ESPM2: 2022 ACM/IEEE 7th International Workshop on Extreme Scale Programming Models and Middleware

Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX

Kevin Huck, Senior Research Associate, Oregon Advanced Computing Institute for Science and Society (OACISS), University of Oregon

Kevin Huck is a Senior Research Associate in the Oregon Advanced Computing Institute for Science and Society (OACISS) at the University of Oregon. Dr. Huck has several years of experience in the computer software industry prior to engaging in Computer Science research. He has worked in the areas of performance data mining, performance analysis, software tool design and software engineering. He is interested in the unique problems of intelligent, automated performance analysis of very large datasets as well as automated methods for diagnosing and treating performance problems both offline and with runtime controls. He has collaborated on over 70 publications and has served on several conference/workshop committees, including SC, ISC, ICCS, ICPP, VECPAR, IPDPS, and ParCo. His MS and PhD degrees in Computer and Information Science are from the University of Oregon, and his BS in Computer Science is from the University of Cincinnati.

Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX

Kevin Huck
iD 0000-0001-7064-8417
Email: khuck@cs.uoregon.edu

GitHub: https://github.com/UO-OACISS/apex
Documentation: http://uo-oaciss.github.io/apex/

Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX

Kevin Huck

khuck@cs.uoregon.edu

https://github.com/UO-OACISS/apex

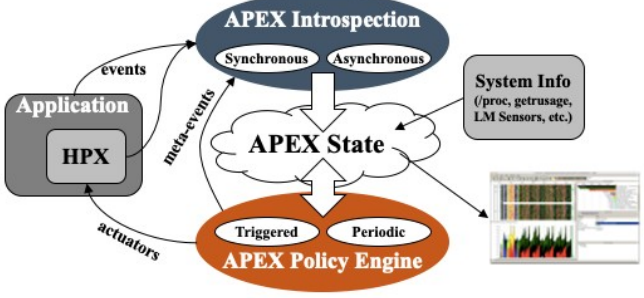
RAPIDS

U.S. DEPARTMENT OF ENERGY | Office of Science

UNIVERSITY OF OREGON

APEX Introduction

- Autonomic Performance Environment for eXascale
  1. **Performance Measurement**
  2. **Runtime Adaptation**
- Designed for AMT runtimes (HPX)
  - But works with “conventional” parallel models



- Focus on **task dependency** graph, not calling context graph
- Supports HPX, C/C++ threads, OpenMP, OpenACC, Kokkos, Raja, CUDA, HIP, (OneAPI, StarPU in dev)…
- https://github.com/UO-OACISS/apex
- Active Harmony* (Nelder Mead), Simulated Annealing, hill climbing for parametric search methods

---

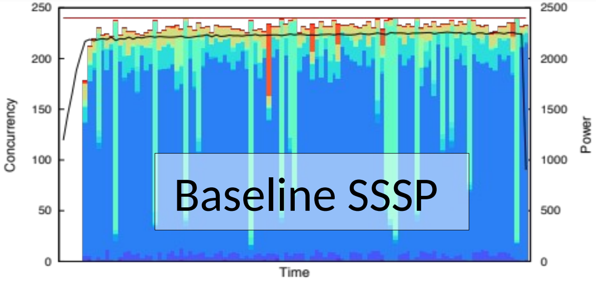
*https://www.dyninst.org/harmony

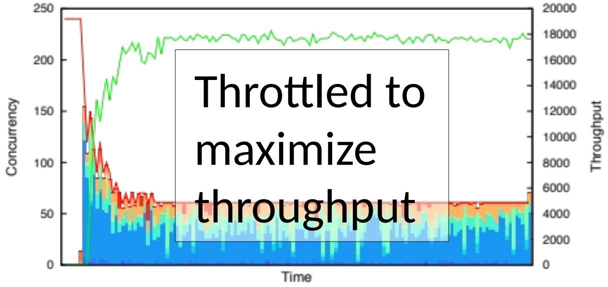
APEX and HPX

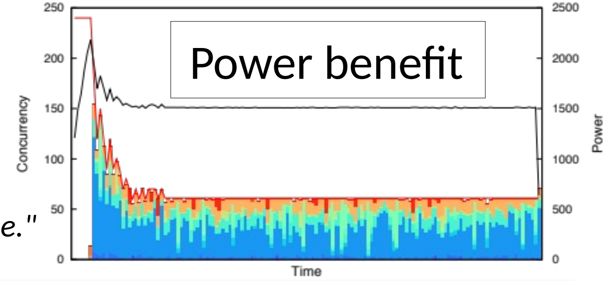
- **HPX**: Asynchronous Many-Task Runtime system in C++
- Data and task dependencies can be expressed with HPX **futures** and **continuations**, chained together in an **execution graph**.
- The graph can be built asynchronously
- HPX tasks are created, scheduled, executed, and usually yielded and resumed by the runtime system scheduler
- This is particularly challenging because many different OS threads may have participated in the execution of the HPX task during its lifetime, and the calling context tree is meaningless to the application developer because it consists of runtime system functions, not application tasks

APEX and HPX

- APEX is **integrated into the HPX thread scheduler**, uniquely identifies each task with a **GUID**, and tracks all state transitions for a given task.
- Policy Engine used for **tuning heuristic control knobs** in HPX thread scheduler, networking
    - Soft power caps, maximize throughput, reduce network latency, …

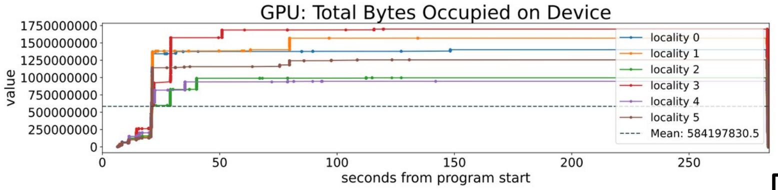




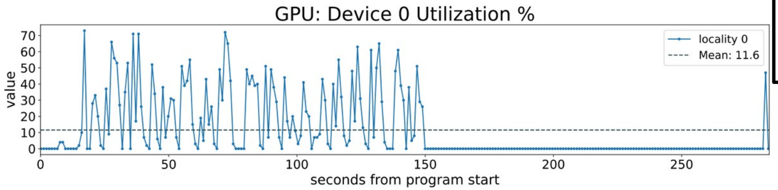


Figures: Huck, et al. "An autonomic performance environment for exascale." Supercomputing frontiers and innovations 2.3 (2015): 49-66.

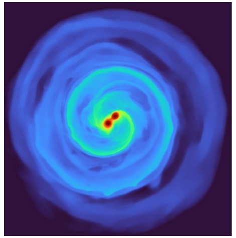
APEX example – Octo-Tiger (HPX)

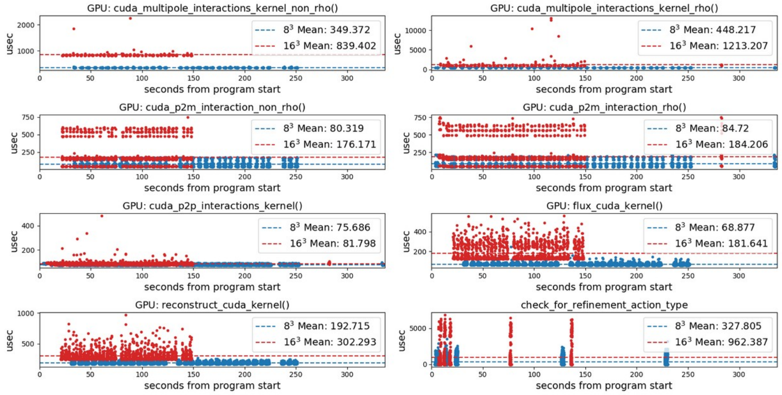


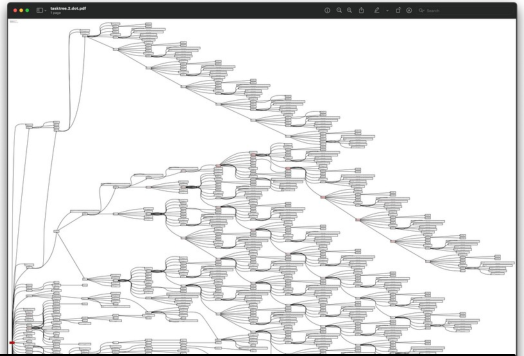
Tracking GPU memory usage with CUPTI



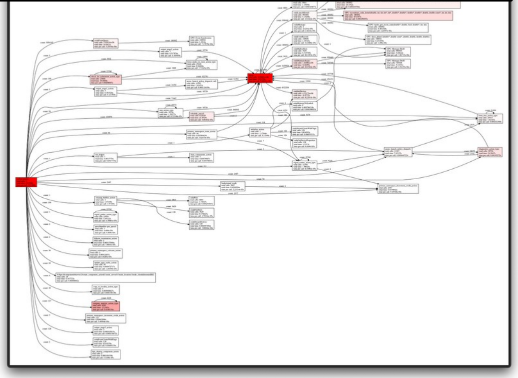
Monitoring GPU utilization with NVML library







Full task tree (above) and task graph (below) showing task dependencies



Comparing subgrid sizes and relative kernel performance with CUPTI device activity

https://github.com/STEllAR-GROUP/octotiger
https://github.com/STEllAR-GROUP/hpx

Diehl, Patrick, et al. "Octo-Tiger's New Hydro Module and Performance Using HPX+CUDA on ORNL's Summit." *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021.

How is APEX different?

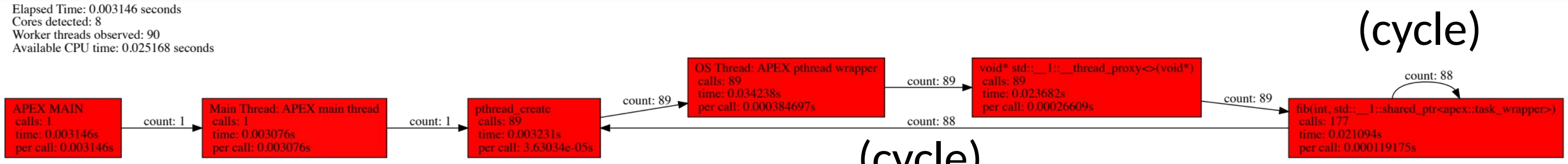
- **No shortage** of existing performance measurement tools
  - Primarily designed for **post-mortem** analysis
  - First-person measurement of **tied tasks/functions** on an OS thread – not untied tasks, runtime thread control, third-person measurement, runtime control of parameters
  - Not designed for **permanent integration** into applications
  - OS-thread context can be limiting
  - Vendor tools are great!...but limited to each vendors' architecture/process – can't easily do cross-platform analysis
- APEX helps address these needs

Measurement Capabilities

- **Timers** – "per-process" aggregation for all timers in runtime performance state
  - Start, stop, **yield, resume**
  - All times are inclusive (unless yielded/resumed)
- **Counters** – discrete sampling of some data point in time
  - Can be associated with timed regions, e.g. MPI_Send #bytes
  - Can be periodically captured and aggregated, e.g. power, utilization, hardware counters, OS counters
- **Task dependency chains**
  - Each task has a unique parent
  - APEX builds graphs/trees of dependencies at runtime

Example: C++ std::thread fib(10)

Task Graph: aggregated, contains cycles


Elapsed Time: 0.003146 seconds
Cores detected: 8
Worker threads observed: 90
Available CPU time: 0.025168 seconds

APEX MAIN
calls: 1
time: 0.003146s
per call: 0.003146s

count: 1

Main Thread: APEX main thread
calls: 1
time: 0.003076s
per call: 0.003076s

count: 1

pthread_create
calls: 89
time: 0.003231s
per call: 3.63034e-05s

count: 89

OS Thread: APEX pthread wrapper
calls: 89
time: 0.034238s
per call: 0.000384697s

count: 89

void* std::__1::__thread_proxy<>(void*)
calls: 89
time: 0.023682s
per call: 0.00026609s

count: 89

fib(int, std::__1::shared_ptr<apex::task_wrapper>)
calls: 177
time: 0.021094s
per call: 0.000119175s

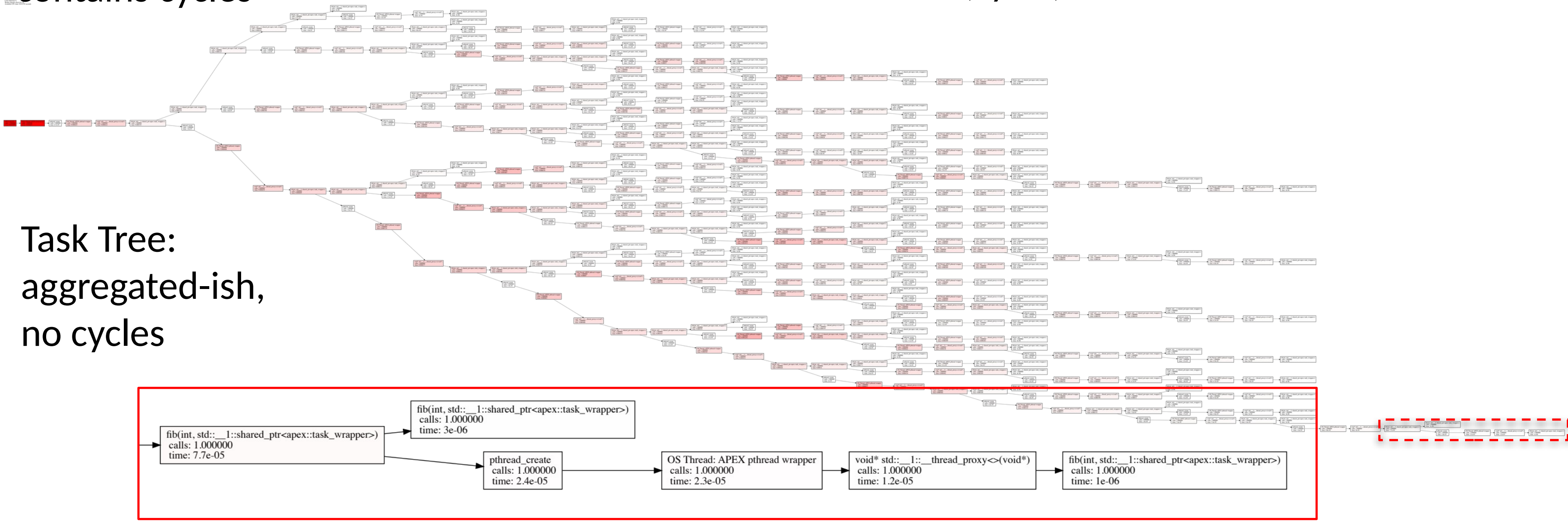
count: 88

count: 88

(cycle)

(cycle)

Task Tree: aggregated-ish, no cycles


fib(int, std::__1::shared_ptr<apex::task_wrapper>)
calls: 1.000000
time: 7.7e-05

fib(int, std::__1::shared_ptr<apex::task_wrapper>)
calls: 1.000000
time: 3e-06

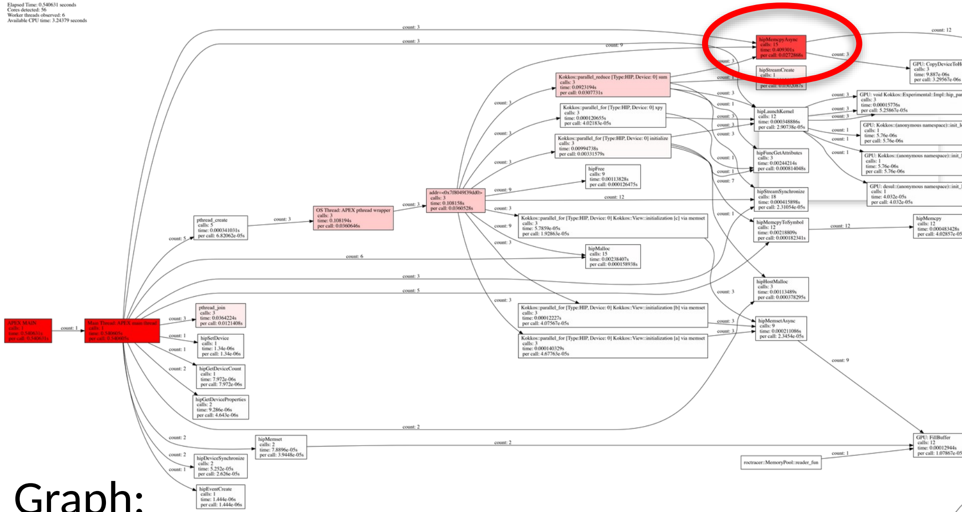
pthread_create
calls: 1.000000
time: 2.4e-05

OS Thread: APEX pthread wrapper
calls: 1.000000
time: 2.3e-05

void* std::__1::__thread_proxy<>(void*)
calls: 1.000000
time: 1.2e-05

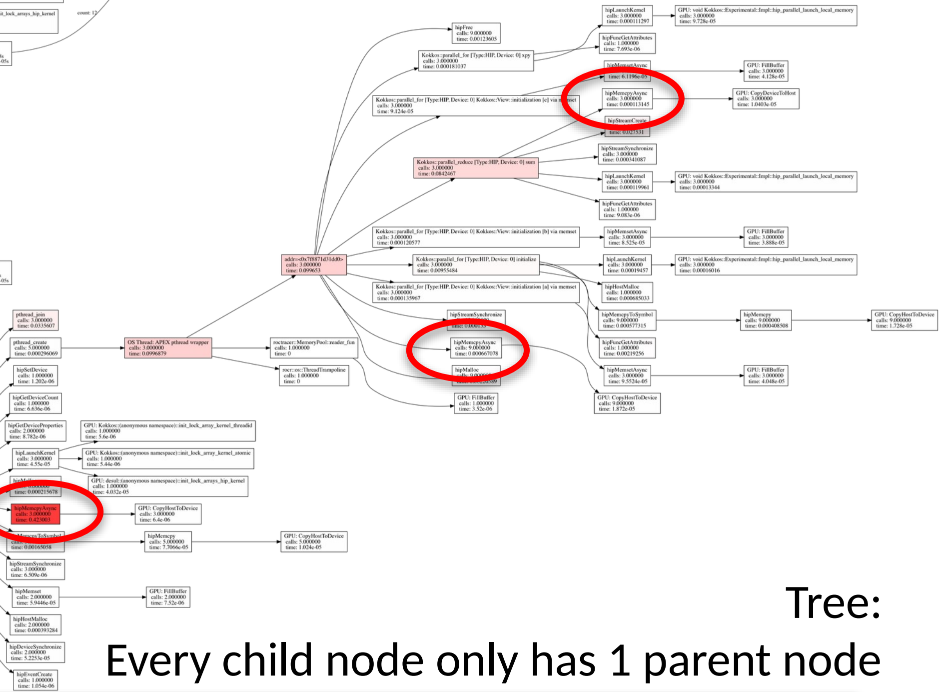
fib(int, std::__1::shared_ptr<apex::task_wrapper>)
calls: 1.000000
time: 1e-06

Kokkos Example



Graph:

Every child node can have *n* parent nodes



Tree:

Every child node only has 1 parent node

Hardware Counters

- APEX is integrated with **PAPI** (Performance Application Programming Interface) https://icl.utk.edu/papi/
- Portable access to native hardware counters
  - CPU (cache misses, FLOPs, instructions, stalls…)
  - GPU (cache misses, FLOPs, instructions, stalls…)
  - Off-core (permission dependent)
  - Node/OS health (LM Sensors, network)
  - Power/energy (RAPL, powercap…)
  - Filesystems
- HW counters collected with timers, or periodically
- APEX also monitors some/other counters natively

GPU Measurement

CUDA

- Support provided with **CUPTI** library
- Monitoring support provided with **NVML** library
- Hardware counters provided **CUPTI** and through **PAPI**
- Host callback and device activity dependencies linked using **correlation IDs**

HIP/ROCm

- Support provided by **Roctracer** and **Rocprofiler** libraries
- Monitoring support provided by **rocm-smi** library
- Hardware counters provided **CUPTI** and through **PAPI**
- Host callback and device activity dependencies linked using **correlation IDs**

Intel SYCL/DPC++/OneAPI support in development…

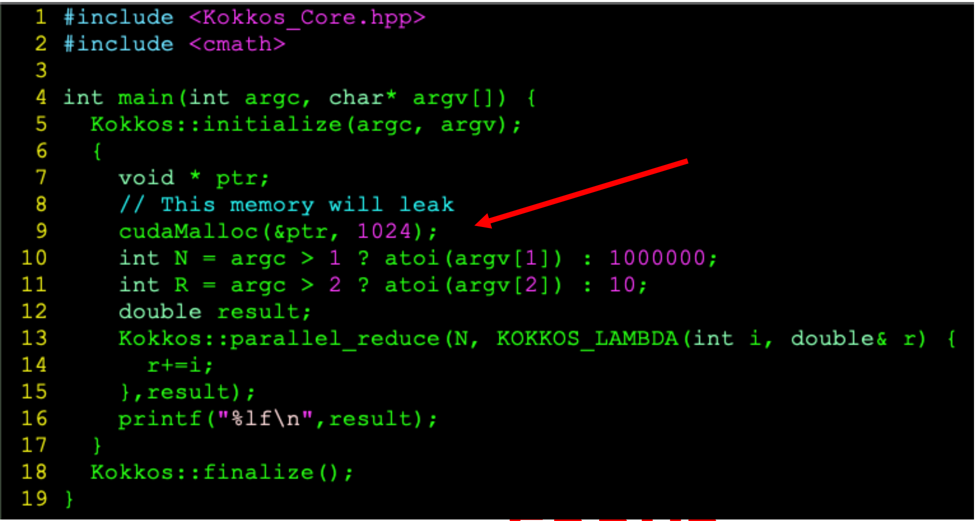
GPU Memory Tracking

- For both CUDA and HIP, when memory is allocated or freed through the cuda/hip API, APEX captures:
  - Allocation type (host/gpu)
  - Bytes allocated
  - thread ID that requested it
  - Address of allocated memory
  - Backtrace from when allocation happened
- At application exit, any leaked allocations are reported to the user, similar to **`cuda-memcheck`**
- …but finds leaks that it doesn’t
- Counters saved by APEX (bytes allocated/freed/total – see figure)

Motivating paper: Wei, Weile, et al. "*Memory Reduction Using a Ring Abstraction Over GPU RDMA for Distributed Quantum Monte Carlo Solver*." Proceedings of the Platform for Advanced Scientific Computing Conference, 2021.

(b) Distributed G_t^d method with sub-ring size of three.

Example Program: memory error

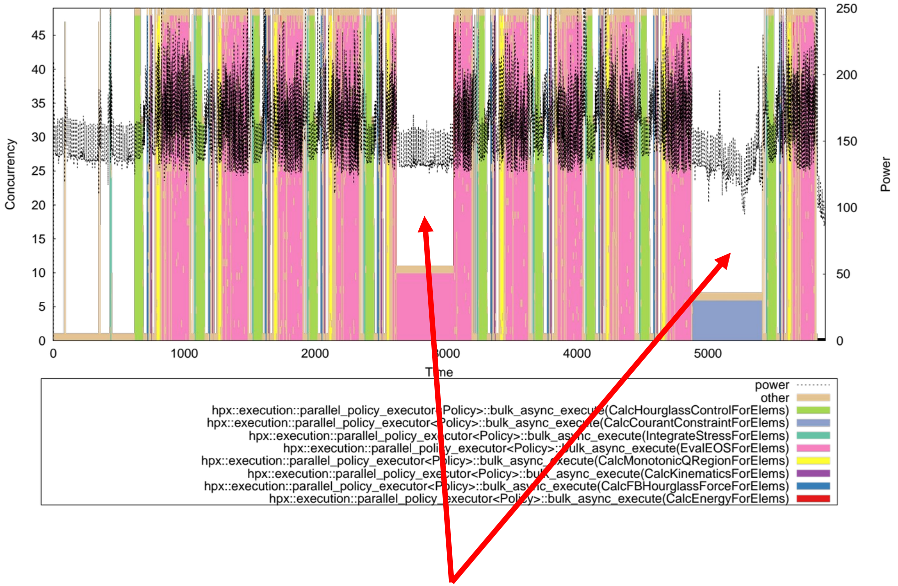


```cpp
#include <Kokkos_Core.hpp>
#include <cmath>

int main(int argc, char* argv[]) {
  Kokkos::initialize(argc, argv);
  {
    void * ptr;
    // This memory will leak
    cudaMalloc(&ptr, 1024);
    int N = argc > 1 ? atoi(argv[1]) : 1000000;
    int R = argc > 2 ? atoi(argv[2]) : 10;
    double result;
    Kokkos::parallel_reduce(N, KOKKOS_LAMBDA(int i, double& r) {
      r+=i;
    },result);
    printf("%lf\n",result);
  }
  Kokkos::finalize();
}
```

```
========= ERROR
SUMMARY: 0 errors
```

```
1024 bytes leaked at 0x1465937ea400 from task
cudaMalloc on tid 0 with backtrace:
gpu_device_malloc
addr=<0x1465fa0f409b> [{(unknown)}
{0x1465fa0f409b}]
addr=<0x1465fa0f43b7> [{(unknown)}
{0x1465fa0f43b7}]
addr=<0x1465fa0f6c1c> [{(unknown)}
{0x1465fa0f6c1c}]
addr=<0x1465fe5e3143> [{(unknown)}
{0x1465fe5e3143}]
addr=<0x1465fdfb247b> [{(unknown)}
{0x1465fdfb247b}]
main [{/home/khuck/polaris/test/test.cpp}
{9,0}]
__libc_start_main [{/lib64/libc-2.31.so}
{0x1465fb4e434d}]
_start [{/home/abuild/rpmbuild/BUILD/glibc-
2.31/csu/../sysdeps/x86_64/start.S} {122,0}]
```

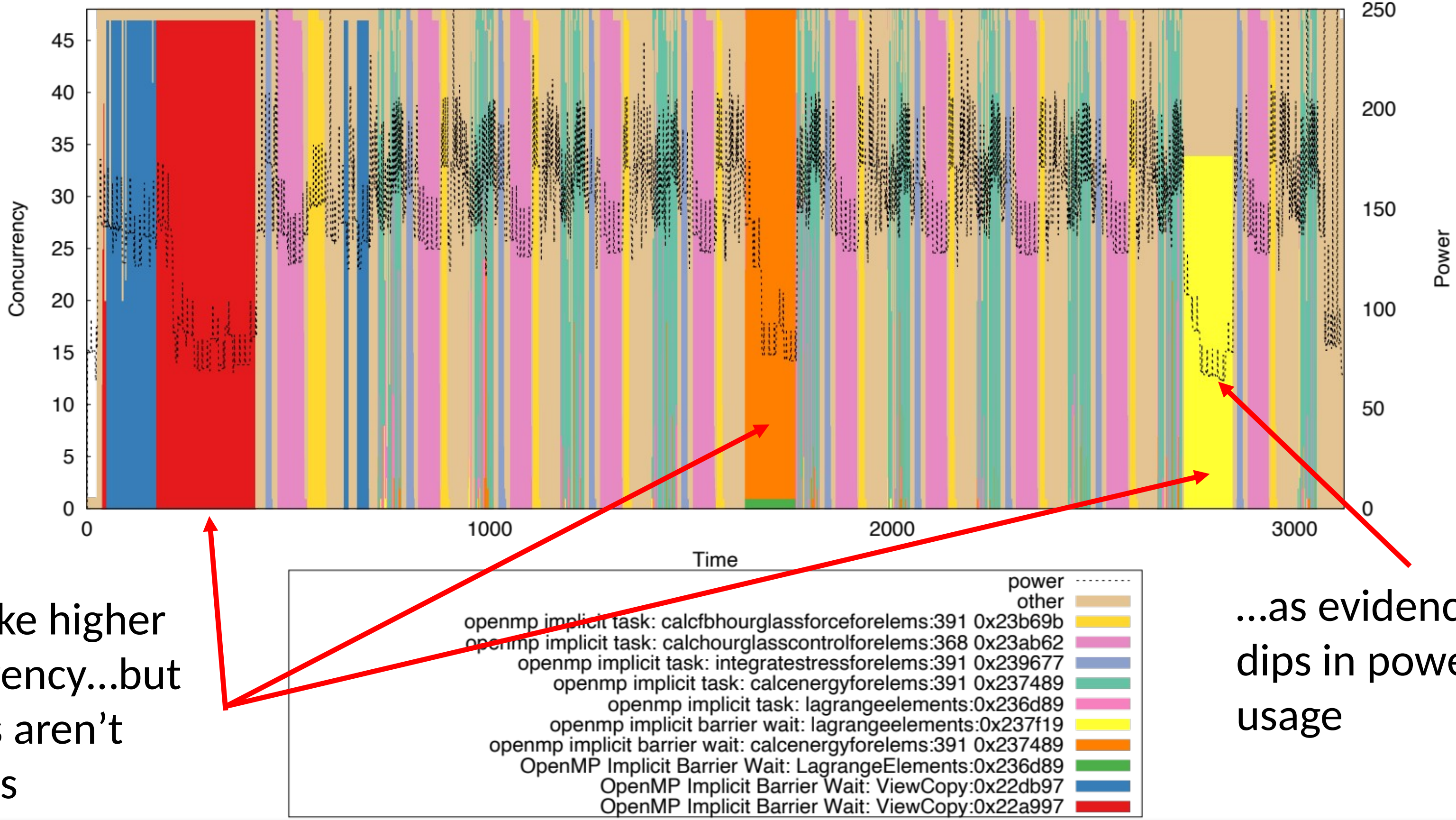
Concurrency Tracking



Helps identify regions of low concurrency

- Periodically sample all the currently executing tasks (timers, really)
- Aggregate across N timer types/names
- Example shown:
  - Kokkos Lulesh with HPX back end
  - Sampled 200 times per second
  - 10 iterations, size 256

Concurrency: OpenMP back end

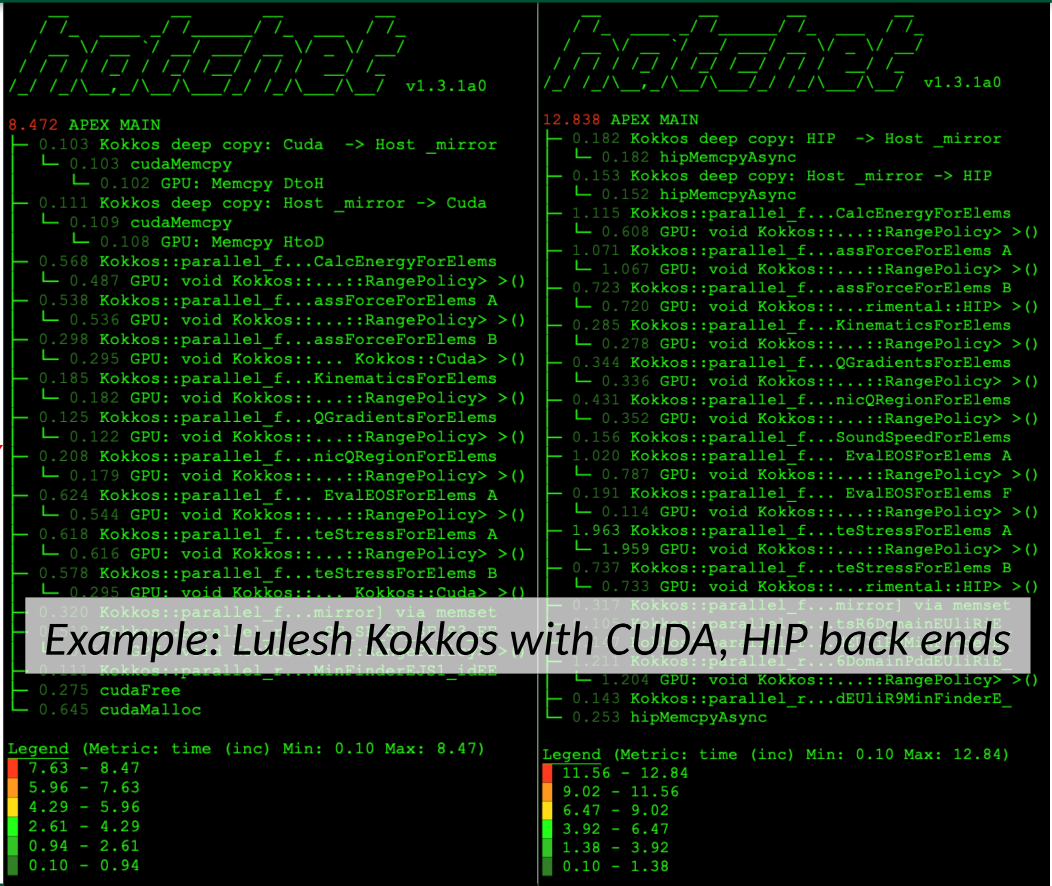


Looks like higher concurrency…but barriers aren't progress

…as evidenced by dips in power usage

Profile Formats

- **All timer & counter data**
- **Flat profile:**
  - Text summary to screen – all ranks merged with MPI or HPX at end of execution
  - CSV (for Python ingestion)
  - TAU Profiles (ParaProf)
- **Task graphs/trees:**
  - Hatchet-like JSON (still working on importer library for Hatchet) https://hatchet.readthedocs.io
  - Graphviz dot files
  - Txt files (similar to Trilinos profiler output)



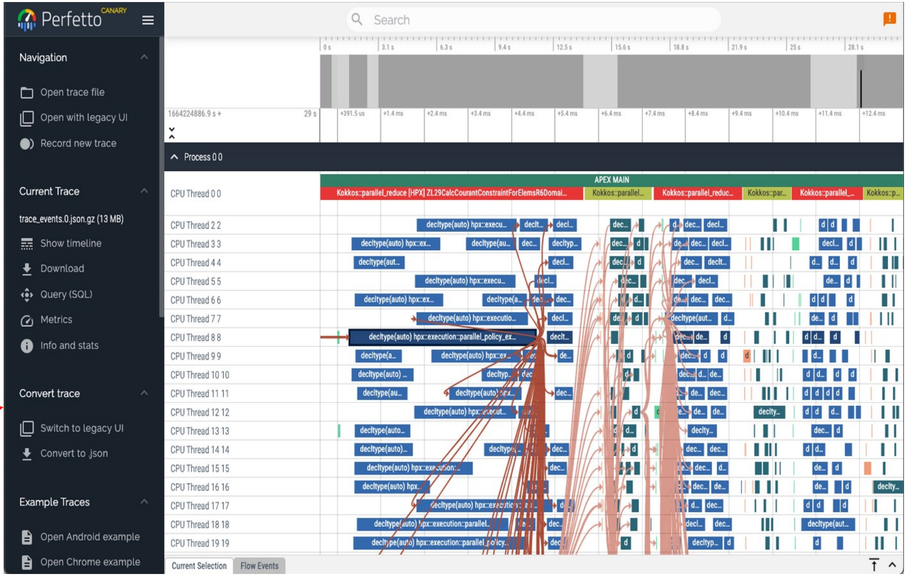
```
    __          __       __         __
   / /_  ____ _/ /______/ /_  ___  / /_
  / __ \/ __ `/ __/ ___/ __ \/ _ \/ __/
 / / / / /_/ / /_/ /__/ / / /  __/ /_
/_/ /_/\__,_/\__/\___/_/ /_/\___/\__/  v1.3.1a0

8.472 APEX MAIN
├─ 0.103 Kokkos deep copy: Cuda  -> Host _mirror
│  └─ 0.103 cudaMemcpy
│     └─ 0.102 GPU: Memcpy DtoH
├─ 0.111 Kokkos deep copy: Host _mirror -> Cuda
│  └─ 0.109 cudaMemcpy
│     └─ 0.108 GPU: Memcpy HtoD
├─ 0.568 Kokkos::parallel_f...CalcEnergyForElems
│  └─ 0.487 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.538 Kokkos::parallel_f...assForceForElems A
│  └─ 0.536 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.298 Kokkos::parallel_f...assForceForElems B
│  └─ 0.295 GPU: void Kokkos::... Kokkos::Cuda> >()
├─ 0.185 Kokkos::parallel_f...KinematicsForElems
│  └─ 0.182 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.125 Kokkos::parallel_f...QGradientsForElems
│  └─ 0.122 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.208 Kokkos::parallel_f...nicQRegionForElems
│  └─ 0.179 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.624 Kokkos::parallel_f... EvalEOSForElems A
│  └─ 0.544 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.618 Kokkos::parallel_f...teStressForElems A
│  └─ 0.616 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.578 Kokkos::parallel_f...teStressForElems B
│  └─ 0.295 GPU: void Kokkos::... Kokkos::Cuda> >()
├─ 0.320 Kokkos::parallel_f...mirror] via memset
├─ [illegible]
│  └─ [illegible] GPU: void Kokkos::...::RangePolicy> >()
├─ 0.111 Kokkos::parallel_r...MinFinderEJS1_idEE
├─ 0.275 cudaFree
└─ 0.645 cudaMalloc

Legend (Metric: time (inc) Min: 0.10 Max: 8.47)
🟥 7.63 - 8.47
🟧 5.96 - 7.63
🟨 4.29 - 5.96
🟩 2.61 - 4.29
🟩 0.94 - 2.61
🟩 0.10 - 0.94
```

```
    __          __       __         __
   / /_  ____ _/ /______/ /_  ___  / /_
  / __ \/ __ `/ __/ ___/ __ \/ _ \/ __/
 / / / / /_/ / /_/ /__/ / / /  __/ /_
/_/ /_/\__,_/\__/\___/_/ /_/\___/\__/  v1.3.1a0

12.838 APEX MAIN
├─ 0.182 Kokkos deep copy: HIP  -> Host _mirror
│  └─ 0.182 hipMemcpyAsync
├─ 0.153 Kokkos deep copy: Host _mirror -> HIP
│  └─ 0.152 hipMemcpyAsync
├─ 1.115 Kokkos::parallel_f...CalcEnergyForElems
│  └─ 0.608 GPU: void Kokkos::...::RangePolicy> >()
├─ 1.071 Kokkos::parallel_f...assForceForElems A
│  └─ 1.067 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.723 Kokkos::parallel_f...assForceForElems B
│  └─ 0.720 GPU: void Kokkos::...rimental::HIP> >()
├─ 0.285 Kokkos::parallel_f...KinematicsForElems
│  └─ 0.278 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.344 Kokkos::parallel_f...QGradientsForElems
│  └─ 0.336 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.431 Kokkos::parallel_f...nicQRegionForElems
│  └─ 0.352 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.156 Kokkos::parallel_f...SoundSpeedForElems
├─ 1.020 Kokkos::parallel_f... EvalEOSForElems A
│  └─ 0.787 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.191 Kokkos::parallel_f... EvalEOSForElems F
│  └─ 0.114 GPU: void Kokkos::...::RangePolicy> >()
├─ 1.963 Kokkos::parallel_f...teStressForElems A
│  └─ 1.959 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.737 Kokkos::parallel_f...teStressForElems B
│  └─ 0.733 GPU: void Kokkos::...rimental::HIP> >()
├─ 0.317 Kokkos::parallel_f...mirror] via memset
├─ 0.105 Kokkos::parallel_r...tsR6DomainEUliRiE
├─ [illegible]
├─ 1.211 Kokkos::parallel_r...6DomainPddEUliRiE_
│  └─ 1.204 GPU: void Kokkos::...::RangePolicy> >()
├─ 0.143 Kokkos::parallel_r...dEUliR9MinFinderE_
└─ 0.253 hipMemcpyAsync

Legend (Metric: time (inc) Min: 0.10 Max: 12.84)
🟥 11.56 - 12.84
🟧 9.02 - 11.56
🟨 6.47 - 9.02
🟩 3.92 - 6.47
🟩 1.38 - 3.92
🟩 0.10 - 1.38
```

Example: Lulesh Kokkos with CUDA, HIP back ends

Trace Formats

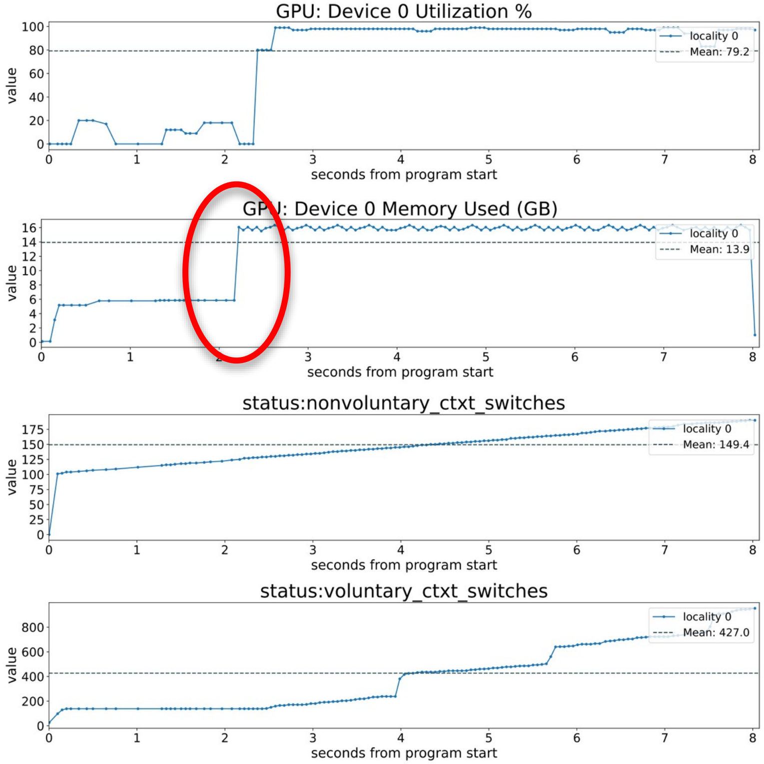
- **OTF2** up to v 2.3
  - 3.0 has API changes, APEX hasn't been updated yet
  - Visualized with Vampir or Traveler / JetLag
  - Problems with asynchrony, overlapping timers, high thread counts
- **Google Trace Events Format**
  - JSON support only (native support coming soon)
  - Visualized with Perfetto
  - Some scaling issues (memory limit of web browser)
  - Handles asynchrony, overlapping timers just fine



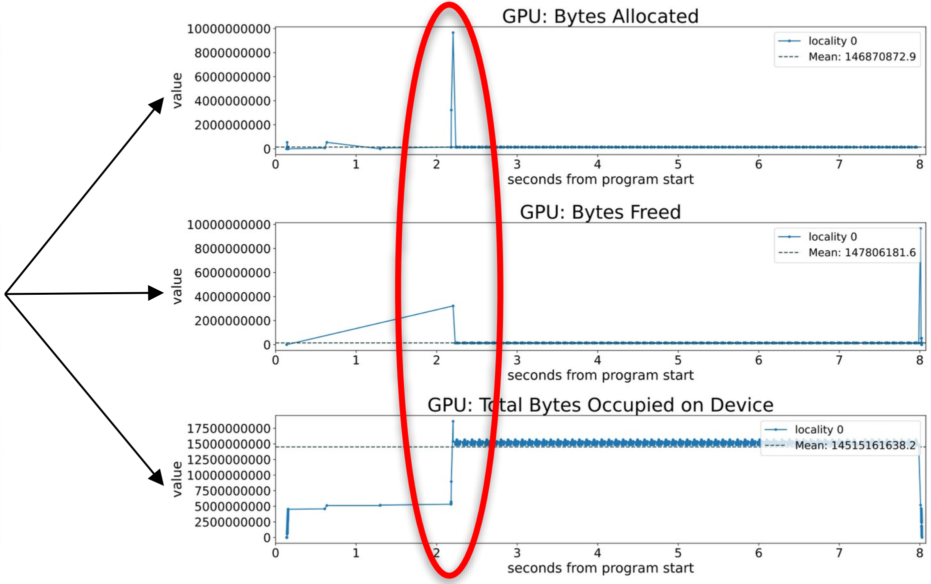
Example: Lulesh Kokkos with HPX back end

Scatterplots

Monitoring Data (periodic)



Progress Data (events)



Example: Lulesh Kokkos with CUDA back end

Supported Programming Models

- HPX
- POSIX / C++ threads
- OpenMP/OpenACC
- GPU offload: CUDA, HIP, OpenMP target
- Abstractions: Kokkos, Raja
- MPI (subset)
- In development: Intel Level0/OneAPI GPU support, StarPU

POSIX / C++ Threads

- APEX wraps the pthread_create() call:
  - Wraps target function with a proxy function
  - Times target function
  - Captures task dependency hierarchy between parent, child
- Provides support for C++ thread activity too
  - std::thread
  - std::async
  - Only on POSIX compliant systems

OpenMP and OpenACC

- OpenMP 5.0 included OMP-Tools (OMPT) API
  - Callbacks, query functions, sampling states
  - Buffer processing for **target offload** asynchronous activity
  - Tested with AMD Clang 5.0+, NVHPC 22.7+, Intel OneAPI 2022
- OpenACC profiling callbacks to intercept entry/exit of all OpenACC routines
  - CUDA/CUPTI provides support for device activity

Kokkos and Raja

- C++ abstraction models for **performance portability**
- 1 source code implementation to target different architectural / model back ends
  - Serial, Pthreads, OpenMP, OpenACC, CUDA, HIP, SYCL, etc.
- Both provide **host-side profiling callbacks** for tool support
- Kokkos includes a prototype "tuning" interface for tools to hook utilize at runtime
  - APEX has implemented tuning policies and tested with CUDA back end tuning Range, MDRange, Team policies

Examples: Lulesh with Kokkos

- https://github.com/kokkos/kokkos-miniapps
- Tested lulesh-2.0 mini-app (https://asc.llnl.gov/codes/proxy-apps/lulesh)
  - HPX
  - OpenMP
  - CUDA
  - HIP

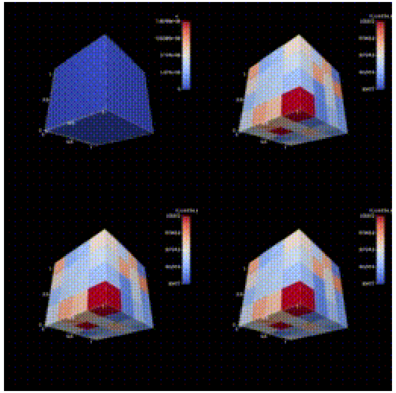
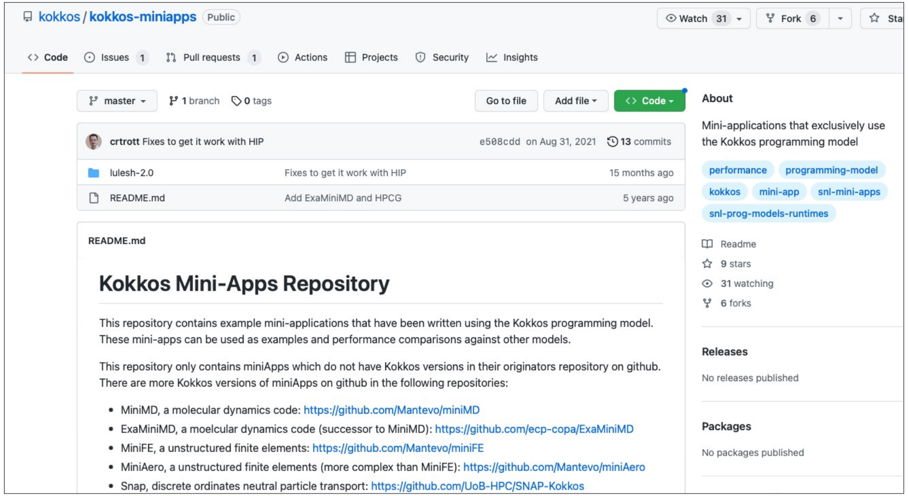


Figure: LULESH 2.0.3 executed with 64 MPI ranks, measured by TAU. Time spent in main loop rendered every 100 timesteps by Alpine-Ascent. Clockwise from upper left: Computed energy, accumulated time in main loop, time in main loop during last 100 timesteps, Δ time in main loop from previous 100 time steps. Source: Malony, et al. "When Parallel Performance Measurement and Analysis Meets In Situ Analytics and Visualization." Parallel Computing: Technology Trends. IOS Press, 2020. 521-530.



Mini-applications that exclusively use the Kokkos programming model

No releases published

No packages published

crtrott Fixes to get it work with HIP e508cdd on Aug 31, 2021 13 commits

lulesh-2.0	Fixes to get it work with HIP	15 months ago
README.md	Add ExaMiniMD and HPCG	5 years ago

README.md

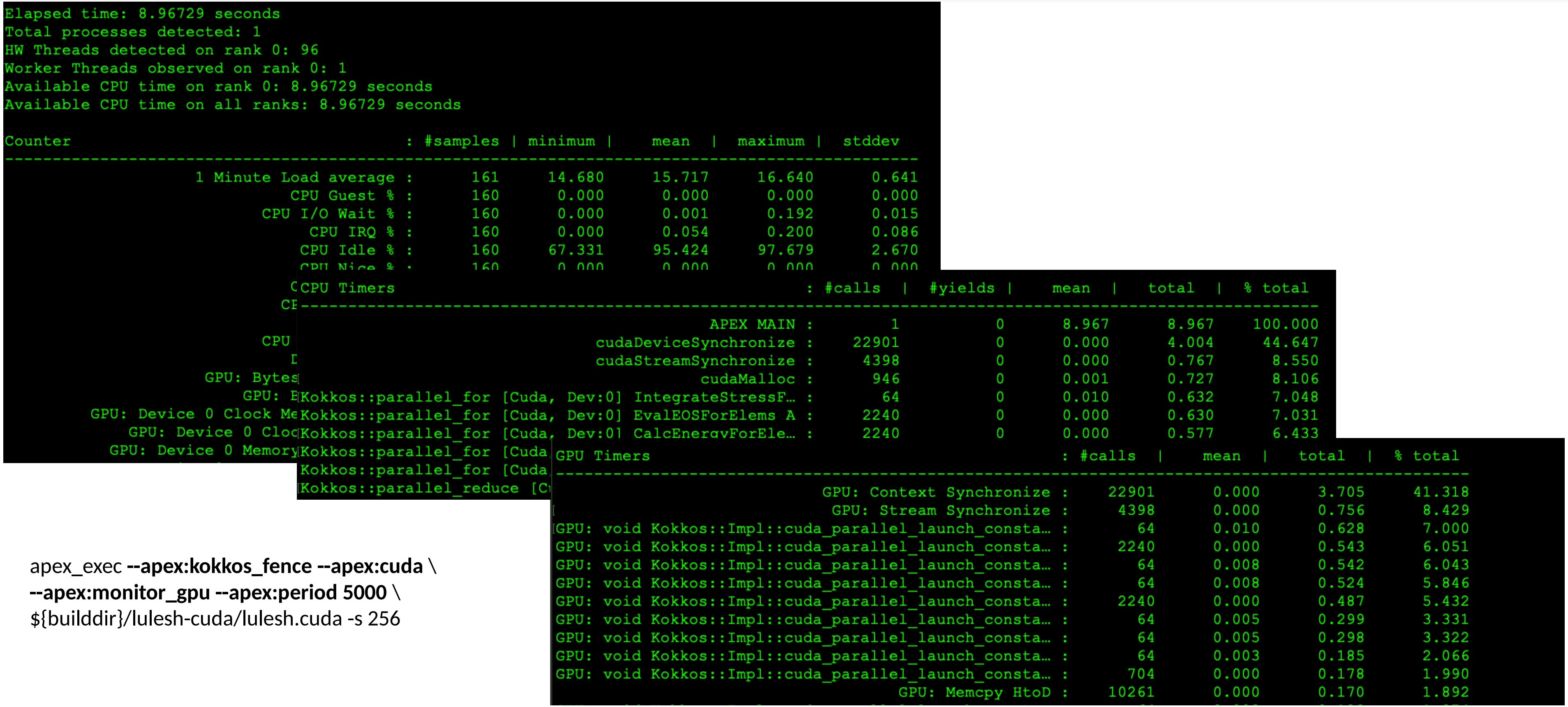
Kokkos Mini-Apps Repository

This repository contains example mini-applications that have been written using the Kokkos programming model. These mini-apps can be used as examples and performance comparisons against other models.

This repository only contains miniApps which do not have Kokkos versions in their originators repository on github. There are more Kokkos versions of miniApps on github in the following repositories:

- MiniMD, a molecular dynamics code: https://github.com/Mantevo/miniMD
- ExaMiniMD, a moelcular dynamics code (successor to MiniMD): https://github.com/ecp-copa/ExaMiniMD
- MiniFE, a unstructured finite elements: https://github.com/Mantevo/miniFE
- MiniAero, a unstructured finite elements (more complex than MiniFE): https://github.com/Mantevo/miniAero
- Snap, discrete ordinates neutral particle transport: https://github.com/UoB-HPC/SNAP-Kokkos

Lulesh: CUDA back end



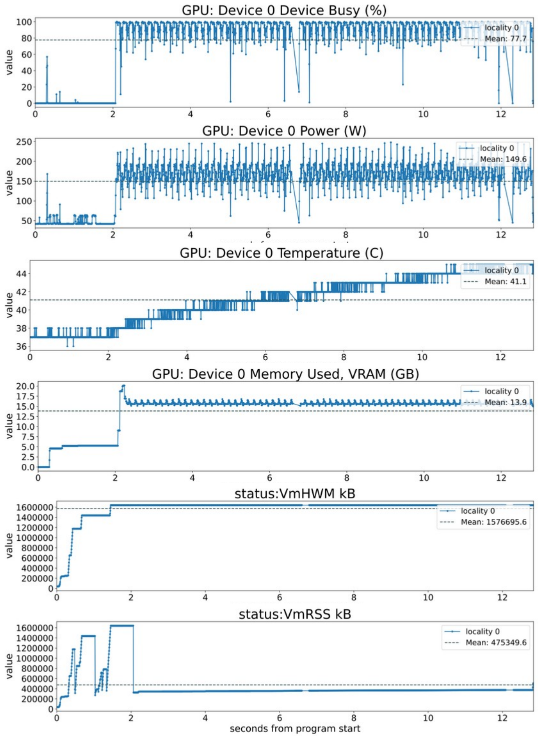
```
Elapsed time: 8.96729 seconds
Total processes detected: 1
HW Threads detected on rank 0: 96
Worker Threads observed on rank 0: 1
Available CPU time on rank 0: 8.96729 seconds
Available CPU time on all ranks: 8.96729 seconds

Counter                                   : #samples | minimum |    mean  |  maximum |  stddev
------------------------------------------------------------------------------------------------
                    1 Minute Load average :      161     14.680     15.717     16.640      0.641
                              CPU Guest % :      160      0.000      0.000      0.000      0.000
                            CPU I/O Wait % :     160      0.000      0.001      0.192      0.015
                                CPU IRQ % :      160      0.000      0.054      0.200      0.086
                               CPU Idle % :      160     67.331     95.424     97.679      2.670
                               CPU Nice % :      160      0.000      0.000      0.000      0.000
                               C
                              CP

                            CPU
                               D
                     GPU: Bytes
                         GPU: B
         GPU: Device 0 Clock Me
             GPU: Device 0 Cloc
           GPU: Device 0 Memory
```

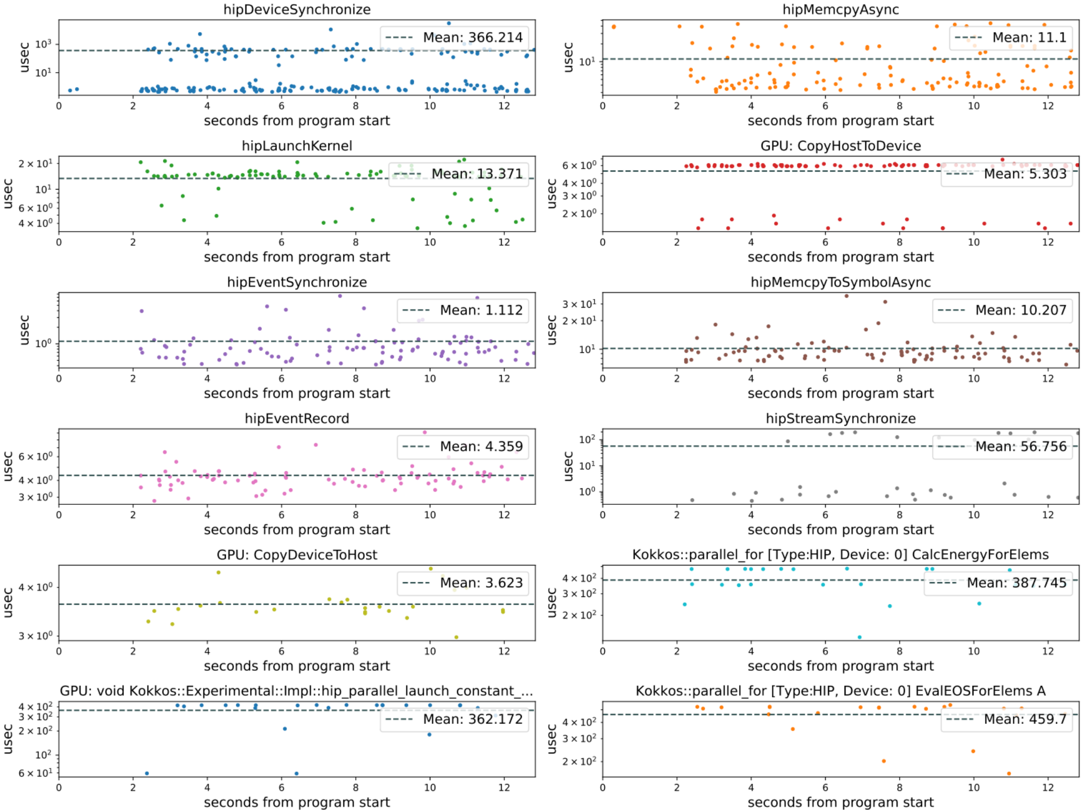
```
CPU Timers                                                    : #calls    | #yields |    mean   |   total   |  % total
-----------------------------------------------------------------------------------------------------------------------
                                                    APEX MAIN :         1          0      8.967       8.967    100.000
                                      cudaDeviceSynchronize :     22901          0      0.000       4.004     44.647
                                      cudaStreamSynchronize :      4398          0      0.000       0.767      8.550
                                                 cudaMalloc :       946          0      0.001       0.727      8.106
Kokkos::parallel_for [Cuda, Dev:0] IntegrateStressF… :        64          0      0.010       0.632      7.048
Kokkos::parallel_for [Cuda, Dev:0] EvalEOSForElems A :      2240          0      0.000       0.630      7.031
Kokkos::parallel_for [Cuda, Dev:0] CalcEnergyForEle… :      2240          0      0.000       0.577      6.433
Kokkos::parallel_for [Cuda
Kokkos::parallel_for [Cuda
Kokkos::parallel_reduce [Cu
```

```
GPU Timers                                                  : #calls   |    mean   |   total   |  % total
-----------------------------------------------------------------------------------------------------------
                                  GPU: Context Synchronize :    22901        0.000       3.705     41.318
                                   GPU: Stream Synchronize :     4398        0.000       0.756      8.429
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.010       0.628      7.000
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :     2240        0.000       0.543      6.051
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.008       0.542      6.043
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.008       0.524      5.846
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :     2240        0.000       0.487      5.432
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.005       0.299      3.331
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.005       0.298      3.322
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :       64        0.003       0.185      2.066
GPU: void Kokkos::Impl::cuda_parallel_launch_consta… :      704        0.000       0.178      1.990
                                        GPU: Memcpy HtoD :    10261        0.000       0.170      1.892
```

apex_exec **--apex:kokkos_fence --apex:cuda** \
--apex:monitor_gpu --apex:period 5000 \
${builddir}/lulesh-cuda/lulesh.cuda -s 256

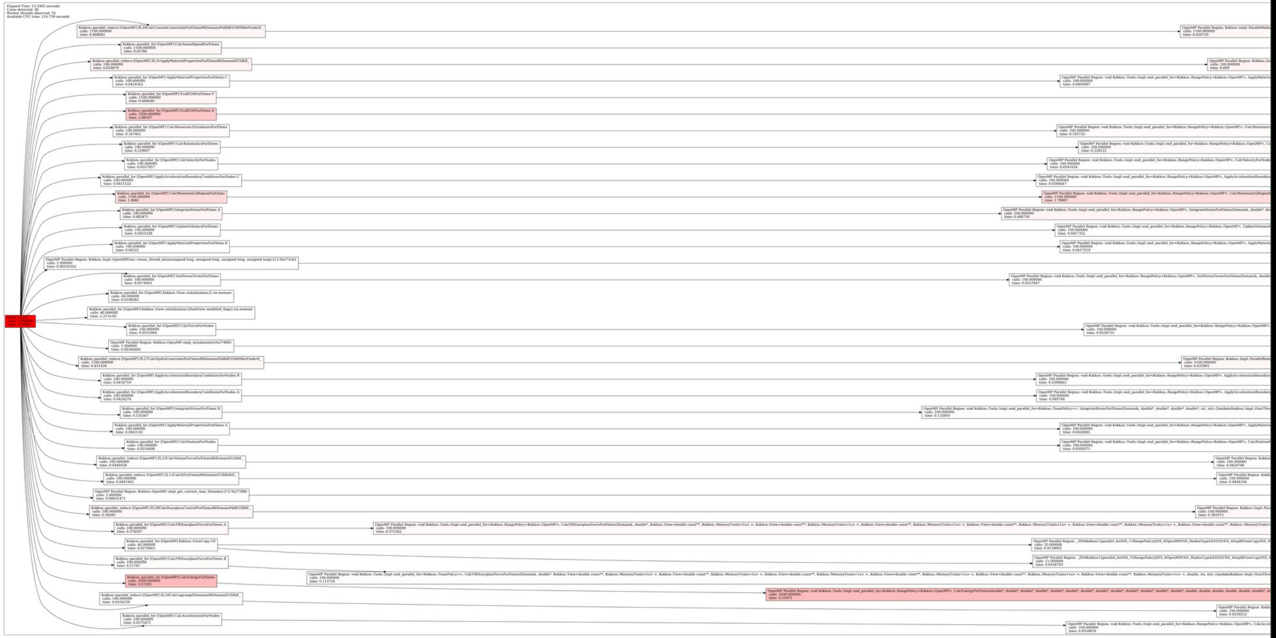
Lulesh: HIP back end

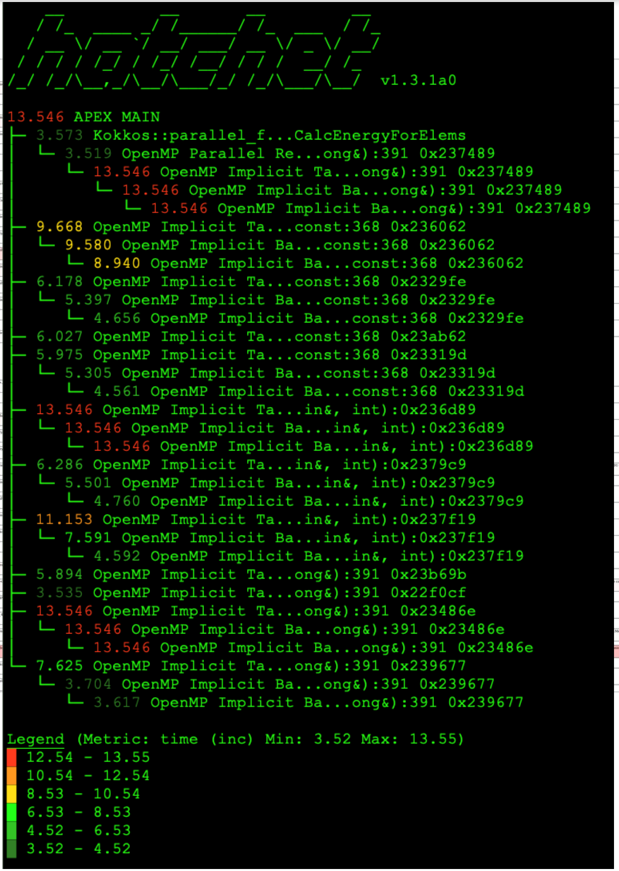




```
apex_exec --apex:kokkos_fence --apex:scatter \
--apex:hip --apex:monitor_gpu \
--apex:period 5000 ${builddir}/lulesh-hip/lulesh.hip -s 256
```

Lulesh: OpenMP back end

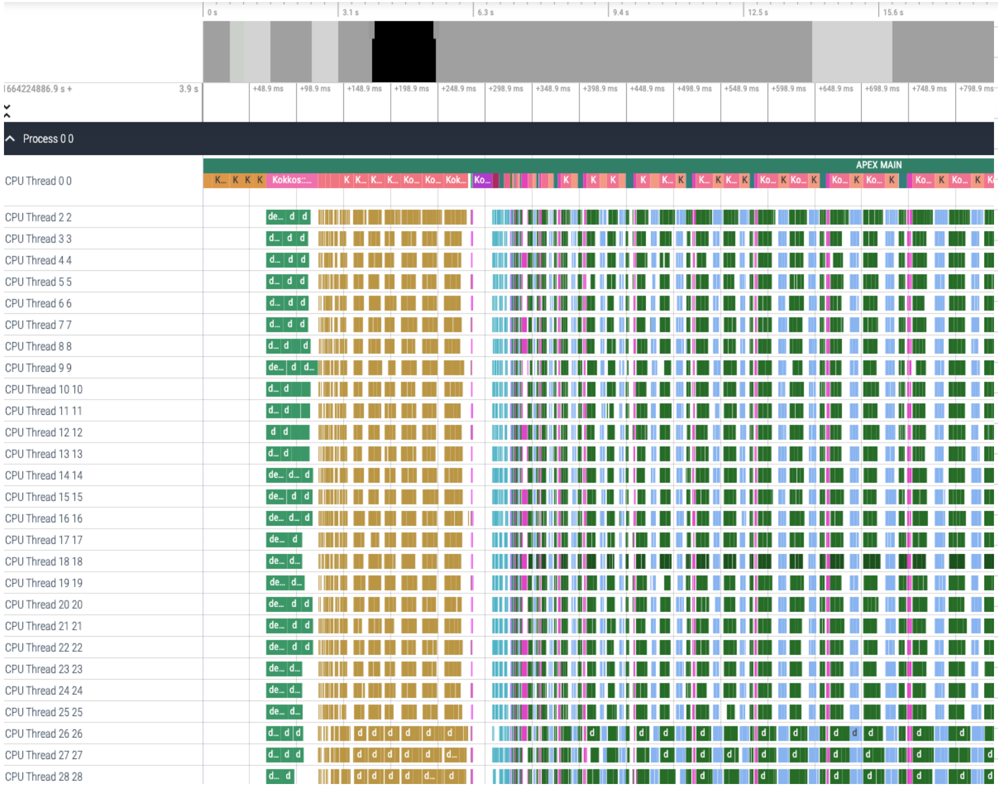


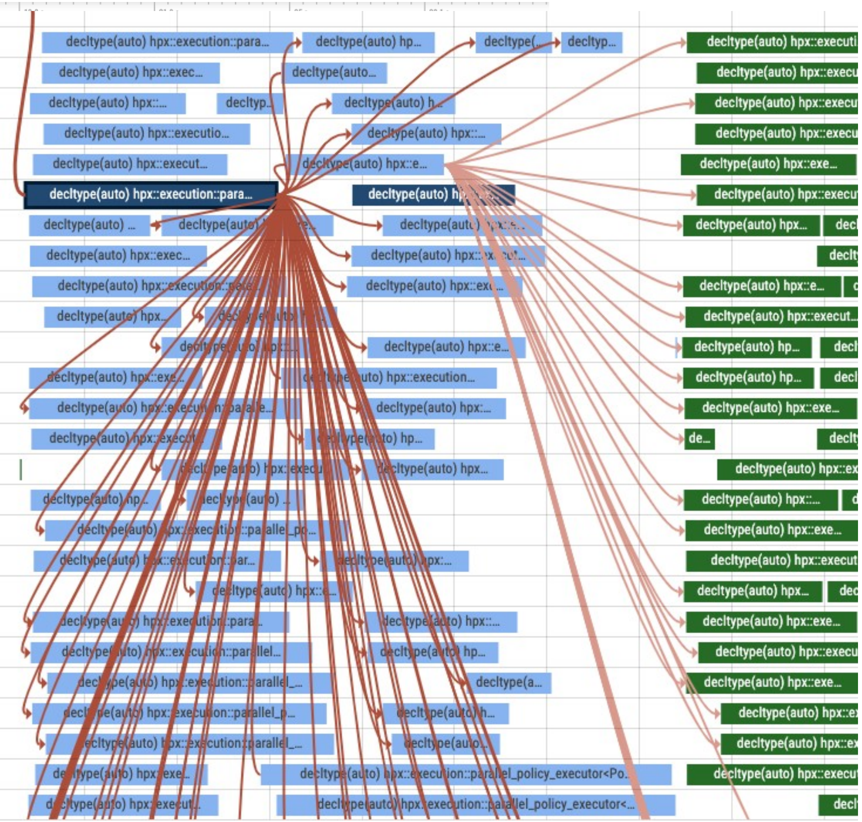


```
    __          __       __         __
   / /_  ____ _/ /______/ /_  ___  / /_
  / __ \/ __ `/ __/ ___/ __ \/ _ \/ __/
 / / / / /_/ / /_/ /__/ / / /  __/ /_
/_/ /_/\__,_/\__/\___/_/ /_/\___/\__/  v1.3.1a0

13.546 APEX MAIN
├─ 3.573 Kokkos::parallel_f...CalcEnergyForElems
│  └─ 3.519 OpenMP Parallel Re...ong&):391 0x237489
│     └─ 13.546 OpenMP Implicit Ta...ong&):391 0x237489
│        └─ 13.546 OpenMP Implicit Ba...ong&):391 0x237489
│           └─ 13.546 OpenMP Implicit Ba...ong&):391 0x237489
├─ 9.668 OpenMP Implicit Ta...const:368 0x236062
│  └─ 9.580 OpenMP Implicit Ba...const:368 0x236062
│     └─ 8.940 OpenMP Implicit Ba...const:368 0x236062
├─ 6.178 OpenMP Implicit Ta...const:368 0x2329fe
│  └─ 5.397 OpenMP Implicit Ba...const:368 0x2329fe
│     └─ 4.656 OpenMP Implicit Ba...const:368 0x2329fe
├─ 6.027 OpenMP Implicit Ta...const:368 0x23ab62
├─ 5.975 OpenMP Implicit Ta...const:368 0x23319d
│  └─ 5.305 OpenMP Implicit Ba...const:368 0x23319d
│     └─ 4.561 OpenMP Implicit Ba...const:368 0x23319d
├─ 13.546 OpenMP Implicit Ta...in&, int):0x236d89
│  └─ 13.546 OpenMP Implicit Ba...in&, int):0x236d89
│     └─ 13.546 OpenMP Implicit Ba...in&, int):0x236d89
├─ 6.286 OpenMP Implicit Ta...in&, int):0x2379c9
│  └─ 5.501 OpenMP Implicit Ba...in&, int):0x2379c9
│     └─ 4.760 OpenMP Implicit Ba...in&, int):0x2379c9
├─ 11.153 OpenMP Implicit Ta...in&, int):0x237f19
│  └─ 7.591 OpenMP Implicit Ba...in&, int):0x237f19
│     └─ 4.592 OpenMP Implicit Ba...in&, int):0x237f19
├─ 5.894 OpenMP Implicit Ta...ong&):391 0x23b69b
├─ 3.535 OpenMP Implicit Ta...ong&):391 0x22f0cf
├─ 13.546 OpenMP Implicit Ta...ong&):391 0x23486e
│  └─ 13.546 OpenMP Implicit Ba...ong&):391 0x23486e
│     └─ 13.546 OpenMP Implicit Ba...ong&):391 0x23486e
└─ 7.625 OpenMP Implicit Ta...ong&):391 0x239677
   └─ 3.704 OpenMP Implicit Ba...ong&):391 0x239677
      └─ 3.617 OpenMP Implicit Ba...ong&):391 0x239677

Legend (Metric: time (inc) Min: 3.52 Max: 13.55)
█ 12.54 - 13.55
█ 10.54 - 12.54
█ 8.53 - 10.54
█ 6.53 - 8.53
█ 4.52 - 6.53
█ 3.52 - 4.52
```

apex_exec **--apex:kokkos_fence --apex:ompt --apex:ompt_details** \
--apex:tasktree ${builddir}/lulesh-openmp/lulesh.host -s 256 -p -i 10

Lulesh: HPX back end





```
apex_exec --apex:kokkos_fence --apex:gtrace \
${builddir}/lulesh-hpx/lulesh.kk -p -i 10 -s 256
```

Future Work

- **Intel Level0/OneAPI** support has been prototyped, but not yet merged
- **StarPU** support added by Camille Coti, needs additional testing and tighter integration
- Perfetto native trace output
- PowerAPI integration for broader power/energy support
- Kokkos runtime autotuning development

Acknowledgements

Parts of this research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

U.S. DEPARTMENT OF ENERGY | Office of Science

RAPIDS

ECP
EXASCALE COMPUTING PROJECT