Teuchos::Timers to Kokkos::parallel_* calls.

- #2 and #3 can also be enabled through explicit function calls.

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.

- Output without any ENV variables set:

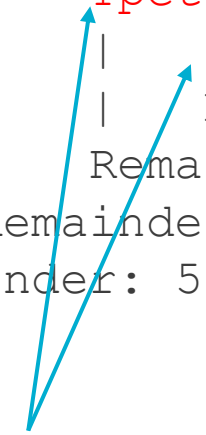
```
Driver: 3.32417 [1]
|   Global: 3.32411 - 99.9983% [1]
|   |   createImport: 0.000472806 - 0.0142235% [1]
|   |   Remainder: 3.32364 - 99.9858%
|   Remainder: 5.5183e-05 - 0.00166006%
[Passed] (3.32 sec)
```

What we named the Timer surrounding the test

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.
  - TPETRA_USE_TEUCHOS_TIMERS

```
Driver: 1.60914 [1]
|   Global: 1.60908 - 99.9965% [1]
|   |   createImport: 0.000482423 - 0.0299813% [1]
|   |   |   Tpetra::Import::init: 0.000376352 - 78.0129% [1]
|   |   |   |   Tpetra::Import::setupSamePermuteRemote: 0.000317982 - 84.4906% [1]
|   |   |   |   Remainder: 5.837e-05 - 15.5094%
|   |   |   Remainder: 0.000106071 - 21.9871%
|   |   Remainder: 1.6086 - 99.97%
|   Remainder: 5.6199e-05 - 0.00349249%
```



Profiling Regions

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.
  - TPETRA_USE_TEUCHOS_TIMERS
  - TPETRA_TIME_KOKKOS_DEEP_COPY

```
Driver: 3.34351 [1]
 |   Global: 3.34346 - 99.9985% [1]
 |    |   createImport: 0.000588269 - 0.0175946% [1]
 |    |    |   Tpetra::Import::init: 0.000497712 - 84.6062% [1]
 |    |    |    |   Tpetra::Import::setupSamePermuteRemote: 0.000462021 - 92.829% [1]
 |    |    |    |    |   Kokkos::deep_copy [Cuda=>Host]: 3.6505e-05 - 7.90116% [1]
 |    |    |    |    |   Kokkos::deep_copy_small [Cuda=>Host]: 1.0617e-05 - 2.29795% [1]
 |    |    |    |    |   Remainder: 0.000414899 - 89.8009%
 |    |    |    |   Remainder: 3.5691e-05 - 7.17101%
 |    |    |   Remainder: 9.0557e-05 - 15.3938%
 |    |   Remainder: 3.34287 - 99.9824%
 |   Remainder: 4.8542e-05 - 0.00145183%
```

Deep Copies

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.
  - TPETRA_USE_TEUCHOS_TIMERS
  - TPETRA_TIME_KOKKOS_DEEP_COPY_VERBOSE1

```
Driver: 3.2199 [1]
 |   Global: 3.21985 - 99.9983% [1]
 |   |   createImport: 0.000559054 - 0.0173628% [1]
 |   |   |   Tpetra::Import::init: 0.000465691 - 83.2998% [1]
 |   |   |   |   Tpetra::Import::setupSamePermuteRemote: 0.000431026 - 92.5562% [1]
 |   |   |   |   |   Kokkos::deep_copy [Cuda=>Host] {lgMap=>lgMap_mirror}: 3.6773e-05 …
 |   |   |   |   |   Kokkos::deep_copy_small [Cuda=>Host] {lgMap=>lgMap_mirror}: 1.0237e-05 …
 |   |   |   |   |   Remainder: 0.000384016 - 89.0935%
 |   |   |   |   Remainder: 3.4665e-05 - 7.44378%
 |   |   |   Remainder: 9.3363e-05 - 16.7002%
 |   |   Remainder: 3.21929 - 99.9826%
 |   Remainder: 5.3361e-05 - 0.00165723%
 [Passed] (3.22 sec)
```

Src=>dest View names

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.
  - TPETRA_USE_TEUCHOS_TIMERS
  - TPETRA_TIME_KOKKOS_DEEP_COPY_VERBOSE2

```
Driver: 1.88643 [1]
|   Global: 1.88638 - 99.9973% [1]
|   |   createImport: 0.000558055 - 0.0295833% [1]
|   |   |   Tpetra::Import::init: 0.000466191 - 83.5385% [1]
|   |   |   |   Tpetra::Import::setupSamePermuteRemote: 0.000428582 - 91.9327% [1]
|   |   |   |   |   Kokkos::deep_copy [Cuda=>Host] {lgMap=>lgMap_mirror,80}:…
|   |   |   |   |   Kokkos::deep_copy_small [Cuda=>Host] {lgMap=>lgMap_mirror,40}…
|   |   |   |   |   Remainder: 0.000383877 - 89.5691%
|   |   |   |   Remainder: 3.7609e-05 - 8.06729%
|   |   |   Remainder: 9.1864e-05 - 16.4615%
|   |   Remainder: 1.88582 - 99.9704%
|   Remainder: 5.0911e-05 - 0.0026988%
[Passed] (1.89 sec)
```

Transfer size

EXAMPLE: TPETRA::IMPORT CONSTRUCTOR

- We added Tpetra::StackedTimer to a test which builds a Tpetra::Import object.
  - TPETRA_USE_TEUCHOS_TIMERS
  - TPETRA_TIME_KOKKOS_DEEP_COPY_VERBOSE2
  - TPETRA_TIME_KOKKOS_FENCE

```
Driver: 3.33936 [1]
|   Global: 3.3393 - 99.9984% [1]
|   |   createImport: 0.000616963 - 0.0184758% [1]
|   |   |   Tpetra::Import::init: 0.000510063 - 82.6732% [1]
|   |   |   |   Tpetra::Import::setupSamePermuteRemote: 0.000473831 - 92.8966% [1]
|   |   |   |   |   Kokkos::fence SharedAllocationRecord<Kokkos::CudaSpace, void>::SharedAllocationRecord(): fence
after copying header from HostSpace (Cuda Instance 1): 1.9424e-05 …
|   |   |   |   |   Kokkos::fence Kokkos::Impl::ViewValueFunctor: View init/destroy fence (Cuda Instance 1):…
|   |   |   |   |   Kokkos::fence HostSpace fence (Serial Instance 1): 1.009e-06 - 0.212945% [2]
|   |   |   |   |   Kokkos::fence Kokkos::Impl::ViewValueFunctor: View init/destroy fence (Serial Instance 1): …
|   |   |   |   |   Kokkos::deep_copy [Cuda=>Host]: 2.9377e-05 - 6.19989% [1]
|   |   |   |   |   Kokkos::fence Kokkos::Cuda::fence(): Unnamed Instance Fence (Cuda Instance 1): …
|   |   |   |   |   Kokkos::deep_copy_small [Cuda=>Host]: 1.0142e-05 - 2.14043% [1]
|   |   |   |   |   Remainder: 0.000400651 - 84.5557%
|   |   |   |   Remainder: 3.6232e-05 - 7.10344%
|   |   |   Remainder: 0.0001069 - 17.3268%
|   |   Remainder: 3.33868 - 99.9815%
|   Remainder: 5.4954e-05 - 0.00164565%
[Passed] (3.34 sec)
```

Execution space fences. Device fences would say “All Instances”

TRILINOS TOOLING (REGRESSION)

- In addition to *diagnostic* tools, Tpetra also provides count-based tools for use in testing.
- Verify: “This code should only have X Kokkos::deep_copy() calls between Spaces”
- We can do this through Tpetra’s DeepCopyCounter and FenceCounter.
  - See: `packages/tpetra/core/src/Tpetra_Details_KokkosCounter.*`
- These require code modification since they’re designed to test specific code fragments.

EXAMPLE: DEEPCOPY COUNTER

```
using Tpetra::Details;
DeepCopyCounter::start();
Kokkos::deep_copy(view1,view2);
size_t ct_same = DeepCopyCounter::get_count_same_space();
size_t ct_diff = DeepCopyCounter::get_count_different_space();
```

- The counter separately tallies copies *within the same* space vs. copies *between* spaces.
- To verify there are no H2D/D2H copies, you want `get_count_different_space();`
- You can also `stop()` the counter and `reset()` it to zero.

EXAMPLE: FENCE COUNTER

```
using Tpetra::Details;
FenceCounter::start();
Kokkos::fence();
size_t ct_inst = FenceCounter::get_count_instance("Cuda");
size_t ct_gbl = FenceCounter::get_count_global("Cuda");
```

- *Instance* and *Global* fences are tallied separately.
  - Think cudaStreamSynchronize vs. cudaDeviceSynchronize.
- Fences are tracked on a space by space basis and you have to ask for the one you want. `execution_space().name()` will give you the string you want.
- You can also `stop()` the counter and `reset()` it to zero.

CONCLUSIONS

- Kokkos doesn't solve all problems. Often, you can build solutions using Kokkos.
- WrappedDualView provides a safe encapsulation for DualView
  - Still need to consider ExecutionSpace instances.
- Use Kokkos Tools to augment existing stacked timers.
  - Controllable output with Trilinos-level ENV vars.
- Use Kokkos Tools for regression testing.
  - Ensure nobody introduces extraneous fences / copies.