Kokkos Tools Training

The Kokkos Team & Kevin Huck

August 17-19, 2021, OLCF

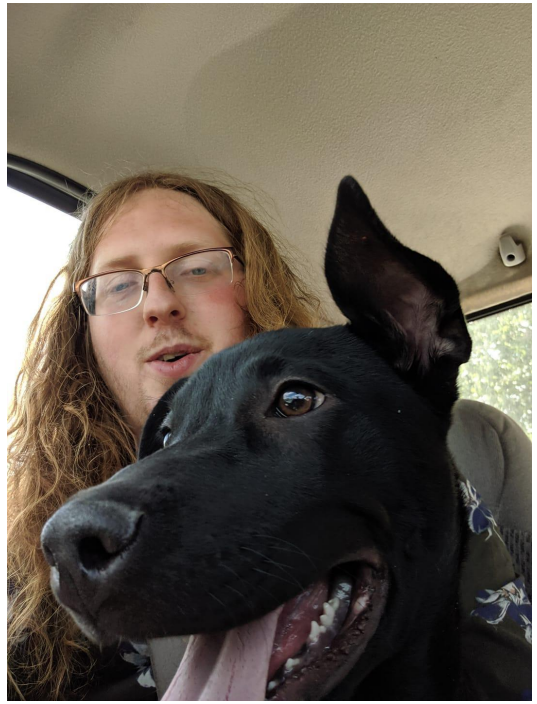
Lawrence Berkeley
National Laboratory

SAND2021-10002 TR

U.S. DEPARTMENT OF ENERGY | Office of Science

What is the Kokkos Tools effort?

- Kokkos aims to provide a unified interface to a variety of hardware and programming models
- Kokkos *Tools* does the same, but for tooling
- Current mature capability areas
  - Profiling
  - Autotuning
- Exploratory
  - Compilers
  - IDE integrations
  - Debuggers



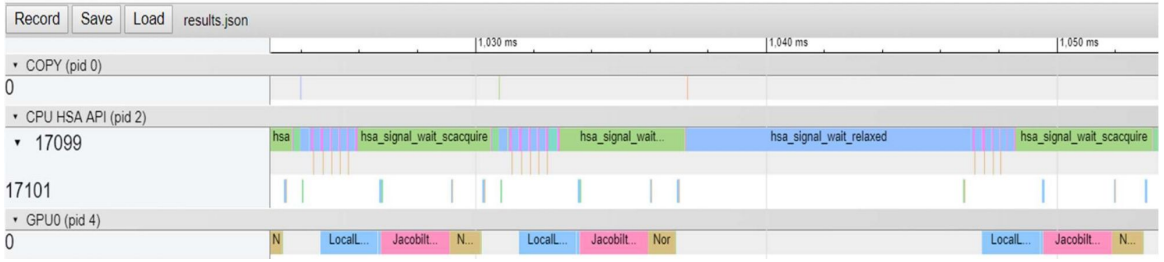
David Poliakoff:
Profiling tools,
Debuggers,
Autotuning,
IDEs,
Dog facts

Drew Lewis:
Compilers

Why Kokkos Tools?

- "Toolchain-per-architecture" undesirable

NVIDIA nsys



AMD Rocprof

- Tooling with Kokkos Semantics, not C++
- In C++: "void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<_GLOBAL__N__49_tmpxft_00004d6b_00000000_7_integrator_nve_cpp1_ii_28abe736::InitialIntegrateFunctor, Kokkos::RangePolicy<>, Kokkos::Cuda>>(_GLOBAL__N__49_tmpxft_00004d6b_00000000_7_integrator_nve_cpp1_ii_28abe736::InitialIntegrateFunctor"
- In Kokkos: "IntegratorNVE::initial_integrate"

Design Goals

- Don’t have “tool-enabled builds,” always enable tools. Turning them on or off should be a *runtime decision*
  - Necessitates *zero or very low overhead when not in use* (we achieve this)
- Function-pointer callback-based system. On Unix, we dlopen a tool library and fill out function pointers from it
  - Comparing those function pointers to nullptr is very fast
- Events we track
  - Kernels, Regions, Metadata, Memory alloc/free (including Views), DualView operations
- Events we will soon track
  - Using a View in a kernel

How do I integrate these into my Kokkos code?

```
Kokkos::Tools::pushRegion("my_region");
Kokkos::View<float*> my_view("my_view",100);
Kokkos::parallel_for("my_kernel",RP(0,5),KOKKOS_LAMBDA(int i){});
Kokkos::Tools::popRegion();
```

Instrumentation “built-in” to Kokkos Core

./my_application [run with no tool]

./my_application --kokkos-tools-library=/path/to/tool.so [run with a tool]

KOKKOS_PROFILE_LIBRARY=/path/to/tool.so ./my_application [run with a tool]

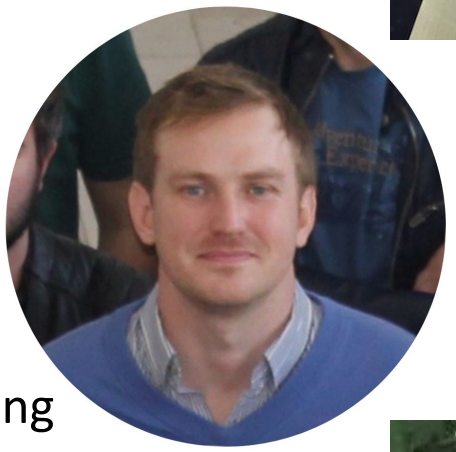
No recompilation, just add a command line argument!

Where to get Tools that support this?

- Kokkos Tools repo
  - git@github.com:kokkos/kokkos-tools
  - Simple tools to do simple tasks, builds are *trivial* (just type “make”)
- Caliper
  - git@github.com:LLNL/caliper
  - More complicated, more powerful. I (David P) tend to prototype functionality here
  - UVM Profiling, SPOT performance tracking
- APEX
  - git@github.com:khuck/xpress-apex
  - Developed out of University of Oregon, popular with many ORNL users
  - Supports profiling a wide variety of programming models, *and* autotuning
  - Handles asynchronous tasks, unlike many other tools
  - Slices, dices, juliennes fries

Tools not discussed here

- TAU
  - TAU has six or seven *thousand* excellent trainings a year ☺️
  - Sameer Shende could do a Kokkos Tools/TAU one if this is of interest
- Timemory
  - Great tool. So great the developer got hired by AMD
    - Oops
  - Worth investigating if you're interested in incorporating advanced measurements through template metaprogramming
- Score-P
  - Another very good tool, with Bill Williams heading up Kokkos Tools efforts
  - Happy to put you in contact with Bill



Code samples to show this in action

- git@github.com:DavidPoliakoff/kokkos-tools-examples
- Will build:
  - Caliper
  - Kokkos Tools
  - APEX
  - Examples
- Configure it like you would Kokkos, it will configure these tools appropriately (also builds Kokkos)
- NVIDIA: cmake -DKokkos_ENABLE_TUNING=ON -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=Debug -DKokkos_ENABLE_CUDA_LAMBDA=ON -DKokkos_ENABLE_CUDA=ON -DKokkos_ARCH_VOLTA70=ON ..
- Known bug: there's a bug in our install. If you add the install location/lib64 to LD_LIBRARY_PATH, the examples run

Simple Tools

Why Simple Tools?

- *Suppose* DOE was purchasing new architectures with new toolchains at an incredible clip
  - I know, it's inconceivable
- Do we *really* have to learn a toolchain per architecture for simple tasks?
  - No

```
BEGIN KOKKOS PROFILING REPORT:

DEVICE ID: Cuda device 256, instance Global Instance
TOTAL TIME: 0.0993753 seconds
TOP-DOWN TIME TREE:
<average time> <percent of total time> <percent time in Kokkos> <percen
kernels per second> <number of calls> <name> [type]
====================
|-> 7.27e-02 sec 73.2% 96.4% 0.0% 2.7% 9.66e+15 200 edit_step [region]
|   |-> 7.01e-02 sec 70.5% 100.0% 0.0% ------ 200 edit [reduce]
|   |   |-> 3.76e-04 sec 0.4% 0.0% 0.0% ------ 400 Kokkos::Tools::invok
file Tool Fence [fence]
|   |-> 5.77e-04 sec 0.6% 0.0% 0.0% ------ 800 Kokkos::Tools::invoke_ko
 Tool Fence [fence]
|-> 2.42e-02 sec 24.4% 100.0% 0.0% ------ 200 decrease_temp [for]
|   |-> 2.26e-02 sec 22.7% 0.0% 0.0% ------ 400 Kokkos::Tools::invoke_k
e Tool Fence [fence]
|-> 5.56e-04 sec 0.6% 0.0% 0.0% ------ 800 Kokkos::Tools::invoke_kokkos
l Fence [fence]
|-> 3.25e-04 sec 0.3% 0.0% 0.0% ------ 1 Kokkos::View<...>::View: fence
```

Space-time Stack: where am I spending time and memory?

Space-Time Stack: Dead simple, highly useful tool

- For this part, I recommend using your own Kokkos code. If you don't have one, though, try the "instances" example in the examples repo
- Running is *extremely* complicated:
  - Set KOKKOS_PROFILE_LIBRARY to [examples install dir]/lib64/kp_space_time_stack.so
  - Run your program

```
TOP-DOWN TIME TREE:
<average time> <percent of total time> <percent time in Kokkos> <percent MPI imbalance> <remainder> <
kernels per second> <number of calls> <name> [type]
==================
|-> 6.01e+00 sec 28.0% 100.0% 0.0% ------ 200000 "temperature_two_mirror"="temperature_two" [copy]
|   |-> 3.19e-01 sec 1.5% 0.0% 0.0% ------ 400000 Kokkos::deep_copy: copy between contiguous views, p
ost deep copy fence [fence]
```

Space-time-stack: continued

```
KOKKOS HOST SPACE:
==================
MAX MEMORY ALLOCATED: 125.0 kB
ALLOCATIONS AT TIME OF HIGH WATER MARK:
  50.0% temperature_one_mirror
  50.0% temperature_two_mirror

KOKKOS CUDA SPACE:
==================
MAX MEMORY ALLOCATED: 309.3 kB
ALLOCATIONS AT TIME OF HIGH WATER MARK:
  20.7% Kokkos::InternalScratchSpace
  20.7% Kokkos::InternalScratchSpace
  20.2% temperature_one
  20.2% temperature_two
```

Try it yourself!

Run space-time-stack on your own code, or on one of the examples

Simple Tools: Advanced Mode

```
Kokkos::DefaultExecutionSpace root_space;
auto instances = Kokkos::Experimental::partition_space(root_space, 1, 1);
view_type temperature_field1("temperature_one", data_size);
view_type temperature_field2("temperature_two", data_size);
auto f1_mirror = Kokkos::create_mirror_view(temperature_field1);
auto f2_mirror = Kokkos::create_mirror_view(temperature_field2);
for (int x = 0; x < repeats; ++x) {
  Kokkos::parallel_for(
      "process_temp1",
      Kokkos::RangePolicy<Kokkos::DefaultExecutionSpace>(instances[0], 0,
                                                         data_size),
      KOKKOS_LAMBDA(int i) { temperature_field1(i) -= 1.0f; });
  Kokkos::deep_copy(f1_mirror, temperature_field1);
```

Finding fences

KOKKOS_PROFILE_LIBRARY=./lib64/kp_space_time_stack.so
./bin/instances_begin --kokkos-tools-args=--separate-devices

```
DEVICE ID: Cuda device 256, instance Global Instance
TOTAL TIME: 27.2033 seconds
TOP-DOWN TIME TREE:
<average time> <percent of total time> <percent time in Kokkos> <per
kernels per second> <number of calls> <name> [type]
====================
|-> 7.58e+00 sec 27.9% 100.0% 0.0% ------ 200000 "temperature_two_mi
|   |-> 3.63e-01 sec 1.3% 0.0% 0.0% ------ 200000 Kokkos::deep_copy:
re view equality check [fence]
|   |-> 3.43e-01 sec 1.3% 0.0% 0.0% ------ 200000 Kokkos::deep_copy:
ost deep copy fence [fence]
```

Try it yourself!

Run space-time-stack on the instances_begin example, showing per-device operations

```
KOKKOS_PROFILE_LIBRARY=./lib64/kp_space_time_stack.so  ./bin/instances_begin --kokkos-tools-args=--separate-devices
```

Fixed

```
Kokkos::DefaultExecutionSpace root_space;
auto instances = Kokkos::Experimental::partition_space(root_space, 1, 1);
view_type temperature_field1("temperature_one", data_size);
view_type temperature_field2("temperature_two", data_size);
auto f1_mirror = Kokkos::create_mirror_view(temperature_field1);
auto f2_mirror = Kokkos::create_mirror_view(temperature_field2);
for (int x = 0; x < repeats; ++x) {
  Kokkos::parallel_for(
      "process_temp1", Kokkos::RangePolicy<Kokkos::DefaultExecutionSpace>(instances[0], 0, data_size),
      KOKKOS_LAMBDA(int i) { temperature_field1(i) -= 1.0f; });
  Kokkos::deep_copy(instan[0], f1_mirror, temperature_field1);
```

Note: the only fences are Tool fences

```
DEVICE ID: Cuda device 256, instance Global Instance
TOTAL TIME: 21.1043 seconds
TOP-DOWN TIME TREE:
<average time> <percent of total time> <percent time in Kokkos> <percent MPI
imbalance> <remainder> <kernels per second> <number of calls> <name> [type]
====================
|-> 5.12e+00 sec 24.3% 100.0% 0.0% ------ 200000 "temperature_one_mirror"="te
mperature_one" [copy]
|-> 5.11e+00 sec 24.2% 100.0% 0.0% ------ 200000 "temperature_two_mirror"="te
mperature_two" [copy]
|-> 2.00e+00 sec 9.5% 0.0% 0.0% ------ 800000 Kokkos::Tools::invoke_kokkosp_c
allback: Kokkos Profile Tool Fence [fence]
```

Using Third-Party Tools

Using Third Party Tools

Kokkos Tools can also be used to interface and augment existing profiling tools.

- - Provide context information like Kernel names
- - Turn data collection on and off in a tool independent way

There are two ways this happens:

- Load a specific connector tool like nvprof-connector
  - For example for Nsight Compute and Vtune

- Tools themselves know about Kokkos instrumentation
  - For example Tau

Using Third Party Tools – Nsight Systems

Use the nvprof-connector to interact with NVIDIA tools

Translates KokkosP hooks into NVTX instrumentation

- Works with all NVIDIA tools which understand NVTX
- Translates Regions and Kernel Dispatches

Initially wasn't very useful since regions are shown independently of Kernels

But CUDA 11 added renaming of Kernels based on Kokkos User feedback!

Using Third Party Tools – Nsight Systems

Run your application using
nsys profile -f true --stats=true -o out.qdrep ./instances_end

```
NVTX Range Statistics:

 Time(%)  Total Time (ns)  Instances  Average (ns)  Minimum (ns)  Maximum (ns)  StdDev (ns)   Style        Range
 -------  ---------------  ---------  ------------  ------------  ------------  -----------  --------  -------------
    50.0       5361259390     200000       26806.3         25562        695944       3236.8  StartEnd  process_temp1
    50.0       5360374005     200000       26801.9         25497        699699       2305.7  StartEnd  process_temp2
```

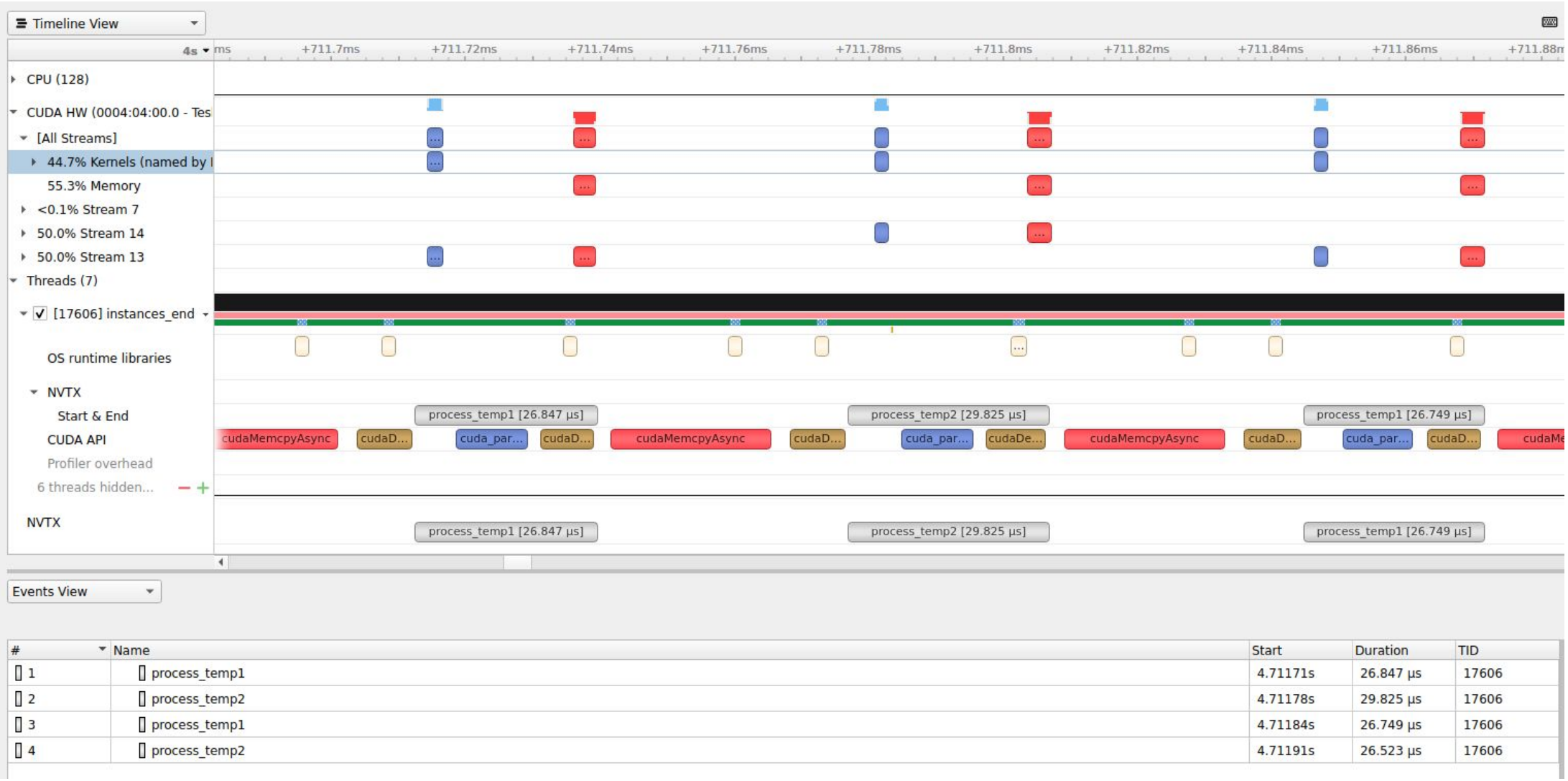
To enable kernel renaming you need to:

- Load the nvprof-connector via setting KOKKOS PROFILE LIBRARY in the run configuration.
- Go to Tools > Preferences > Rename CUDA Kernels by
- NVTX and set it on.

This does a few things:

- User Labels are now used as the primary name.
- You can still expand the row to see which actual kernels are grouped under it.
- The bars are now named Label/GLOBAL FUNCTION NAME.`

Using Third Party Tools – Nsight Systems



Caliper

Caliper: a Performance Analysis Toolbox

- Developed at Lawrence Livermore National Lab
- https://software.llnl.gov/Caliper/
- *Significantly* more than a Kokkos Tool, but a great Kokkos Tool
- KOKKOS_PROFILE_LIBRARY=/path/to/libcaliper.so
- Configuration
  - Set Caliper environment variables
  - Or use prebaked configs
  - "--kokkos-tools-args=config," or "CALI_CONFIG=config"
  - Generally, add "profile.kokkos" to a config to get Kokkos profiling

David Boehme: "the Caliper man"

Simple timing

```
(base) [dzpolia@kokkos-dev-2 tool-playground]$ ./bin/instances_begin --kokkos-tools-args="
runtime-report(profile.kokkos)" --kokkos-tools-library=./lib64/libcaliper.so 2>&1 | tee ca
liper_log
Path                                                Time (E) Time (I) Time % (E) Time % (I)
process_temp2                                       4.254843 6.197905   7.960957  11.596493
  Kokkos::Tools::invoke_~~kos Profile Tool Fence    1.943062 1.943062   3.635536   3.635536
Kokkos::deep_copy: copy~~s, post deep copy fence    3.728552 3.728552   6.976248   6.976248
Kokkos::deep_copy: copy~~pre view equality check    3.842599 3.842599   7.189634   7.189634
process_temp1                                       4.281627 6.225261   8.011071  11.647677
  Kokkos::Tools::invoke_~~kos Profile Tool Fence    1.943634 1.943634   3.636606   3.636606
Kokkos::Tools::invoke_k~~kkos Profile Tool Fence    3.590691 3.590691   6.718306   6.718306
Kokkos::CudaInternal::i~~on space initialization    0.000030 0.000030   0.000056   0.000056
```

Okay, so it does the space-time-stack? Why Caliper?

- In addition to simple timings, Caliper supports an unbelievable array of profiling capabilities
  - Often the first place we prototype functionality
- Tech not discussed here
  - SPOT: performance tracking utility, see whether you're helping or harming the performance of a codebase as you develop it
  - Hatchet: slice and dice your calltrees, calculate which parts of a program are speeding up or slowing down
  - CurIOs: IO profiling
- There are entire Caliper trainings available

UVM Profiling: a Caliper case study

```
for (int x = 0; x < repeats; ++x) {
  Kokkos::parallel_for(
      "decrease_temp", Kokkos::RangePolicy<Kokkos::Cuda>(0, data_size),
      KOKKOS_LAMBDA(int i) { temperature(i) -= 1.0f; });
  Kokkos::Tools::pushRegion("edit_step");
  if ((x % output_interval) == 0) {
    double temperature_sum = 0.0;
    Kokkos::parallel_reduce(
        "edit", Kokkos::RangePolicy<Kokkos::Serial>(0, data_size),
        KOKKOS_LAMBDA(int i, double &contrib) {
          contrib += temperature(i);
        },
        Kokkos::Sum<double>(temperature_sum));
    std::cout << "Sum of temperatures on iteration " << x << ": "
              << temperature_sum << std::endl;
  }
```

What can we see?

- ./bin/uvm_caliper ./bin/uvm_begin
- “uvm_caliper” just sets environment variables

Path	alloc.label#cupti.fault.addr	cupti.uvm.kind	inclusive#sum#cupti.u
edit_step	temperature	DtoH	1310720
edit	temperature	DtoH	1310720
Kokkos::Tools::invoke~~os Profile Tool Fence	temperature	DtoH	1310720
decrease_temp	temperature	HtoD	1310720
Kokkos::Tools::invoke_~~kos Profile Tool Fence	temperature	HtoD	1310720
Kokkos::Tools::invoke_k~~kkos Profile Tool Fence			
|-	temperature	HtoD	6553
|-	temperature	DtoH	6553

Try it yourself!

Run Caliper on your own code using UVM, or on the UVM examples

./bin/uvm_caliper ./bin/uvm_begin

Typical optimization path

- Understand Kokkos Utilization (SpaceTimeStack)
  - Check how much time in kernels
  - Identify HotSpot Kernels
- Run Memory Analysis (MemoryEvents)
  - Are there many allocations/deallocations - 5000/s is OK.
  - Identify temporary allocations which might be able to hoisted
- Identify Serial Code Regions (SpaceTimeStack)
  - Add Profiling Regions
  - Find Regions with low fraction of time spend in Kernels
- Dive into individual Kernels
  - Use connector tools to analyze kernels.
  - E.g. use roof line analysis to find underperforming code.

Autotuning

Why autotune?

- Figuring out the ideal “tuning knobs” for
  - NVIDIA, AMD, Intel (GPU+CPU)
  - OpenMPTarget, HIP, CUDA
  - V100, A100, MI100
  - Every compiler
  - For every kernel
- Nobody has the time

Autotuning: the traditional code team conversation

- Autotuning dev: “I have a technology for you to use for autotuning”
- Code team dev: “How do I get started with it?”
- Autotuning dev: “Okay, so first you build PyTorch, then you build with a compiler wrapper, then you run this Python script to run your application”
- Code team dev: “…”

This autotuning is *simple* and *non-invasive*

Standard Usage in Codes

```
using team_policy = Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace>;
using team_member = typename team_policy::member_type;
Kokkos::parallel_for(
    "compute", team_policy(data_size, Kokkos::AUTO, Kokkos::AUTO),
```

```
using mdrange = Kokkos::MDRangePolicy<Kokkos::Rank<2>>;
Kokkos::parallel_for("mdrange", mdrange({0,0},{data_size, data_size}),
```

Soon

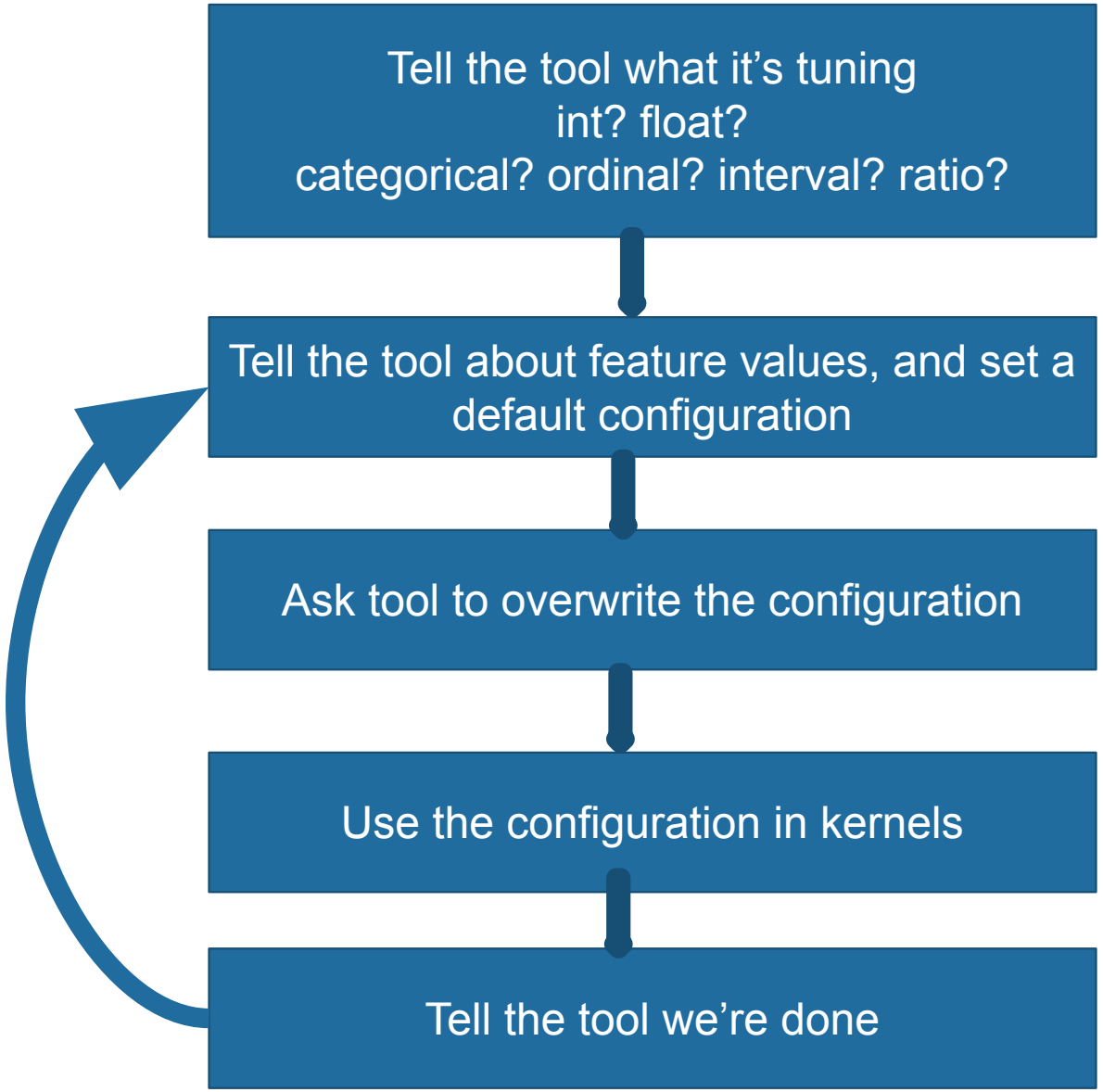
```
Kokkos::parallel_for("range", Kokkos::RangePolicy<>(0,data_size),
```

For most users: *no code changes*

Changes in build/run

- Additional Cmake Option
  - -DKokkos_ENABLE_TUNING=ON
- To tune policy details like those above, a command-line argument
  - --kokkos-tune-internals
- Plan to remove these in time, but these are still undergoing overhead testing
  - Likely to change to needing options to turn these *off* very soon

Mechanics



See the tuning_mechanics example to see how this works in code, if you want to tune your own parameters.

Or don't, you don't have to for most uses

Get the Tools

- Apex: already included in the repo
- Apollo: If you have Spack, download the repo here:
  - https://github.com/DavidPoliakoff/tuning-spack
  - spack repo add /path/to/the/checkout/directory
  - spack install apollo@develop
- Additional Apollo credits:
  - Giorgis Giorgakoudis, LLNL
  - David Beckingsale, LLNL
  - Todd Gamblin, LLNL

Chad Wood, Apollo, UO,
cdw@cs.uoregon.edu

Graduating and looking for jobs

Kevin Huck, APEX, UO

Probably not looking for a new job, but you could always try

Which Tool?

- All notes are as of the date this was delivered (Aug 17, 2021)
- Apollo
  - Con: OpenCV dependency
    - Spack mitigates this
  - Con: reading tuning “outputs” difficult
    - Being worked on
  - Pro: OpenCV capabilities
    - **Decision trees over features**
  - Pro: “Retraining” if results diverge from model
  - Pro: Distributed training
  - Pro: Investment from both LLNL and Sandia
- Apex
  - **Pro: no external dependencies Apex doesn’t build itself**
  - Pro: Already used by some ORNL codes
  - Pro: Investment from DOE, NSF, and DoD
  - Pro: also a capable profiling tool
  - Con: Current tuning simple, but often effective
    - No decision trees over features, tuning a 1k row matric and tuning 1k+1 row matrix are different problems
    - Being worked on
    - For *most* purposes, this is fine

Running a code with these tools

- Apex: ./bin/apex_exec --apex:kokkos_tuning --apex:kokkos ./application --kokkos-tune-internals
- Apollo: KOKKOS_PROFILE_LIBRARY=/path/to/libapollo.so ./application --kokkos-tune-internals

Try it yourself!

Run a tuning tool on the tuning_begin example

Just add (tuning) features!

- Code walkthrough

Advanced Topics

Homegrown Tuning

- Suppose you want to tune something that isn’t a Kokkos parameter
- Code walkthrough

Skylos

- Example only

Where do I get more help?

- Highly responsive GitHub repo(s)
  - github.com/kokkos/kokkos
  - github.com/kokkos/kokkos-kernels
  - github.com/kokkos/kokkos-tools
- Lectures
  - kokkos.link/the-lectures
- **Slack**
  - kokkosteam.slack.com
- Email
  - dzpolia@sandia.gov
  - crtrott@sandia.gov

What do YOU need?

- What is the hardest part about developing Kokkos?
- What's the last bug you had that took a week to debug?
- What are the most bewildering compiler errors you've encountered with Kokkos?