Sandia National Laboratories

Exceptional service in the national interest

Using Conditionals to Kokkos Graphs

Jonathan Lifflander, Sandia National Laboratories

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.

U.S. DEPARTMENT OF ENERGY

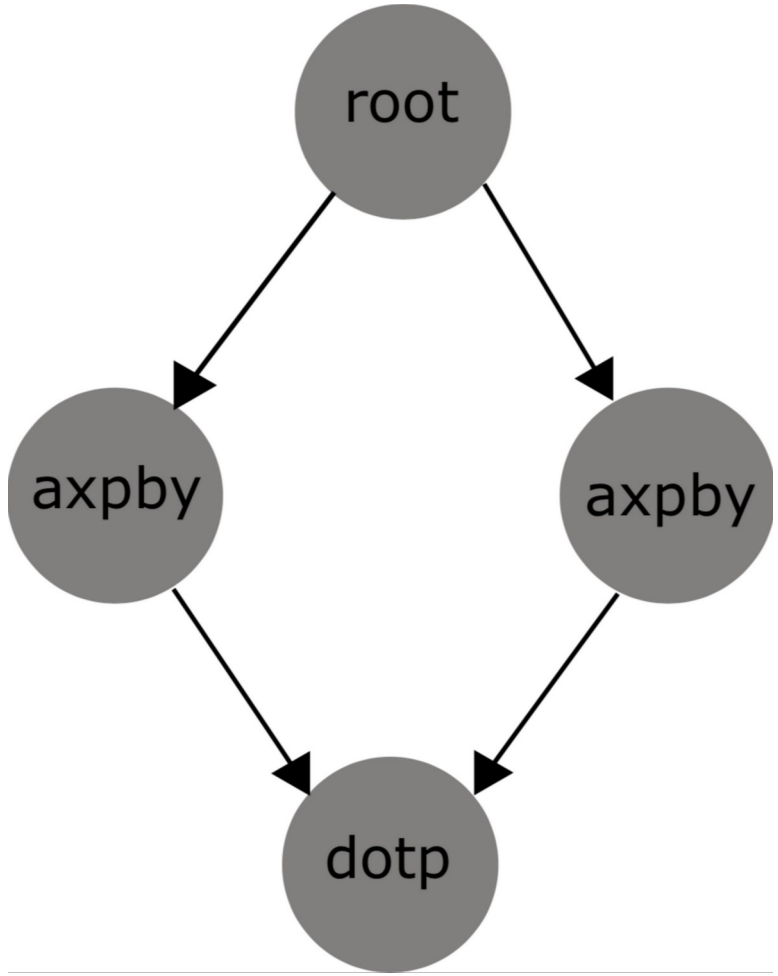
NNSA

Kokkos Graphs

- Provides an abstraction on top of CUDA graphs
- Graph conditionals
  - Variable update vs. control flow on host

Simple Micro-benchmark

- Two AXPBY's followed by a dot product
  - Vary the number of iterations and N
    - Iterations: 10, 100, 1000, 10000
    - N: 1024 – 4194304 (2^10 – 2^22)
- Implemented in vanilla Kokkos and with Kokkos graphs using conditional API
  - Using the same kernels



Micro-benchmark – Vanilla Kokkos

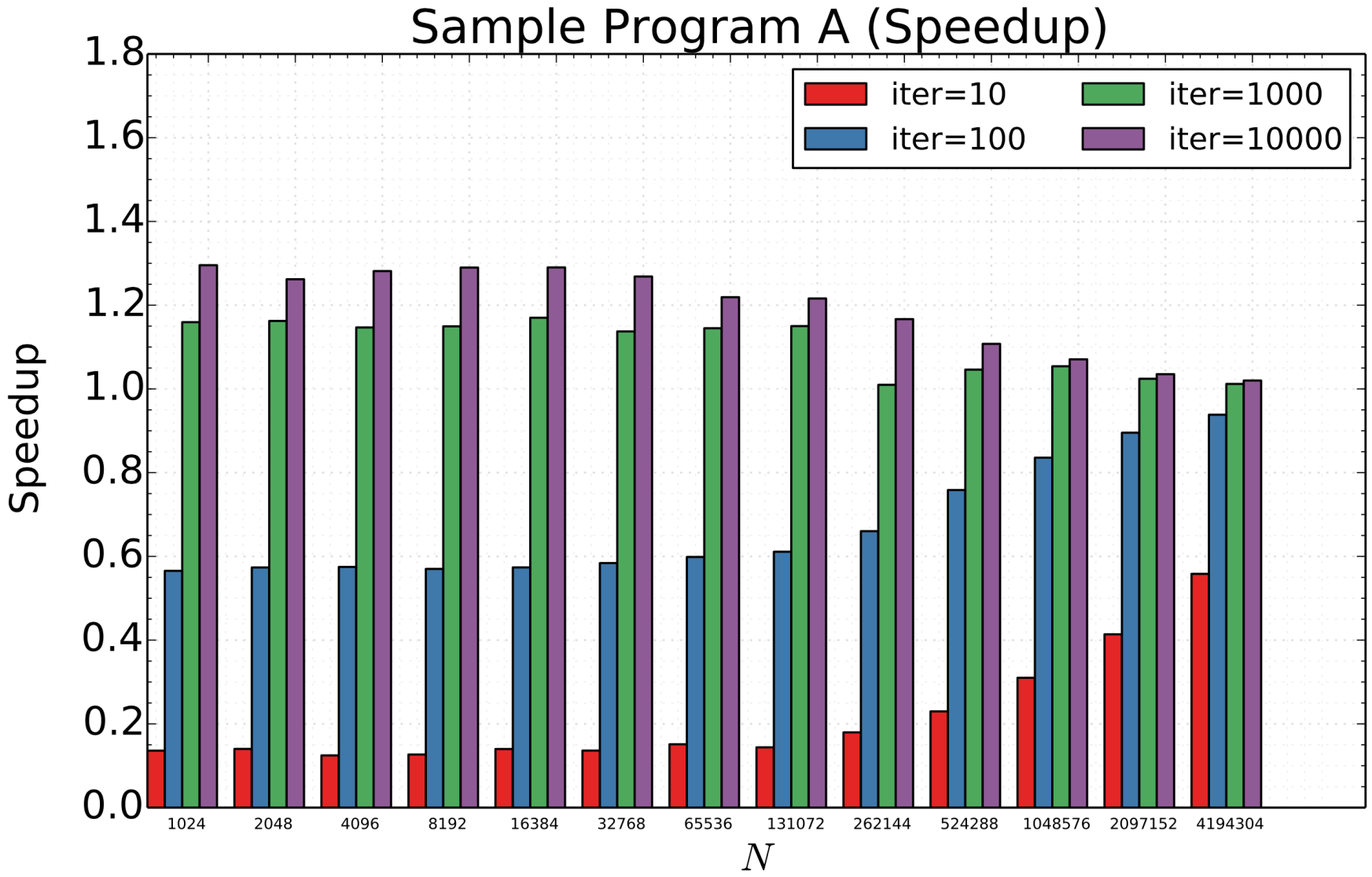
```
while (/*condition on beta value*/) {
  Kokkos::parallel_for(N, Axpby{x, y, alpha, beta});
  Kokkos::parallel_for(N, Axpby{z, y, gamma, beta});
  Kokkos::parallel_reduce(N, Dot{x, z}, beta);
}
```

```
template <class ExecSpace>
struct Axpby {
  Vector_t x, y;
  double alpha;
  double beta;
  KOKKOS_FUNCTION
  void operator()(const int& i) const {
    x(i) = alpha * x(i) + beta * y(i);
  }
};

template <class ExecSpace, class T>
struct Dot {
  Vector_t x, y;

  KOKKOS_FUNCTION
  void operator()(const int& i, T& lsum) const {
    lsum += x(i) * y(i);
  }
};
```

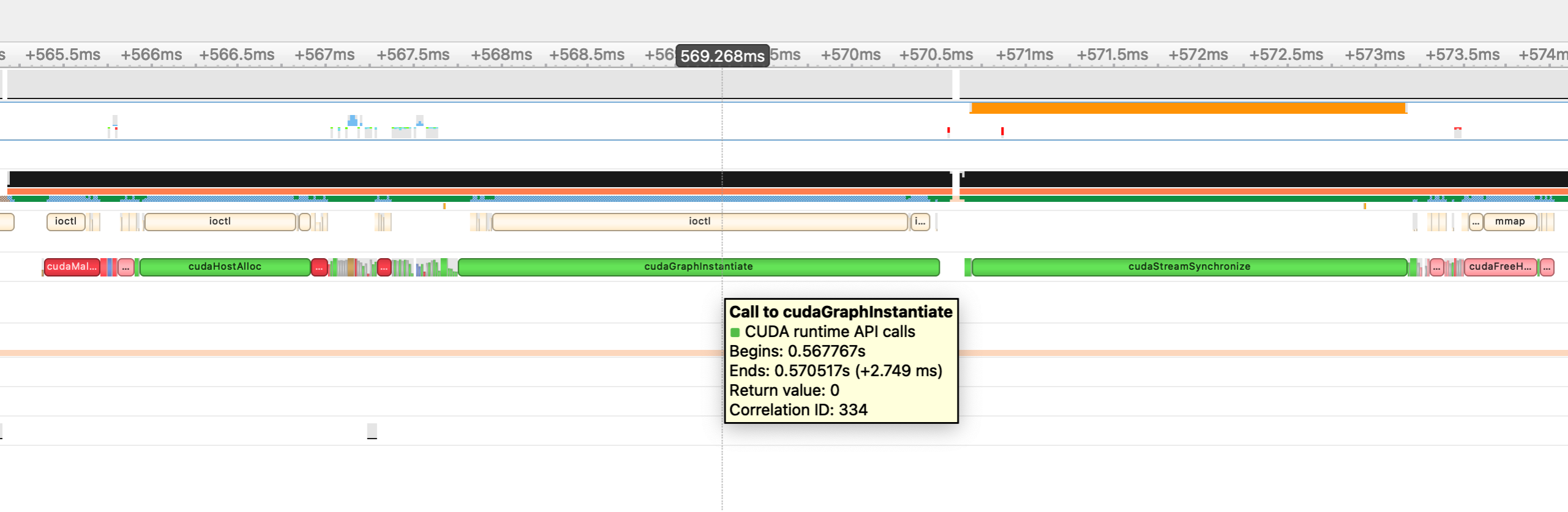
Micro-benchmark – Kokkos Graphs

```
auto iteration = Kokkos::Experimental::create_graph(ex, [&](auto root) {
  auto f_xpy = root.then_parallel_for(N, Axpby {x, y, alpha, beta});
  auto f_zpy = root.then_parallel_for(N, Axpby {z, y, gamma, beta});
  auto ready = when_all(f_xpy, f_zpy);
  ready.then_parallel_reduce(N, Dot{x, z}, beta);
});

auto cond = KOKKOS_LAMBDA() -> bool {
  return beta() < tolerance;
};

auto loop = Kokkos::Experimental::create_while(ex, cond, iteration);
loop.submit();
ex.fence();
```

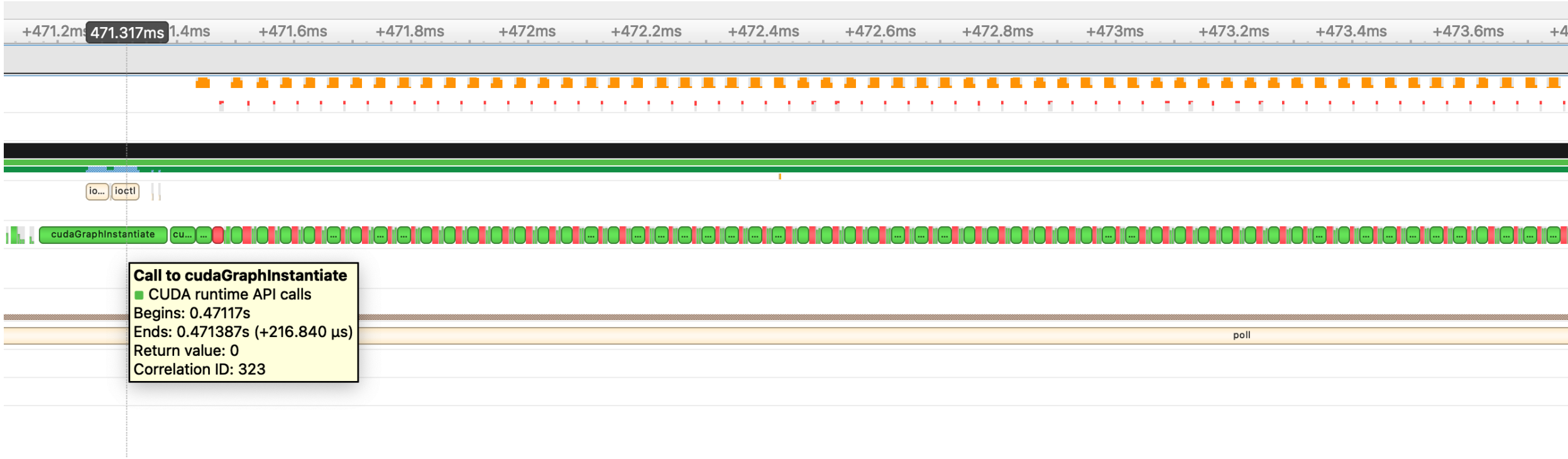
Speedup (with warmup) comparing with conditionals to deep copy



Conditionals – cudaGraphInstantiate takes 2.7ms?



Without Conditionals – cudaGraphInstantiate takes 216us?



Some remarks

- Graph conditionals are beneficial in a limited regime due to instantiation costs
  - I am trying to understand these costs and why they increase significantly by using the conditional API

- Warmup does not seem to affect startup with CUDA 12?