A Tool for Performance Analysis of GPU-Accelerated Applications

Keren Zhou, John Mellor-Crummey

Department of Computer Science

Rice University

Problem

- OpenMP Target, Kokkos, and RAJA generate sophisticated GPU code with many small procedures
  - Complex calling contexts on both CPU and GPU
- Existing performance tools are ill-suited for analyzing such complex kernels because they lack a comprehensive profile view
- At best existing tools only attribute runtime cost to a flat profile view of functions executed on GPUs

Key contribution

- A novel measurement system builds a complete profile view to show performance metrics for GPU-accelerated code for multiple CPU threads
  - Construct calling context trees for GPU programs by analyzing control flow and call graphs
  - Employ wait-free data structures to attribute GPU samples back to heterogenous calling contexts
  - Apportion GPU samples to calling contexts using instruction samples of GPU function calls

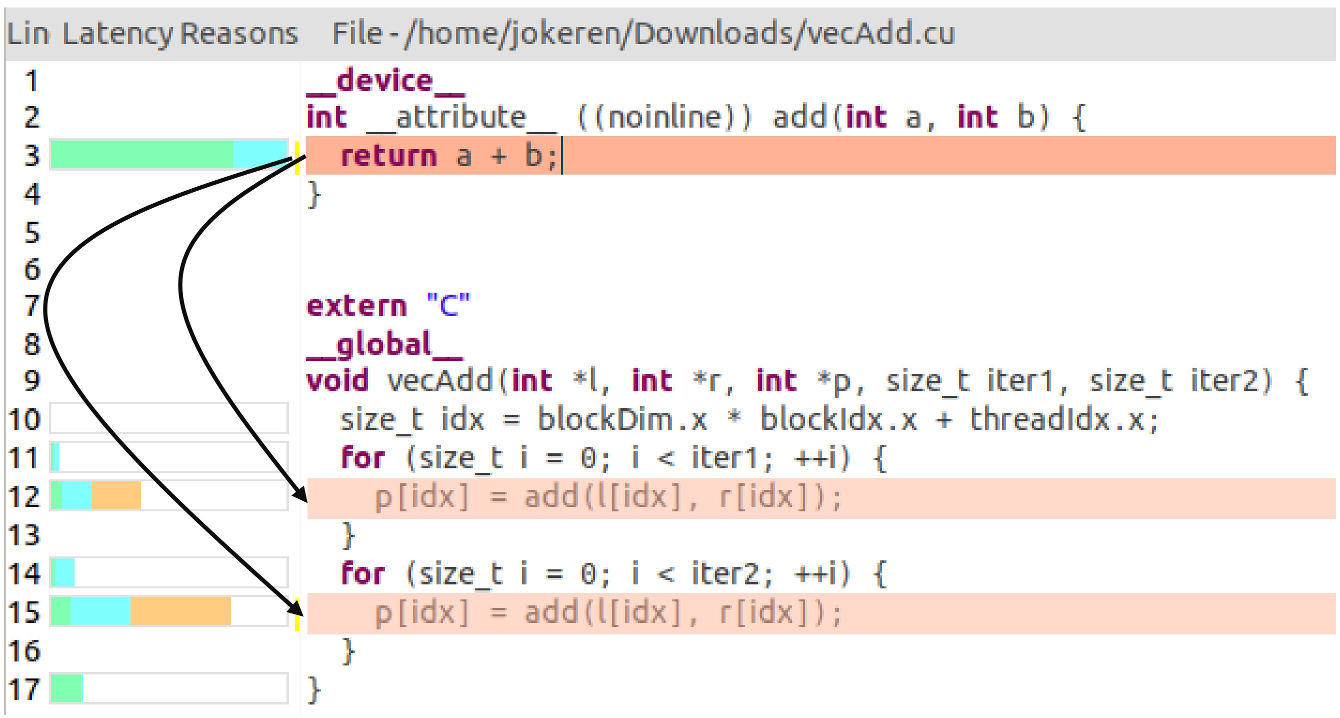
Start from a simple application

Two OpenMP threads launch *vecAdd* kernels concurrently

```
#omp parallel num_threads(2)
  cuLaunchKernel(vecAdd, ...)

int __noinline__ add(int a, int b) {
  return a + b;
}

void vecAdd(int *l, int *r, int *p, size_t iter1, size_t iter2) {
  size_t idx = blockDim.x * blockIdx.x + threadIdx.x;
  for (size_t i = 0; i < iter1; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
  for (size_t i = 0; i < iter2; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
}
```

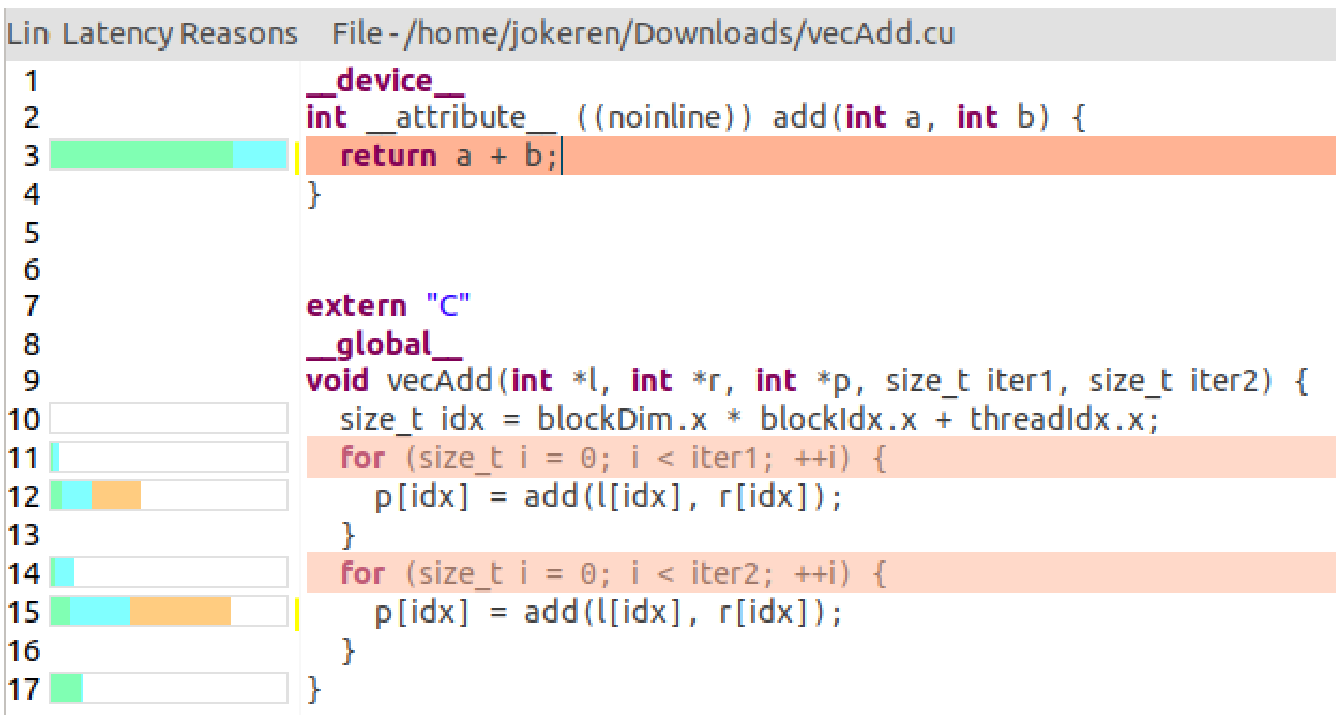
nvvp lacks of calling context

A tool should attribute latencies back to call sites at *line 12* and *line 15*

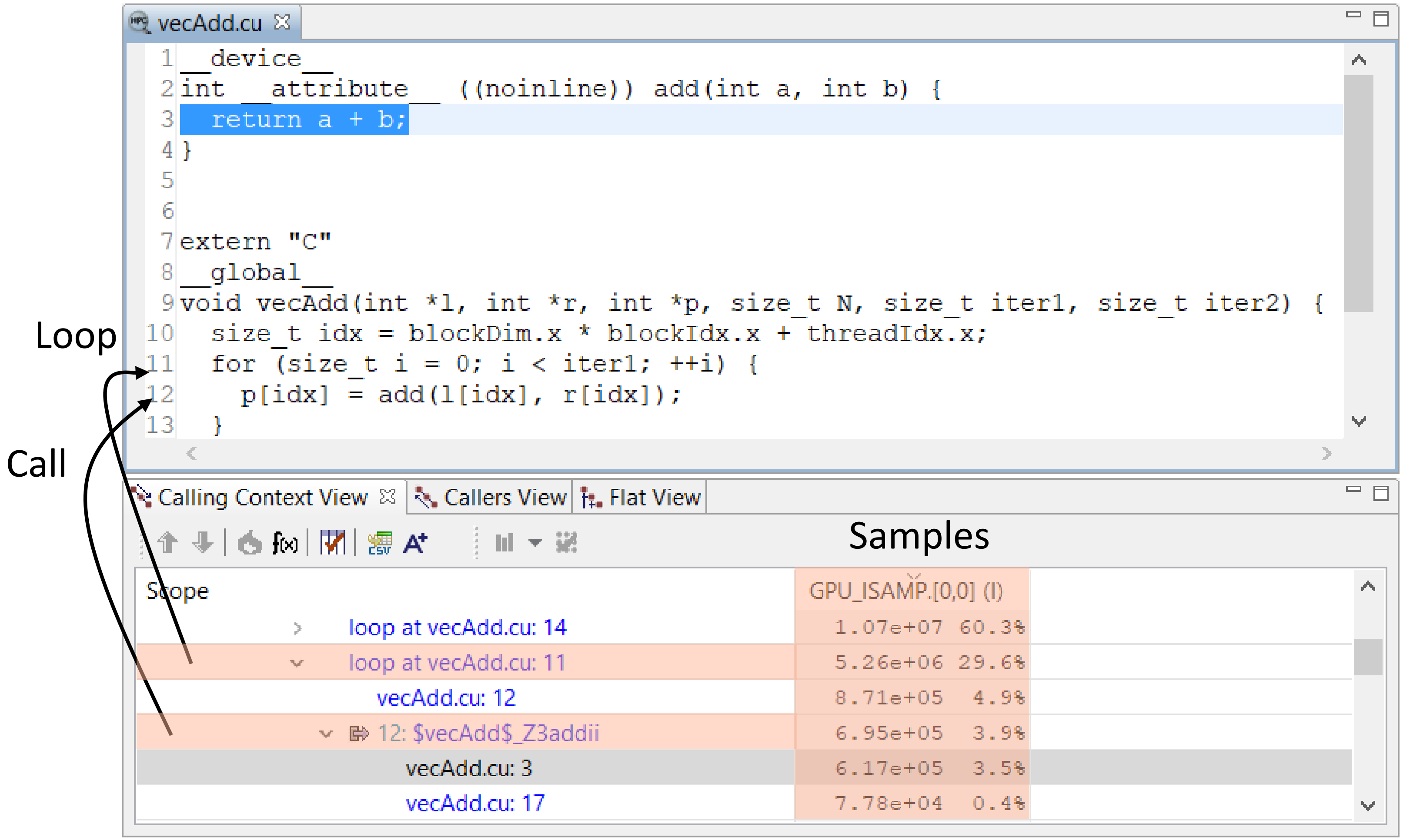


nvvp lacks of control flow analysis

A tool should attribute performance to loops



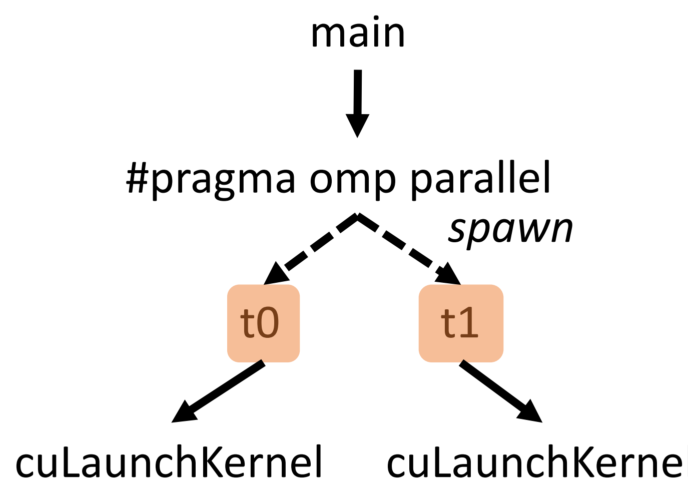
A complete profile view



Step 1: Build calling context tree on CPU

- Use HPCToolkit's CCT-tree

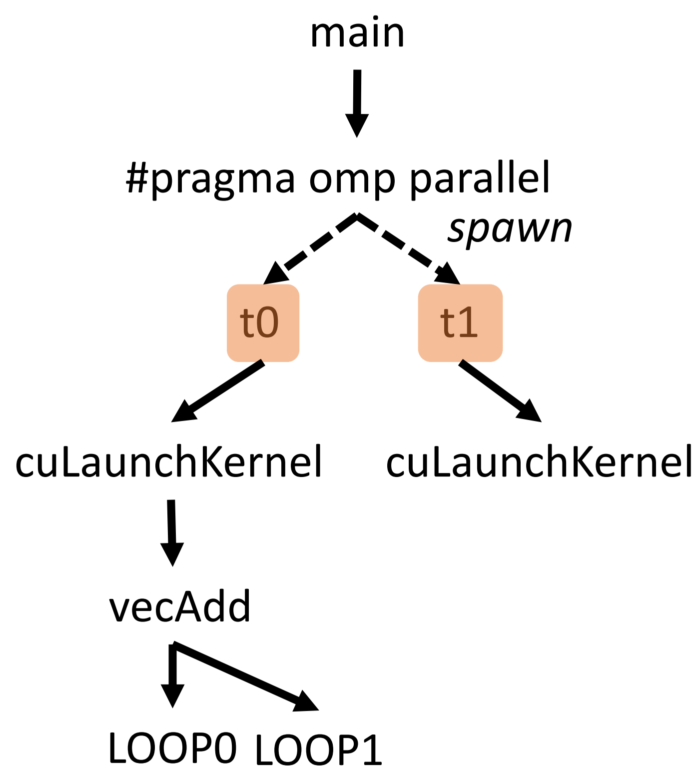
```
#omp parallel num_threads(2)
  cuLaunchKernel(vecAdd, ...)

int __noinline__ add(int a, int b) {
  return a + b;
}

void vecAdd(int *l, int *r, int *p, size_t
iter1, size_t iter2) {
  size_t idx = blockDim.x * blockIdx.x +
threadIdx.x;
  for (size_t i = 0; i < iter1; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
  for (size_t i = 0; i < iter2; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
}
```



Step 2: Apply static control flow analysis

- Identify loops

```
#omp parallel num_threads(2)
  cuLaunchKernel(vecAdd, ...)

int __noinline__ add(int a, int b) {
  return a + b;
}

void vecAdd(int *l, int *r, int *p, size_t iter1, size_t iter2) {
  size_t idx = blockDim.x * blockIdx.x + threadIdx.x;
  for (size_t i = 0; i < iter1; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
  for (size_t i = 0; i < iter2; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
}
```



Step 3: Collect GPU samples

- Two categories of threads
  - Worker threads
    - Launch kernels, move and allocate data, synchronize GPU calls
  - CUPTI thread
    - Collect GPU samples
- Interaction
  - **Notification:** A worker thread T creates a notification record when it launches a kernel and tags the kernel with a correlation ID C, notifying the CUPTI thread that C belongs to T
  - **Sample attribution**: The CUPTI thread collects samples associated with C and communicates sample attribution records back to thread T

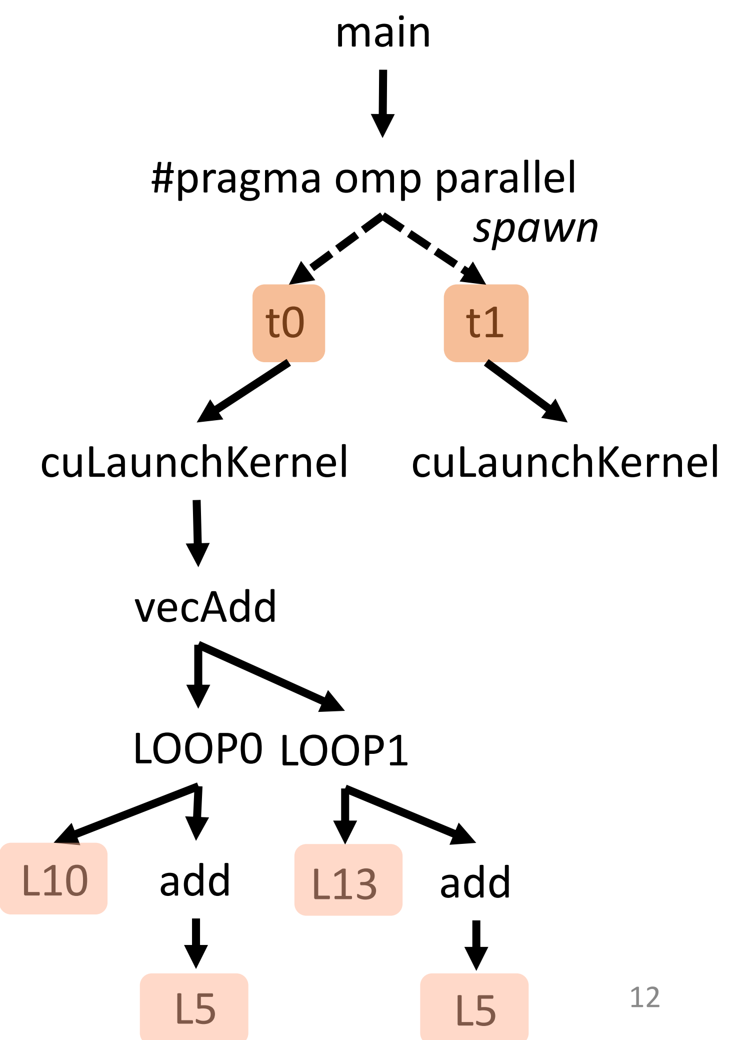
Sample attribution as an example

- The CUPTI thread adds samples to sample attribution queues using a ***push*** (*CAS*) operation. Each worker thread ***steals*** (*XCHG*) the head of its sample queue with NULL to steal all its records
- **Wait-free progress** is guaranteed because a CUPTI thread's CAS fails at most once when tries to add samples
- **Memory reclamation** occurs when a worker thread's samples have been attributed to its calling context tree. The worker puts records into a free queue which can be swapped by the CUPTI thread

Step 4: Attribute GPU samples

- Attribute samples to function calls

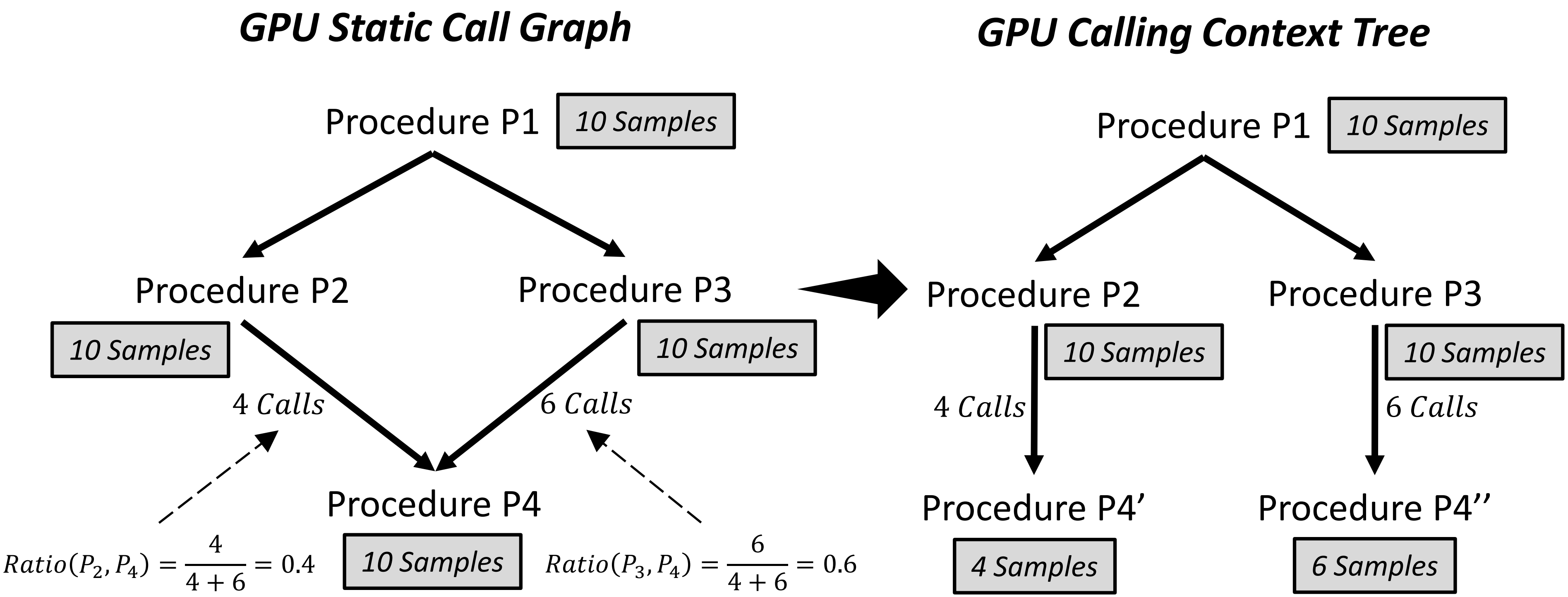
```
#omp parallel num_threads(2)
  cuLaunchKernel(vecAdd, ...)

int __noinline__ add(int a, int b) {
  return a + b;
}

void vecAdd(int *l, int *r, int *p, size_t iter1, size_t iter2) {
  size_t idx = blockDim.x * blockIdx.x + threadIdx.x;
  for (size_t i = 0; i < iter1; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
  for (size_t i = 0; i < iter2; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
}
```



Approximate a calling context tree

- Problem
  - High cost to unwind call stacks on GPU
- Solution
  - Construct a call graph by parsing call instructions and linking corresponding procedures
  - Create "supernode" for recursive procedures
  - Split the call graph into a calling context tree
  - Apportion samples of procedures that have multiple call sites

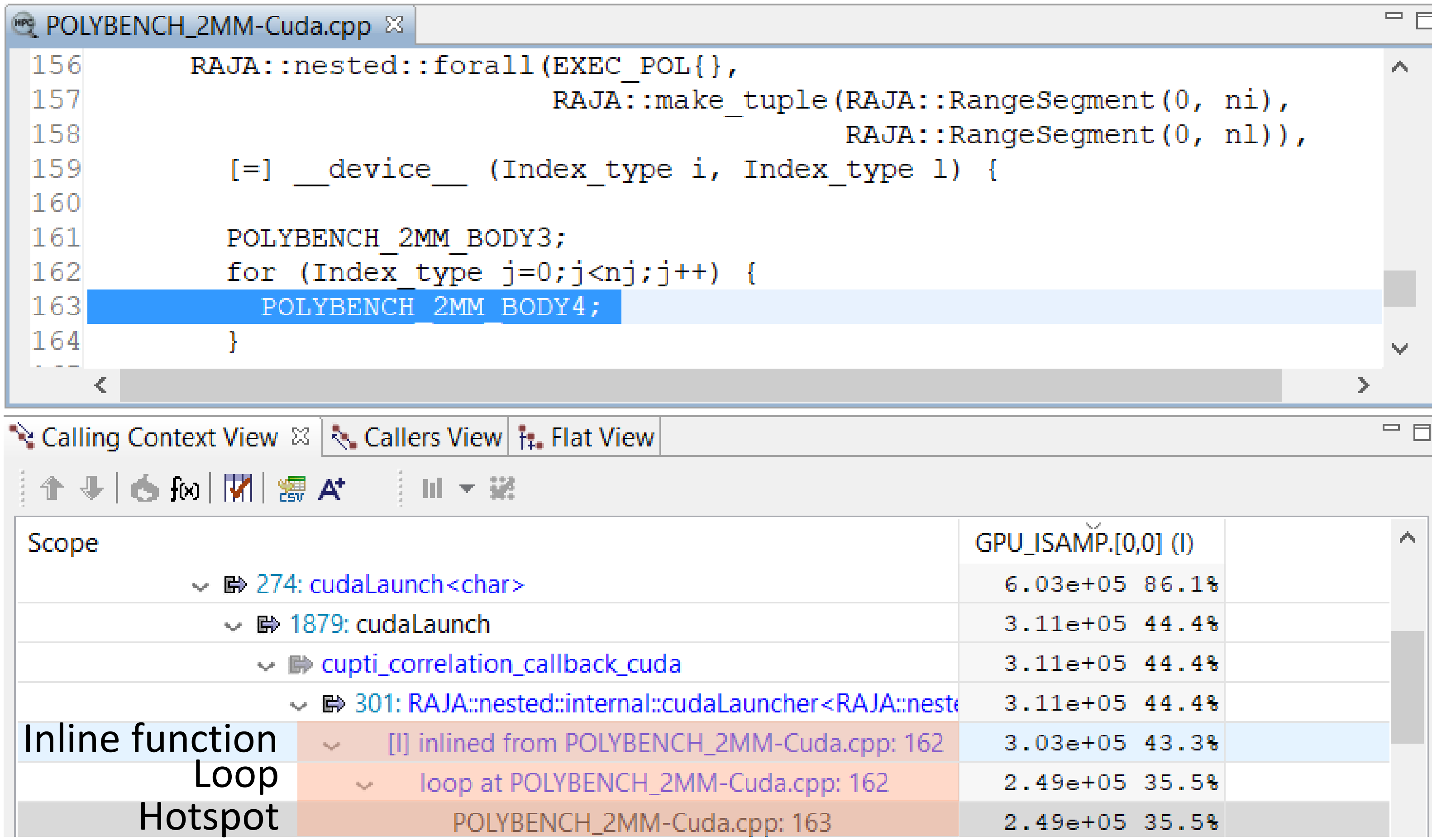
Apportion samples of a procedure based on its call sites



RAJA

- Template-based programming model based on C++
- Loop template can map a C++ lambda function for an iteration onto GPUs using CUDA
- RAJA performance suite
  - Explores performance of 30 loop-based computational kernels
  - https://github.com/LLNL/RAJAPerf

Profile rajaperf



Status and ongoing work

- We extended HPCToolkit to build a complete profile view for analyzing the runtime characteristics of GPU-accelerated applications
- Work in progress
  - Collect all the performance information, including kernel performance, data movement, compute utilization, and PC sampling information in a single phase
  - Study MPI-based GPU-accelerated applications

lulesh.cc

```
2719
2720            vhalf = Real_t(1.) / (Real_t(1.) + compHalfStep) ;
2721
2722            if ( delvc > Real_t(0.) ) {
2723               q_new /* = qq_old[i] = ql_old[i] */ = Real_t(0.) ;
2724            } else {
2725               ssc = ( pbvc * e_new + vhalf * vhalf * bvc * pHalfStep ) / rho0 ;
2726
2727               if ( ssc <= Real_t(.1111111e-36) ) {
2728                  ssc = Real_t(.3333333e-18) ;
2729               } else {
2730                  ssc = SQRT(ssc) ;
2731               }
2732
2733               q_new = (ssc*ql_old + qq_old) ;
2734            }
2735
2736            e_new = e_new + Real_t(0.5) * delvc
2737                  * (Real_t(3.0)*(p_old     + q_old)
2738                     Real_t(4.0)*(pHalfStep + q_new)) ;
```

Top-down view | Bottom-up view | Flat view

Scope	CPUTIME (usec):Sum (I)	GPU_ISAMP:Sum (I)
main	2.65e+07 99.8%	4.57e+06 100 %
3225: LagrangeLeapFrog(Domain&)	2.56e+07 96.3%	4.57e+06 100 %
3056: LagrangeElements(Domain&, int)	8.41e+06 31.7%	2.30e+06 50.2%
2864: ApplyMaterialPropertiesForElems(Domain&, double*)	1.73e+06 6.5%	1.13e+06 24.7%
2846: EvalEOSForElems(Domain&, double*)	1.73e+06 6.5%	1.13e+06 24.7%
2626: __omp_offloading_35_d6ae3ae__ZL15EvalEOSForElemsR6DomainPd_l2626		1.11e+06 24.3%
2627: __omp_offloading_35_d6ae3ae__ZL15EvalEOSForElemsR6DomainPd_l2626_impl__debug__		1.09e+06 23.9%
[I] inlined from lulesh.cc: 2626		5.36e+05 11.7%
2626: __omp_kernel_initialization_$_36		5.29e+05 11.6%
loop at lulesh.cc: 0		5.21e+05 11.4%
2628: _$_omp_outlined_$__debug__$_29		3.09e+05 6.8%
lulesh.cc: 2725		1.97e+04 0.4%
lulesh.cc: 2803		1.97e+04 0.4%
lulesh.cc: 2767		1.95e+04 0.4%
lulesh.cc: 2720		9.92e+03 0.2%
lulesh.cc: 2688		9.89e+03 0.2%
lulesh.cc: 2686		9.86e+03 0.2%
lulesh.cc: 2834		6.94e+03 0.2%

CPU Calling Context

GPU Calling Context

GPU Hotspot

main.c | Operators.hpp | KernelBase.cpp

```
311 template <typename Ret, typename Arg1 = Ret, typename Arg2 = Arg1>
312 struct plus : public detail::binary_function<Arg1, Arg2, Ret>,
313               detail::associative_tag {
314   RAJA_HOST_DEVICE constexpr Ret operator()(const Arg1& lhs,
315                                             const Arg2& rhs) const
316   {
317     return Ret{lhs} + rhs;
318   }
319   RAJA_HOST_DEVICE static constexpr Ret identity() { return Ret{0}; }
320 };
321
322 template <typename Ret, typename Arg1 = Ret, typename Arg2 = Arg1>
323 struct minus : public detail::binary_function<Arg1, Arg2, Ret> {
324   RAJA_HOST_DEVICE constexpr Ret operator()(const Arg1& lhs,
325                                             const Arg2& rhs) const
326   {
327     return Ret{lhs} - rhs;
328   }
329 };
```

Top-down view | Bottom-up view | Flat view

Scope	GPU_ISAMP.[0,0] (I)	CPUTIME (usec).[0,0...
516: main	6.57e+06 100 %	2.80e+07 100.0
34: rajaperf::Executor::runSuite()	6.57e+06 100 %	2.80e+07 100.0
390: rajaperf::KernelBase::execute(rajaperf::VariantID)	6.40e+06 97.4%	7.94e+06 28.4%
72: rajaperf::stream::DOT::runKernel(rajaperf::VariantID)	6.40e+06 97.4%	7.90e+06 28.3%
167: void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<long, long>, __nv_dl_wrapper_t<__nv_dl_tag<void (rajaperf::stream::D	5.61e+06 85.4%	4.25e+06 15.2%
190: cudaLaunchKernel<char>	5.61e+06 85.4%	4.25e+06 15.2%
195: cudaLaunchKernel	5.61e+06 85.4%	4.25e+06 15.2%
RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long, long, long*>, rajaperf::stream::DOT::runC	5.61e+06 85.4%	
151: RAJA::internal::Privatizer<rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}>::~Privatizer	4.78e+06 72.7%	
54: rajaperf::stream::DOT::runCudaVariant	4.69e+06 71.3%	
129: RAJA::ReduceSum<RAJA::policy::cuda::cuda_reduce<256ul, false, false>, double>::~ReduceSum	4.61e+06 70.1%	
190: RAJA::cuda::Reduce<false, RAJA::reduce::sum<double>, double, false>::~Reduce	4.53e+06 68.9%	
848: RAJA::cuda::Reduce<false, RAJA::reduce::sum<double>, double, false>::~Reduce	4.45e+06 67.7%	
843: RAJA::cuda::Reduce_Data<false, RAJA::reduce::sum<double>, double>::grid_reduce	4.32e+06 65.8%	
[I] inlined from reduce.hpp: 203	3.19e+06 48.5%	
loop at reduce.hpp: 203	1.18e+06 17.9%	
loop at reduce.hpp: 203	6.40e+05 9.7%	
72: RAJA::operators::plus<double, double, double>::operator	1.32e+05 2.0%	
Operators.hpp: 317	7.06e+04 1.1%	
Operators.hpp: 314	5.67e+04 0.9%	

Template GPU Procedures above Actual Kernel Code