# Profiling and Debugging GPU-accelerated AI Applications

Keren Zhou
George Mason University
kzhou6@gmu.edu

# AI Applications

IBM Watson



Midjourney

# AI System Software Stack

XI

Meet Claude

Bard

PyTorch

TensorFlow

deepspeed

AMD

intel

NVIDIA

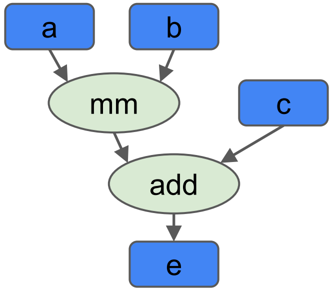
# AI Code Transformation Workflow

```
a = torch.randn(64, 32)
b = torch.randn(32, 64)
c = torch.randn(64, 64)
d = torch.mm(a, b)
e = c + d
```

| Model | Graph | Kernel | Device |
|---|---|---|---|
| • PyTorch<br>• TensorFlow<br>• JAX | • XLA/HLO<br>• TVM/Relay<br>• PyTorch/fx | • CUDA<br>• HIP<br>• OpenCL | • GPU<br>• CPU<br>• FPGA |

# AI Code Transformation Workflow

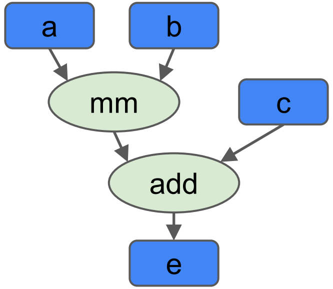
```
a = torch.randn(64, 32)
b = torch.randn(32, 64)
c = torch.randn(64, 64)
d = torch.mm(a, b)
e = c + d
```




| Model | Graph | Kernel | Device |
|---|---|---|---|
| • PyTorch<br>• TensorFlow<br>• JAX | • XLA/HLO<br>• TVM/Relay<br>• TorchDynamo | • CUDA<br>• HIP<br>• OpenCL | • GPU<br>• CPU<br>• FPGA |

# AI Code Transformation Workflow

```
a = torch.randn(64, 32)
b = torch.randn(32, 64)
c = torch.randn(64, 64)
d = torch.mm(a, b)
e = c + d
```

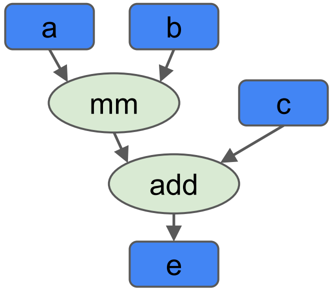



```
__global__
void mm(float *a, float *b,
float *c) {
    float *a_tile;
    float *b_tile;
    …
}
```

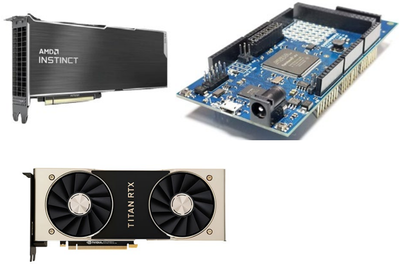
| Model | Graph | Kernel | Device |
| --- | --- | --- | --- |
| • PyTorch | • XLA/HLO | • CUDA | • GPU |
| • TensorFlow | • TVM/Relay | • HIP | • CPU |
| • JAX | • TorchDynamo | • OpenCL | • FPGA |

# AI Code Transformation Workflow

```python
a = torch.randn(64, 32)
b = torch.randn(32, 64)
c = torch.randn(64, 64)
d = torch.mm(a, b)
e = c + d
```



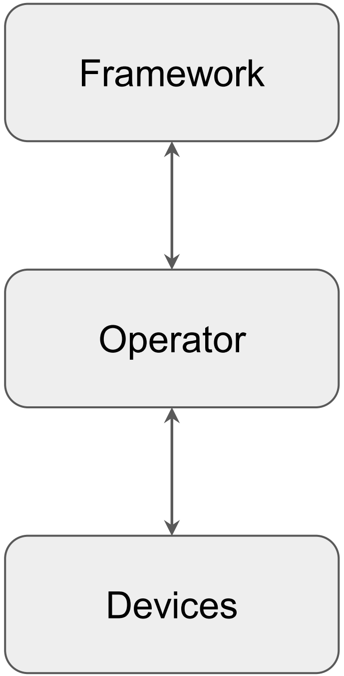

```c
__global__
void mm(float *a, float *b,
float *c) {
    float *a_tile;
    float *b_tile;
    …
}
```




| Model | Graph | Kernel | Device |
|---|---|---|---|
| • PyTorch<br>• TensorFlow<br>• JAX | • XLA/HLO<br>• TVM/Relay<br>• TorchDynamo | • CUDA<br>• HIP<br>• OpenCL | • GPU<br>• CPU<br>• FPGA |

# Understanding Hidden Issues is Difficult

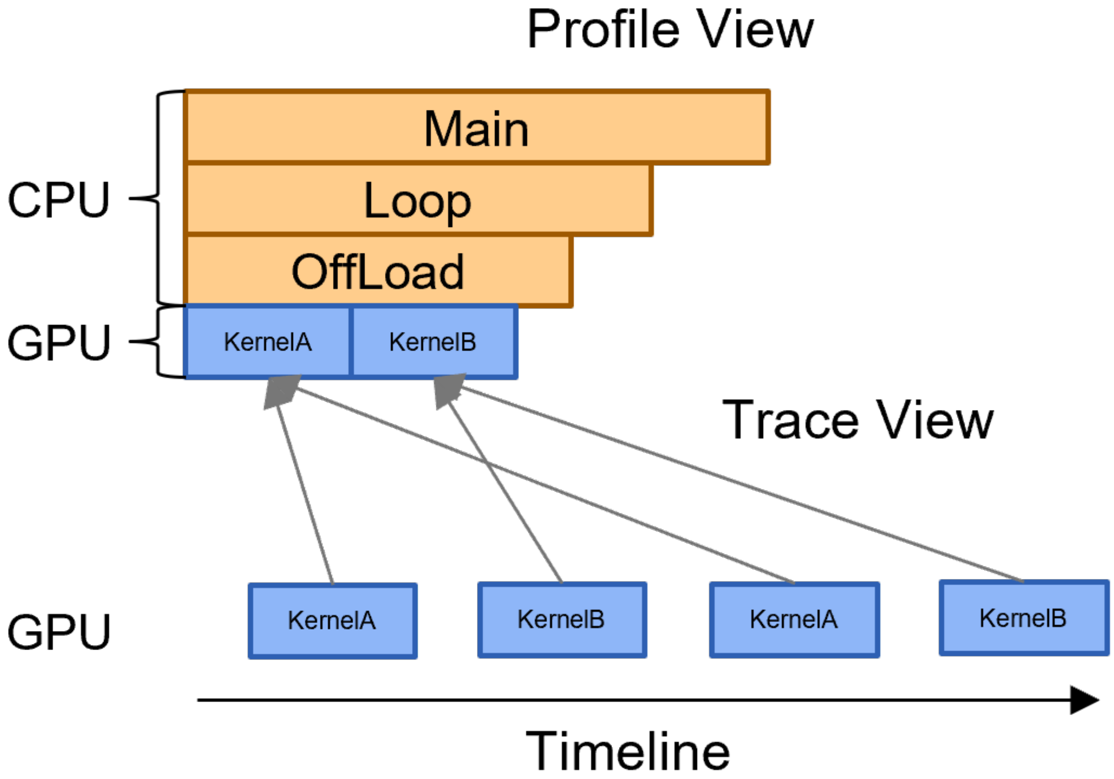
- Cross stack performance issues
    - Framework is not able to schedule/fuse operators
- CPU-GPU interaction
    - GPUs wait for CPUs or communication
- Compiler fail to generate optimal code
    - Deep learning compilers are not perfect



# Profiling Tools

- Linux Perf/gprof
  - CPU cycles, cache misses, and other hardware metrics
- HPCToolkit
  - CPU and GPU profiling with static binary analysis
- Nsight Systems/Intel VTune/AMD RocTracer
  - Profiling GPU events

# Profile and Trace Views



# Debugging Tools

- GDB/LLDB/PDB
  - Supports step-through execution, breakpoints, and watchpoints
- Valgrind
  - Memory leak detection
- CUDA-GDB
  - GPU instruction and memory inspection

# Outline

- Introduction
- DeepContext
- Triton
  - Profiler
  - Interpreter
  - Visualizer
- Ongoing Work

# DeepContext

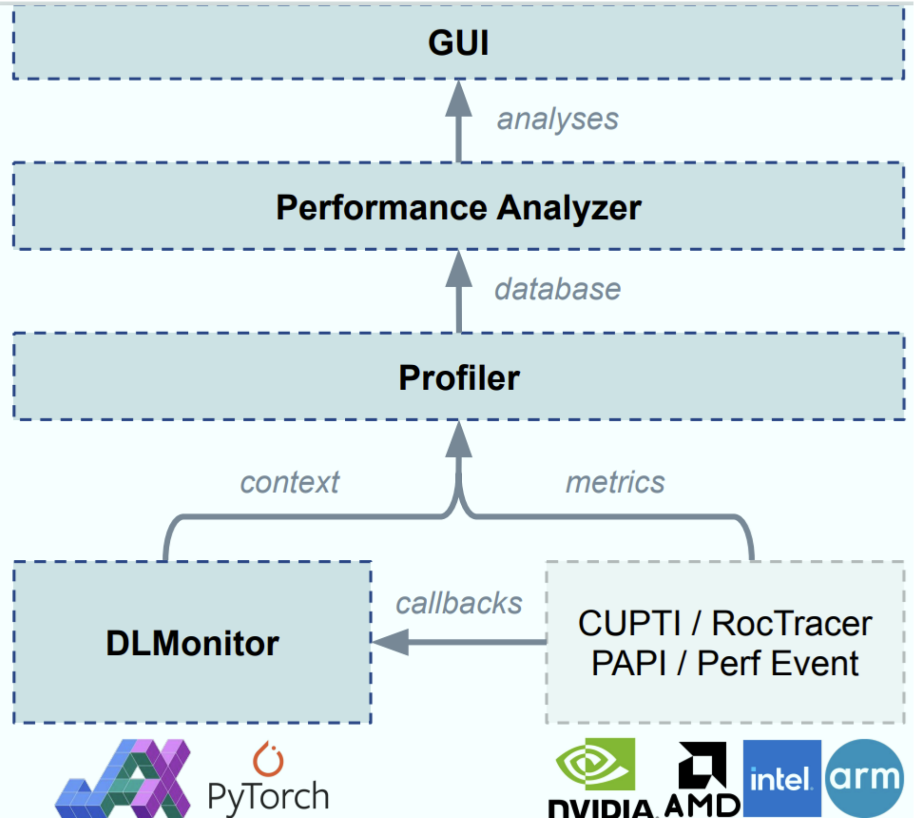
# A Cross-Platform and Cross-Framework Profiler

- GPU vendor-provided tools cannot be applied across platforms
  - Nsight Systems
  - RocTracer
- Framework native tools cannot be applied across frameworks
  - PyTorch profiler
  - JAX profiler

# A Context-Aware Profiler

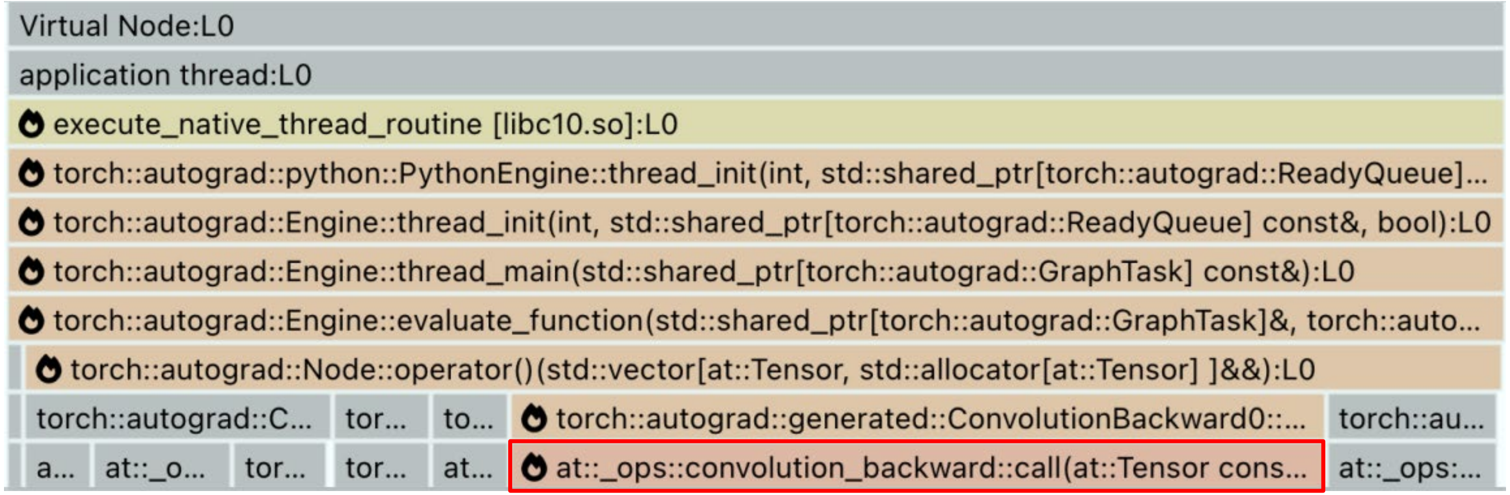
- DeepContext obtains contexts from multiple-sources and concatenates them together to support informed decisions
    - Python
    - Framework
    - C++/C
    - GPU API
    - GPU device

# Implementation



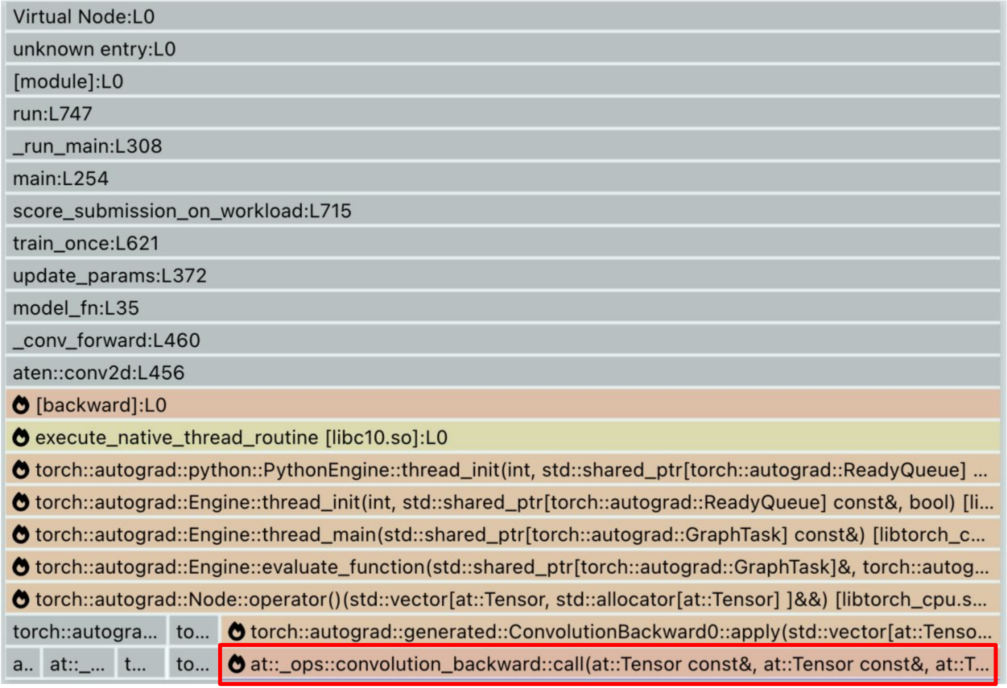
# Case Study

- PyTorch without context correlation



# Case Study

- PyTorch with context correlation

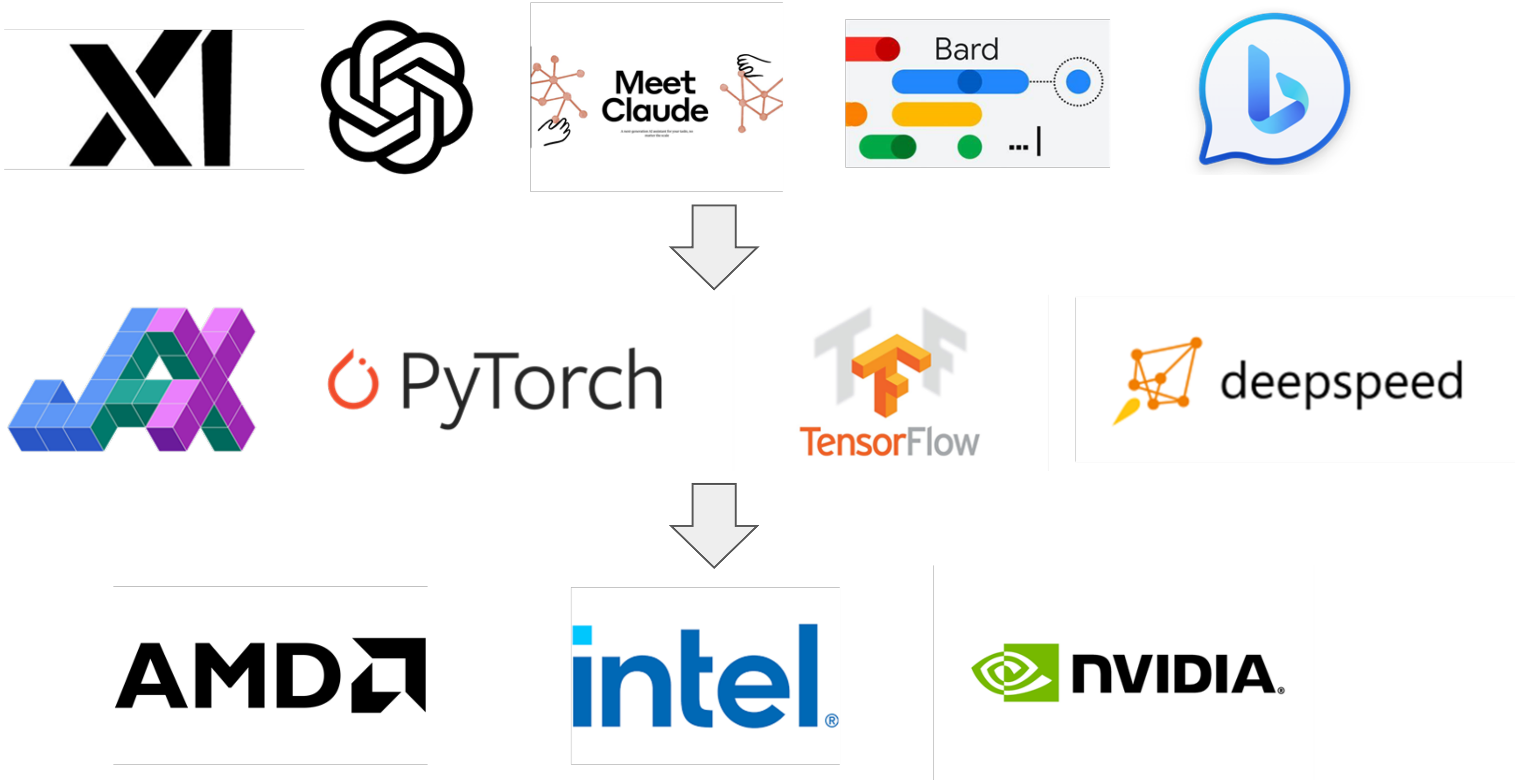


# Triton

# Triton

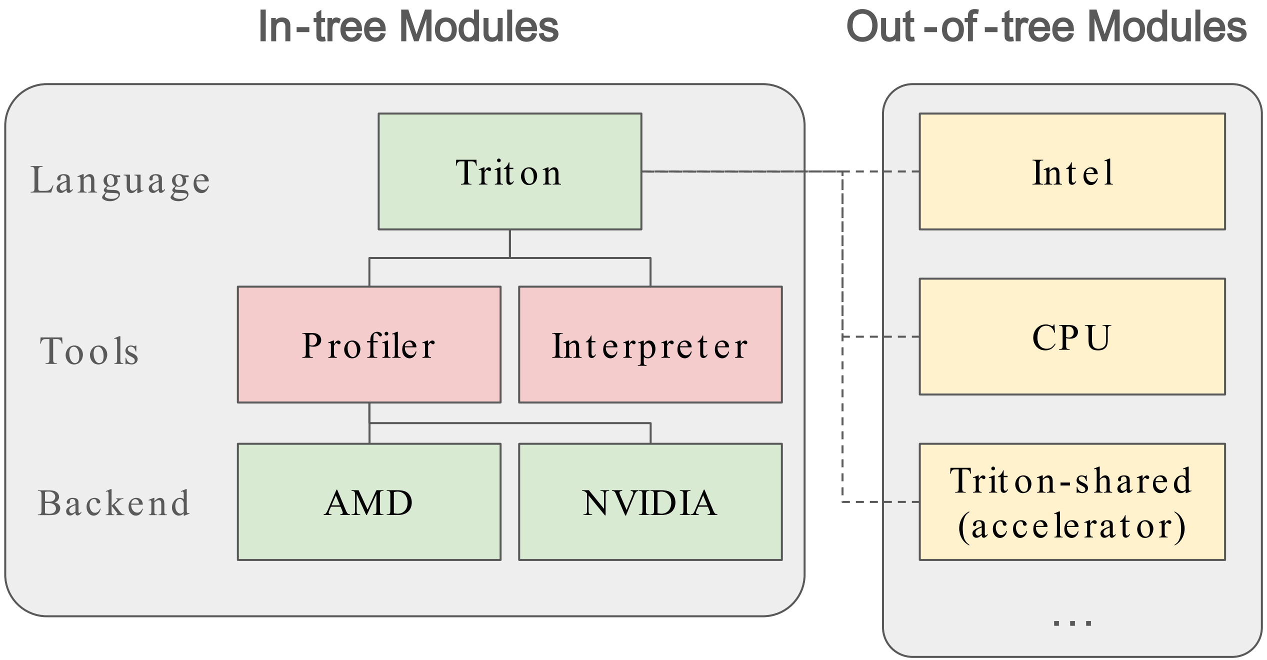
- A Python-like language
- A JIT compiler
- A PyTorch backend
- A set of MLIR dialects
- An organization
- A community

# Why Triton?


Meet Claude
Bard
PyTorch
TensorFlow
deepspeed
AMD
intel
NVIDIA

# Why Triton?

# Triton Modules



# Triton Language

- Python-like language designed for high flexibility and performance in deep learning applications
  - Support tensor interface similar to PyTorch
  - Uses Python-like syntax
- Compared to CUDA/ROCm, Triton simplifies GPU programming
  - Only requiring knowledge that a kernel is divided into multiple blocks (Triton programs)
  - Most underlying details are handled by the compiler

# A Simple Triton Program

z: dim0 x dim1 = x: dim0 x dim1 + y: dim0 x dim1

vecAdd

Kernel decorator

Programming model

Creation ops

Memory ops

```
@triton.jit
def add_kernel(x_ptr, y_ptr, z_ptr, dim0, dim1,
               BLOCK_DIM0: tl.constexpr, BLOCK_DIM1: tl.constexpr):
  pid_x = tl.program_id(axis=0)
  pid_y = tl.program_id(axis=1)
  block_start = pid_x * BLOCK_DIM0 * dim1 + pid_y * BLOCK_DIM1
  offsets_dim0 = tl.arange(0, BLOCK_DIM0)[:,None]
  offsets_dim1 = tl.arange(0, BLOCK_DIM1)[None, :]
  offsets = block_start + offsets_dim0 * dim1 + offsets_dim1
  masks = (offsets_dim0 < dim0) & (offsets_dim1 < dim1)
  x = tl.load(x_ptr + offsets, mask=masks)
  y = tl.load(y_ptr + offsets, mask=masks)
  output = x + y
  tl.store(z_ptr + offsets, output, mask=masks)
```

# Triton Profiler

# Proton (A Profiler for Triton)

- Provide a quick, intuitive, and simple way to check kernel performance
  - Open source
  - Multiple vendor GPUs
  - Flexible metrics collection
    - Hardware metrics
    - Software metrics
  - Call path profiling

# Call Path Profiling

```
55.193 ROOT
├─ 31.212 /home/kzhou6/gh200/triton/third_party/proton/tutorials/dynamic_net.py:<module>@98
|  └─ 31.212 /home/kzhou6/gh200/triton/python/triton/profiler/profile.py:wrapper@151
|     ├─ 0.002 /home/kzhou6/gh200/triton/third_party/proton/tutorials/dynamic_net.py:run@51
|     |  └─ 0.002 _ZN50_GLOBAL__N__c922cf59_17_RangeFactories_cu_38772b0829elementwise_kernel_with_indexIi
_clEvENKUlvE0_clEvEUllE_EEvT_T0_PN15function_traitsISD_E11result_typeE
|     ├─ 0.003 /home/kzhou6/gh200/triton/third_party/proton/tutorials/dynamic_net.py:run@52
|     |  └─ 0.003 _ZN2at6native29vectorized_elementwise_kernelILi4EZZZNS0_15sin_kernel_cudaERNS_18TensorIt
_T1_
|     ├─ 19.610 /home/kzhou6/gh200/triton/third_party/proton/tutorials/dynamic_net.py:run@66
|     |  └─ 19.610 /home/kzhou6/gh200/pytorch/torch/nn/modules/module.py:_wrapped_call_impl@1532
|     |     └─ 19.610 /home/kzhou6/gh200/pytorch/torch/nn/modules/module.py:_call_impl@1541
|     |        ├─ 13.931 /home/kzhou6/gh200/triton/third_party/proton/tutorials/dynamic_net.py:forward@36
|     |        |  ├─ 2.939 /home/kzhou6/gh200/pytorch/torch/_tensor.py:wrapped@40
|     |        |  |  ├─ 1.460 _ZN2at6native29vectorized_elementwise_kernelILi4EZNS0_53_GLOBAL__N__2ced54f0
18TensorIteratorBaseET0_EUlfE0_NS_6detail5ArrayIPcLi2EEEEEviS6_T1_
|     |        |  |  └─ 1.479 _ZN2at6native29vectorized_elementwise_kernelILi4EZNS0_53_GLOBAL__N__2ced54f0
18TensorIteratorBaseET0_EUlfE_NS_6detail5ArrayIPcLi2EEEEEviS6_T1_
|     |        |  ├─ 6.022 _ZN2at6native18elementwise_kernelILi128ELi2EZNS0_22gpu_kernel_impl_nocastINS0_1
eratorBaseERKT_EUliE_EEviT1_
|     |        |  ├─ 2.025 _ZN2at6native18elementwise_kernelILi128ELi2EZNS0_22gpu_kernel_impl_nocastINS0_1
|     |        |  └─ 2.945 _ZN2at6native29vectorized_elementwise_kernelILi4ENS0_15CUDAFunctor_addIfEENS_6d
```

Python Context

```
54.763 ROOT
├─ 25.004 backward
|  ├─ 14.366 _ZN2at6native13reduce_kernelILi512ELi1ENS0_
|  ├─ 2.007 _ZN2at6native18elementwise_kernelILi128ELi2E
vEUlfffE_EEvRNS_18TensorIteratorBaseERKT_EUliE_EEviT1_
|  ├─ 2.461 _ZN2at6native29vectorized_elementwise_kernel
|  ├─ 5.725 _ZN2at6native29vectorized_elementwise_kernel
|  └─ 0.446 _ZN2at6native29vectorized_elementwise_kernel
├─ 19.399 forward
|  ├─ 7.961 _ZN2at6native18elementwise_kernelILi128ELi2E
EUliE_EEviT1_
|  ├─ 2.018 _ZN2at6native18elementwise_kernelILi128ELi2E
|  ├─ 4.415 _ZN2at6native29vectorized_elementwise_kernel
|  ├─ 1.455 _ZN2at6native29vectorized_elementwise_kernel
seET0_EUlfE0_NS_6detail5ArrayIPcLi2EEEEEviS6_T1_
|  ├─ 2.073 _ZN2at6native29vectorized_elementwise_kernel
seET0_EUlfE2_NS_6detail5ArrayIPcLi2EEEEEviS6_T1_
|  └─ 1.477 _ZN2at6native29vectorized_elementwise_kernel
seET0_EUlfE_NS_6detail5ArrayIPcLi2EEEEEviS6_T1_
├─ 0.004 init
|  ├─ 0.003 _ZN2at6native29vectorized_elementwise_kernel
|  └─ 0.001 _ZN50_GLOBAL__N__c922cf59_17_RangeFactories_
NKUlvE0_clEvEUllE_EEvT_T0_PN15function_traitsISD_E11resu
├─ 4.412 loss
|  ├─ 2.949 _ZN2at6native13reduce_kernelILi512ELi1ENS0_8
|  └─ 1.462 _ZN2at6native29vectorized_elementwise_kernel
```

Shadow Context

# Proton *vs* Nsight Systems *vs* Nsight Compute

| Tool | Nsys | NCU | Proton |
|---|---|---|---|
| Overhead | Up to 3x | Up to 1000x | Up to 1.5x |
| Profile size | Large | Large | Tiny (<1MB) |
| Profiling targets | NVIDIA GPUs, CPUs | NVIDIA GPUs | NVIDIA and AMD GPUs |
| Granularity | Kernels | Kernels and instructions | Kernels and instructions |
| Metrics | GPU time<br>GPU utilization<br>CPU samples | A complete set of metrics from hardware counters | GPU time<br>GPU instruction samples<br>User-defined metrics |
| Triton hooks | N/A | N/A | Support |

# User Interface

- Lightweight source code instrumentation
  - Profile start/stop/finalize
  - Scopes
  - Hooks
- Command line
  - python    - m proton main.py
  - proton main.py

# Start/Stop/Finalize Profiling

- Profile only interesting regions
    - proton.start(profile_name: str)                    - > session_id: int
    - proton.finalize()
- Skip some regions, but accumulate to the same profile
    - session_id = proton.start(...)
    - proton.deactive(session_id)
    - … # region skipped
    - proton.activate(session_id)

# Scopes

- A user-defined region with semantic information
    - Initialization
    - Forward
    - Backward
- `with proton.scope(name)`

# Metrics

- Hardware metrics
  - Come from profiling substrates (e.g., CUPTI)
    - Kernel time
    - Instruction samples
- User-defined metrics
  - Come from users
    - Flops
    - Bytes
    - Tokens

# Triton Hook

- A way to compute and associate metrics with each Triton kernel launch
  - @triton.jit(launch_metadata=metadata_fn)
- metadata_fn is a callback function that
  - Takes three input arguments
    - Grid
    - Metadata
      - warps, stages, shared
    - Args
  - Returns a dictionary containing
    - Renamed kernel name
    - Other metric names and values

# Instruction Sampling

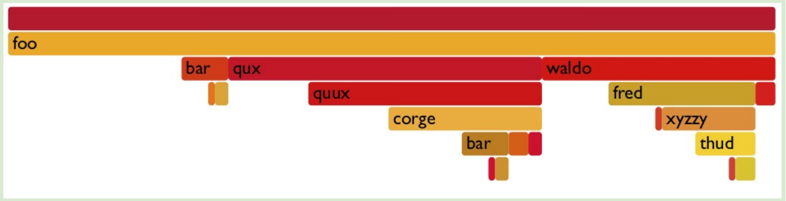
- For large functions, we need fine-grained insights about which lines/IRs/instructions are expensive
- Instruction sampling is an experimental feature we're developing to support this goal
  - It's called *pc sampling* using NVIDIA's terminology

# Instruction Sampling

- Sample an instruction on each active GPU SM every *N* cycles
- Each instruction is associated with a *stall* reason if available
  - Why the instruction was not issued
- “Low overhead” with regard to each kernel’s GPU time
- Available on NVIDIA, AMD and Intel GPUs

# Viewer

- proton - viewer a call path visualization tool
- Load json data into pandas
- Render it on terminal using hatchet
  - [LLNL-Hatchet](): A flexible package for performance data analysis
  - Hatchet can also convert the format into other formats such as flamegraph
- proton - viewer - h for more information



# Case Study: Matmul

- We use scopes to annotate
  - Matmul shapes: matmul_M_N_K
  - Autotuned configurations: <autotune>
- We use hooks to annotate
  - Grid dimensions
  - Number of warps
  - Number of stages

# Case Study: Matmul

```
├─ 1090.280 matmul_1024_1024_1024
│  └─ 981.306 triton
│     ├─ 90.695 <autotune>
│     │  ├─ 18.325 matmul_<grid:128x1x1>_<cluster:1x1x1>_<warps:4>_<shared:36864>_<stages:4>
│     │  ├─ 14.102 matmul_<grid:256x1x1>_<cluster:1x1x1>_<warps:4>_<shared:30720>_<stages:4>
│     │  ├─ 8.593 matmul_<grid:32x1x1>_<cluster:1x1x1>_<warps:8>_<shared:98304>_<stages:3>
│     │  ├─ 32.475 matmul_<grid:512x1x1>_<cluster:1x1x1>_<warps:2>_<shared:24576>_<stages:5>
│     │  ├─ 8.307 matmul_<grid:64x1x1>_<cluster:1x1x1>_<warps:4>_<shared:49152>_<stages:4>
│     │  └─ 8.893 matmul_<grid:64x1x1>_<cluster:1x1x1>_<warps:4>_<shared:61440>_<stages:4>
│     ├─ 882.505 _ZN2at6native29vectorized_elementwise_kernelILi4ENS0_11FillFunctorIiEENS_6de
│     └─ 8.106 matmul_<grid:64x1x1>_<cluster:1x1x1>_<warps:4>_<shared:49152>_<stages:4>
```

# Triton Interpreter

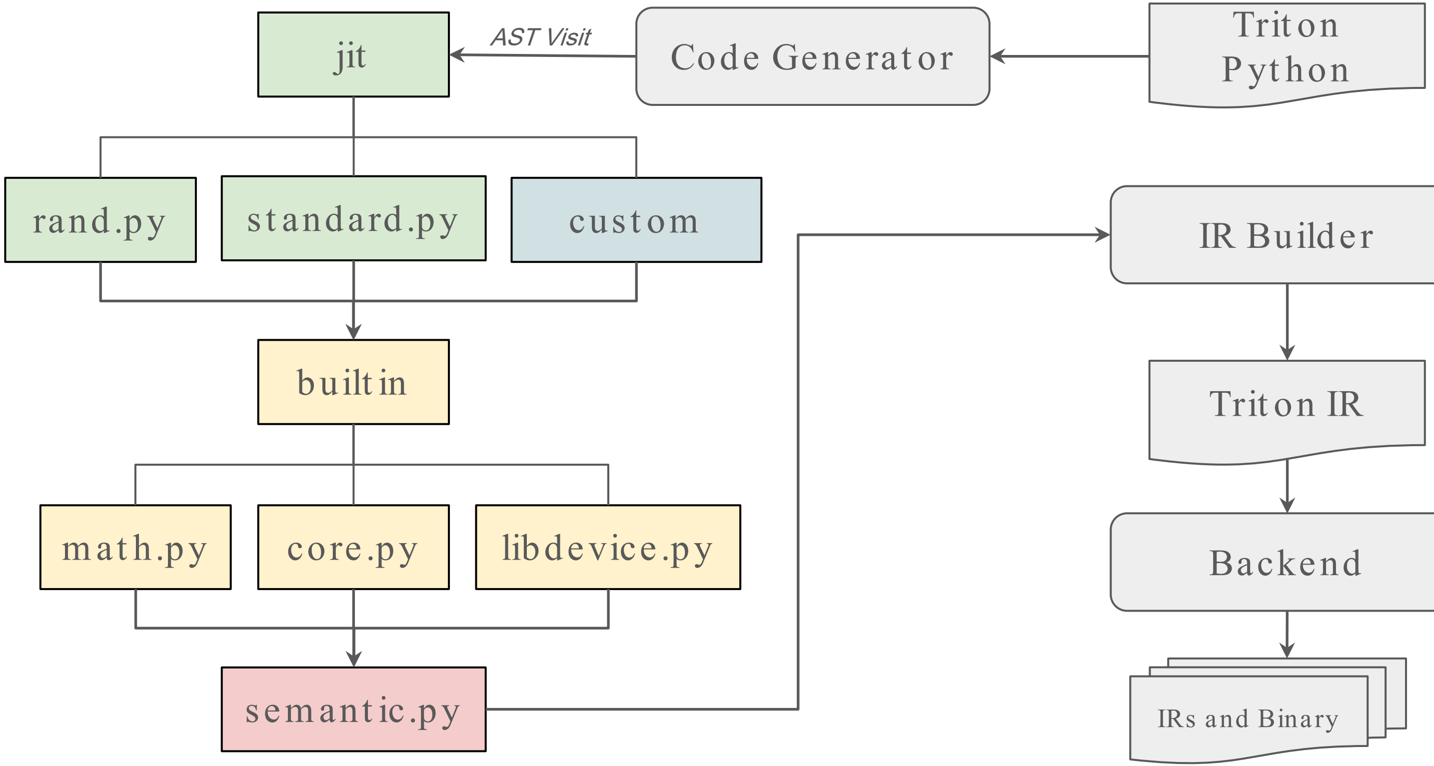
# Debugging Triton Programs is Still Not Easy

- Launch parallel programs instances
    - `kernel[(x, y, z,)](params…)`
- Calculate offsets
    - `tl.arange(0, N)[None, :] // H * stride`
- Access multi-dimensional tensors with masks and others
    - `tl.load(offsets, masks, others)`

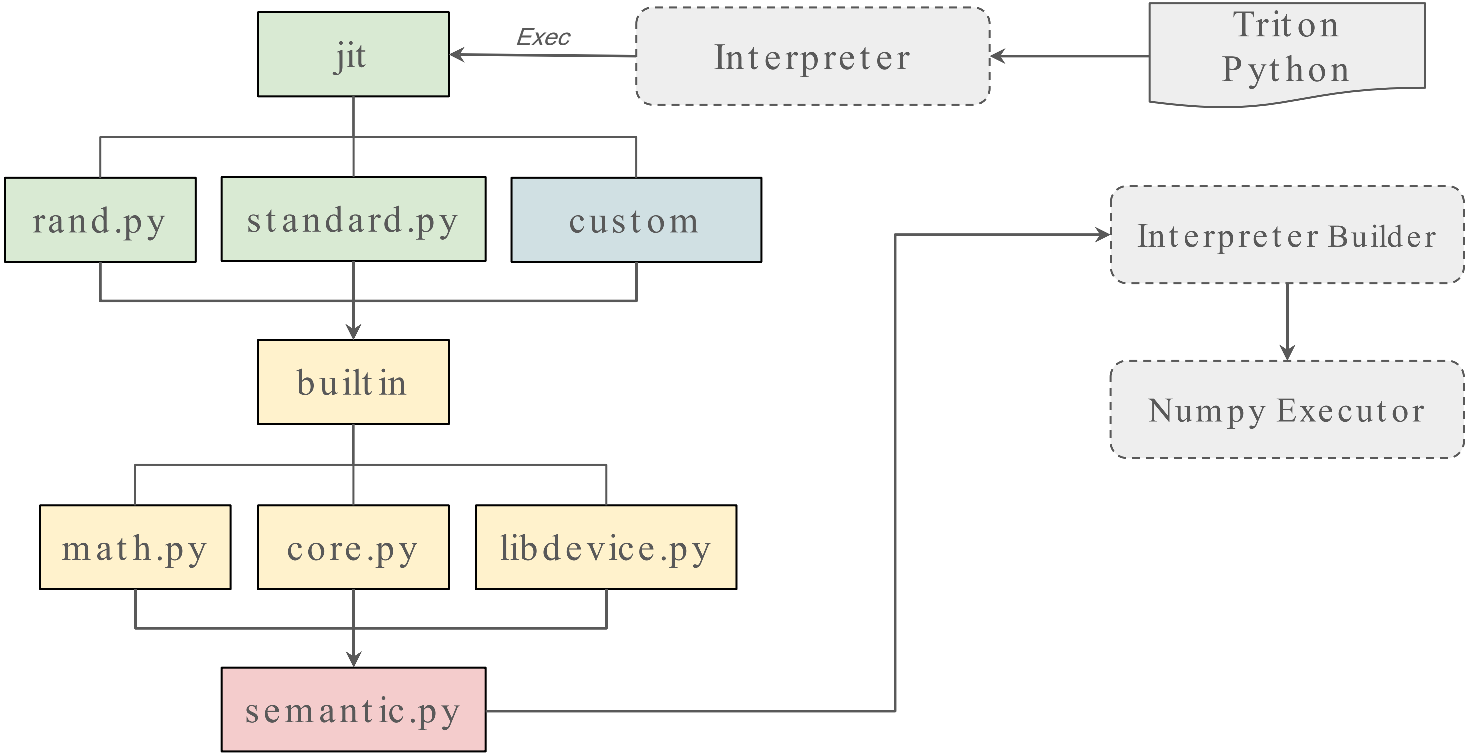
# Triton Interpreter

- A debugger that allows users to debug Triton programs as if they were debugging standard Python programs on the CPU
  - Attach pdb to step through each statement interactively
  - Print tensor values as multidimensional arrays for better visualization
  - Serialize the execution of multiple Triton programs for easier debugging

# Frontend without the Interpreter



# Frontend with the Interpreter



# Case Study: Vector Addition

vecAdd

```
@triton.jit
def add_kernel_device_print(x_ptr, y_ptr, z_ptr, dim0, dim1,
                            BLOCK_DIM0: tl.constexpr, BLOCK_DIM1: tl.constexpr):
  pid_x = tl.program_id(axis=0)
  pid_y = tl.program_id(axis=1)
  block_start = pid_x * BLOCK_DIM0 * dim1 + pid_y * BLOCK_DIM1
  offsets_dim0 = tl.arange(0, BLOCK_DIM0)[:,None]
  offsets_dim1 = tl.arange(0, BLOCK_DIM1)[None, :]
  offsets = block_start + offsets_dim0 * dim1 + offsets_dim1
  tl.device_print("offsets=", offsets)
  masks = (offsets_dim0 < dim0) & (offsets_dim1 < dim1)
  x = tl.load(x_ptr + offsets, mask=masks)
  y = tl.load(y_ptr + offsets, mask=masks)
  output = x + y
  tl.store(z_ptr + offsets, output, mask=masks)
```

# Case Study: Vector Addition Without Interpreter

```
pid (6, 5, 0) idx (10,  0) offsets=: 13648
pid (6, 5, 0) idx (10,  1) offsets=: 13649
pid (6, 5, 0) idx (10,  2) offsets=: 13650
pid (6, 5, 0) idx (10,  3) offsets=: 13651
pid (6, 5, 0) idx (10,  4) offsets=: 13652
pid (6, 5, 0) idx (10,  5) offsets=: 13653
pid (6, 5, 0) idx (10,  6) offsets=: 13654
pid (6, 5, 0) idx (10,  7) offsets=: 13655
pid (6, 5, 0) idx (10,  8) offsets=: 13656
pid (6, 5, 0) idx (10,  9) offsets=: 13657
pid (6, 5, 0) idx (10, 10) offsets=: 13658
pid (6, 5, 0) idx (10, 11) offsets=: 13659
```

# Case Study: Vector Addition With Interpreter

```
offsets= [[    0     1     2     3     4     5     6     7     8     9    10    11    12    13
     14    15]
 [  128   129   130   131   132   133   134   135   136   137   138   139   140   141
    142   143]
 [  256   257   258   259   260   261   262   263   264   265   266   267   268   269
    270   271]
 [  384   385   386   387   388   389   390   391   392   393   394   395   396   397
    398   399]
 [  512   513   514   515   516   517   518   519   520   521   522   523   524   525
    526   527]
 [  640   641   642   643   644   645   646   647   648   649   650   651   652   653
    654   655]
 [  768   769   770   771   772   773   774   775   776   777   778   779   780   781
    782   783]
 [  896   897   898   899   900   901   902   903   904   905   906   907   908   909
    910   911]
 [1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037
  1038 1039]
 [1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165
  1166 1167]
 [1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293
  1294 1295]
 [1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421
  1422 1423]
 [1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549
...
 [16240 16241 16242 16243 16244 16245 16246 16247 16248 16249 16250 16251
  16252 16253 16254 16255]
 [16368 16369 16370 16371 16372 16373 16374 16375 16376 16377 16378 16379
  16380 16381 16382 16383]]
```

# Triton Visualizer

# Education

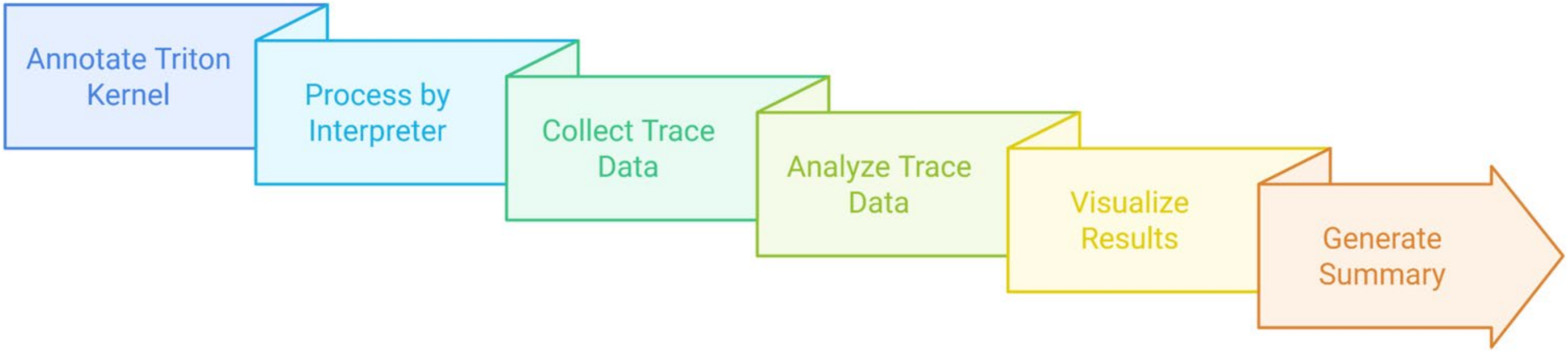
- How to educate the next-generation engineers and scientists on Triton knowledge?
- Parallel programming is hard
  - Though Triton has simplified the abstraction
- Performance optimization and correctness debugging are even harder

# Key Ideas

- Collect the trace of the interpreter
  - Easier than GPU binary / compiler instrumentation
  - No need to access real GPUs
- Design an interactive visualizer
- Design a set of questions for students to practice

# Triton-Viz Workflow

- From @triton_viz.trace to visualization reports



# Triton Puzzles

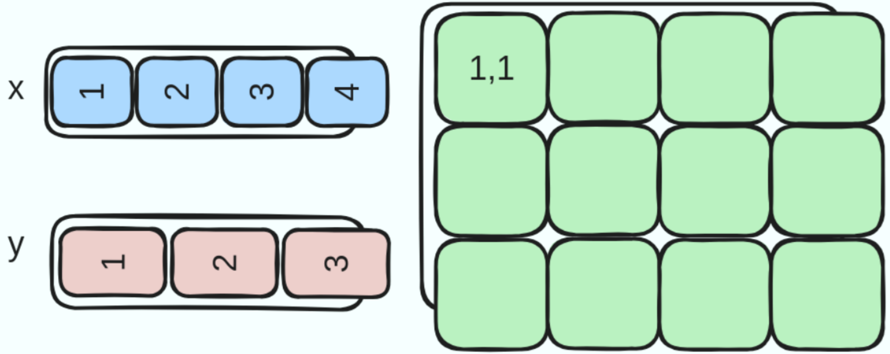
- Teach you how to use Triton from first principles in an interactive fashion
- Collaborated with Sasha Rush @ Cornell Tech

Puzzle 4: Outer Vector Add Block

Add a row vector to a column vector.

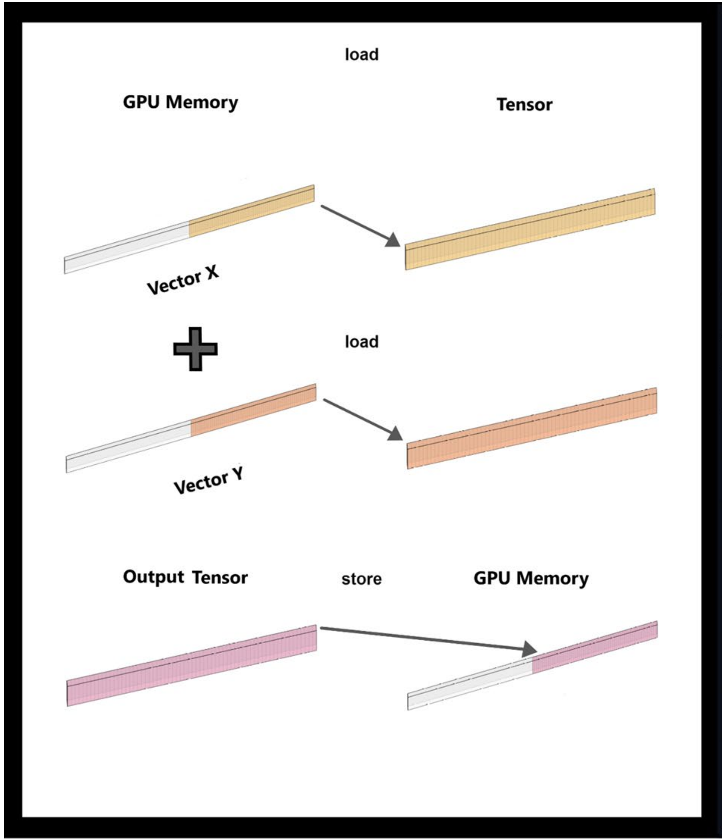
Uses two program block axes. Block size `B0` is always less than the vector `x` length `N0`. Block size `B1` is always less than vector `y` length `N1`.

$$z_{j,i} = x_i + y_j \text{ for } i = 1 \ldots N_0,\ j = 1 \ldots N_1$$




[Triton-Puzzles/README.md at main · srush/Triton-Puzzles]

# Triton-Viz Visualization




Program Id 0 1

Program Id 1 0

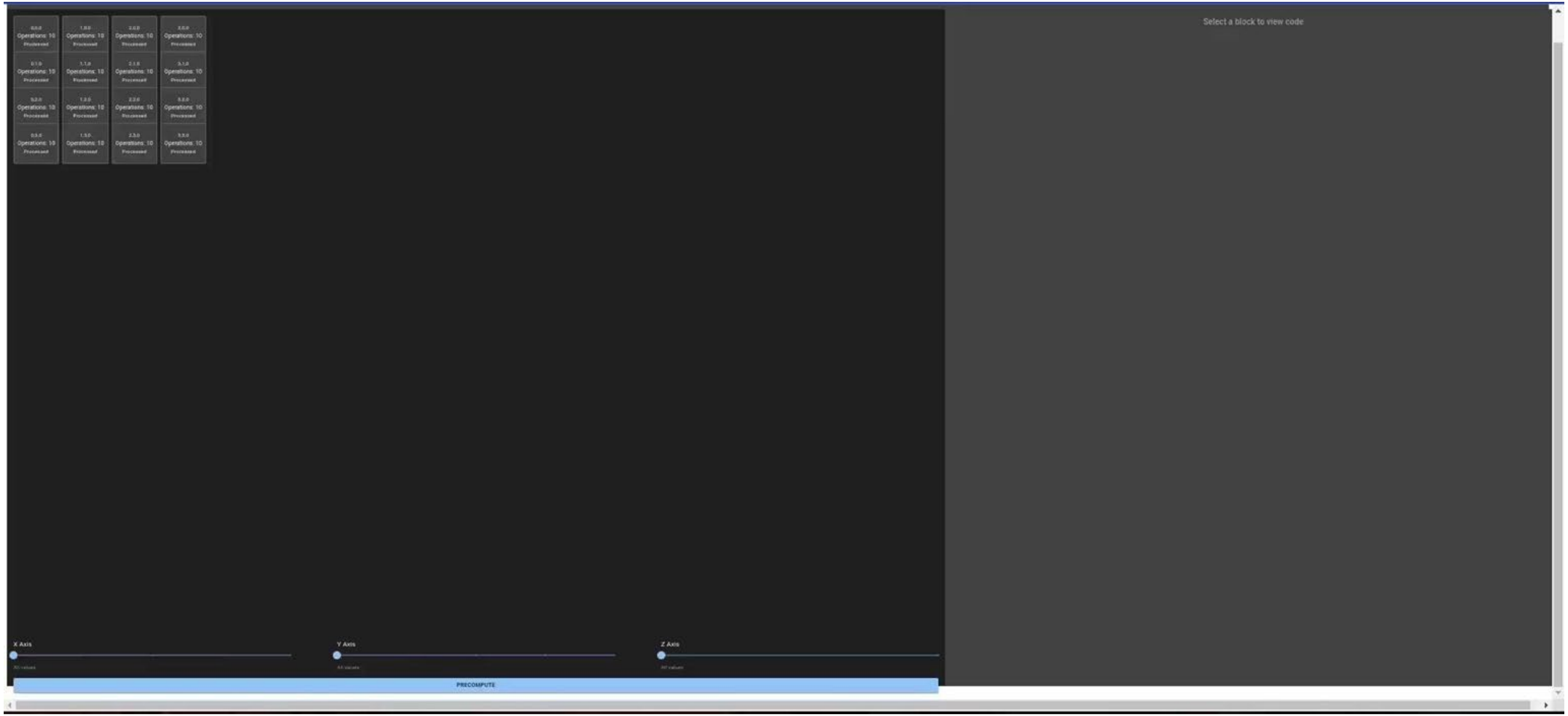
Program Id 2 0

Precompute

## Analysis

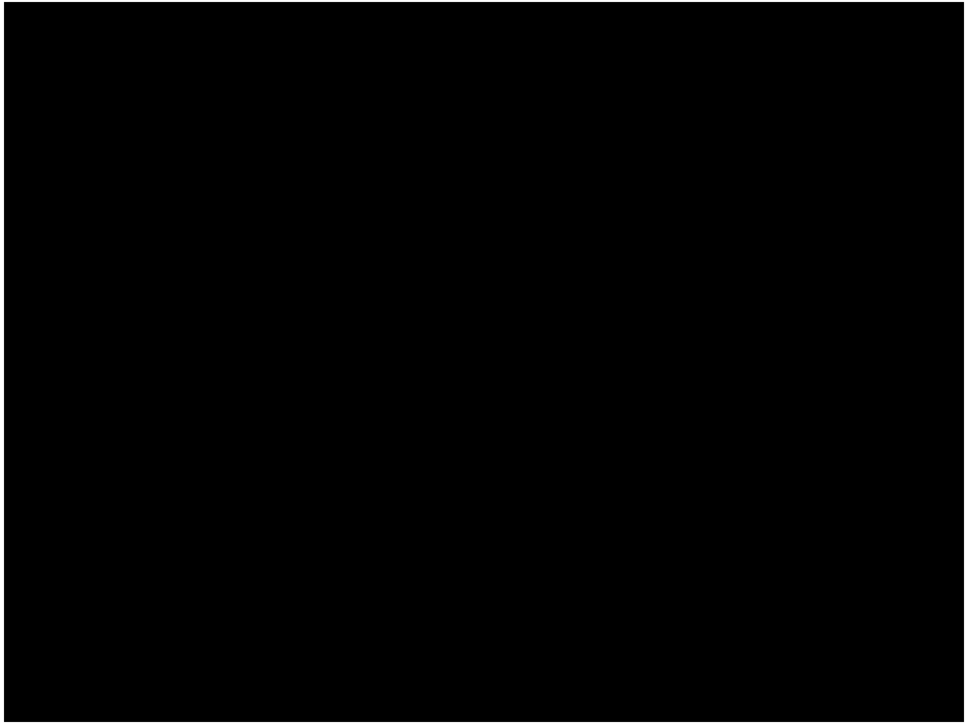
| Metric | Value |
|---|---|
| Grid Size | (2, 1, 1) |
| MakeRange | 2 |
| BinaryOp | 8 |
| Load | 4 |
| Store | 2 |
| Total number of bytes loaded | 1024 |
| Masked Load Ratio | 1.0 |
| Total number of bytes stored | 512 |
| Masked Store Ratio | 1.0 |

# Triton-Viz 2.0 Demo-1




Credits to Daniyal Khan

# Triton-Viz 2.0 Demo-2

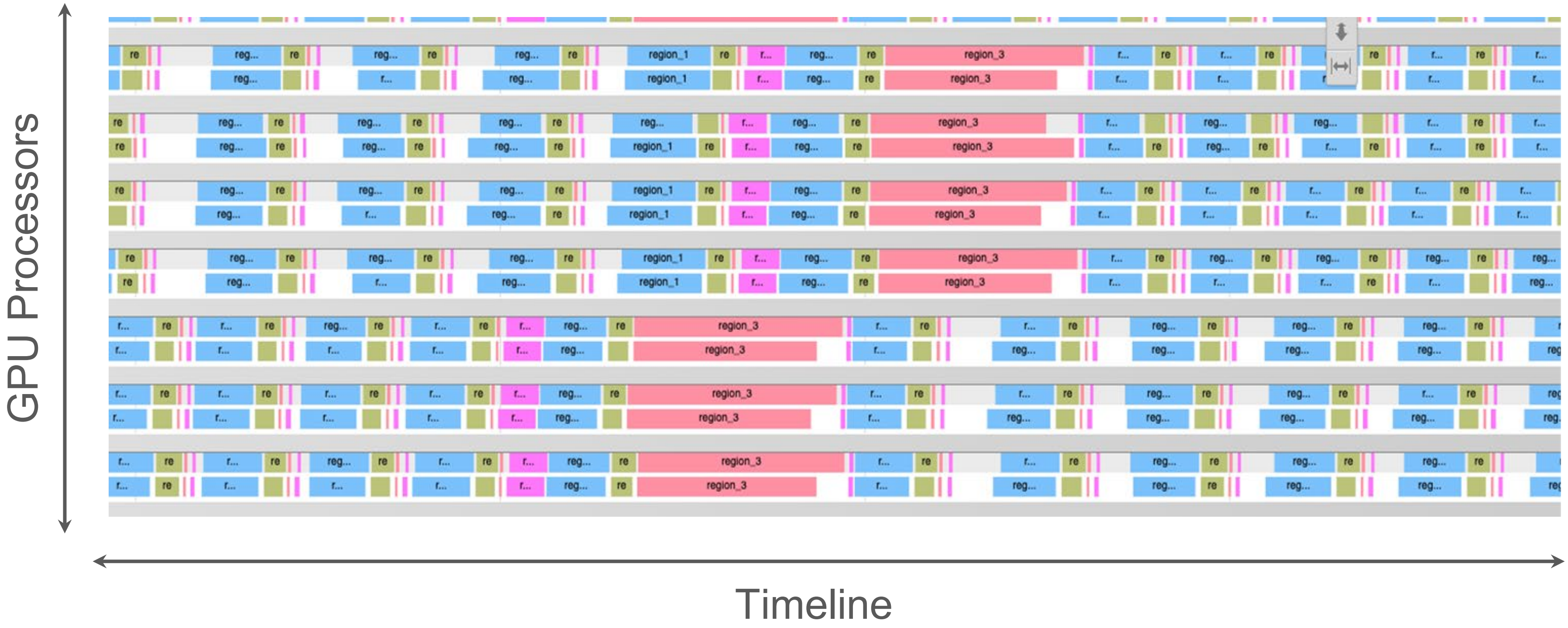


# Ongoing Work

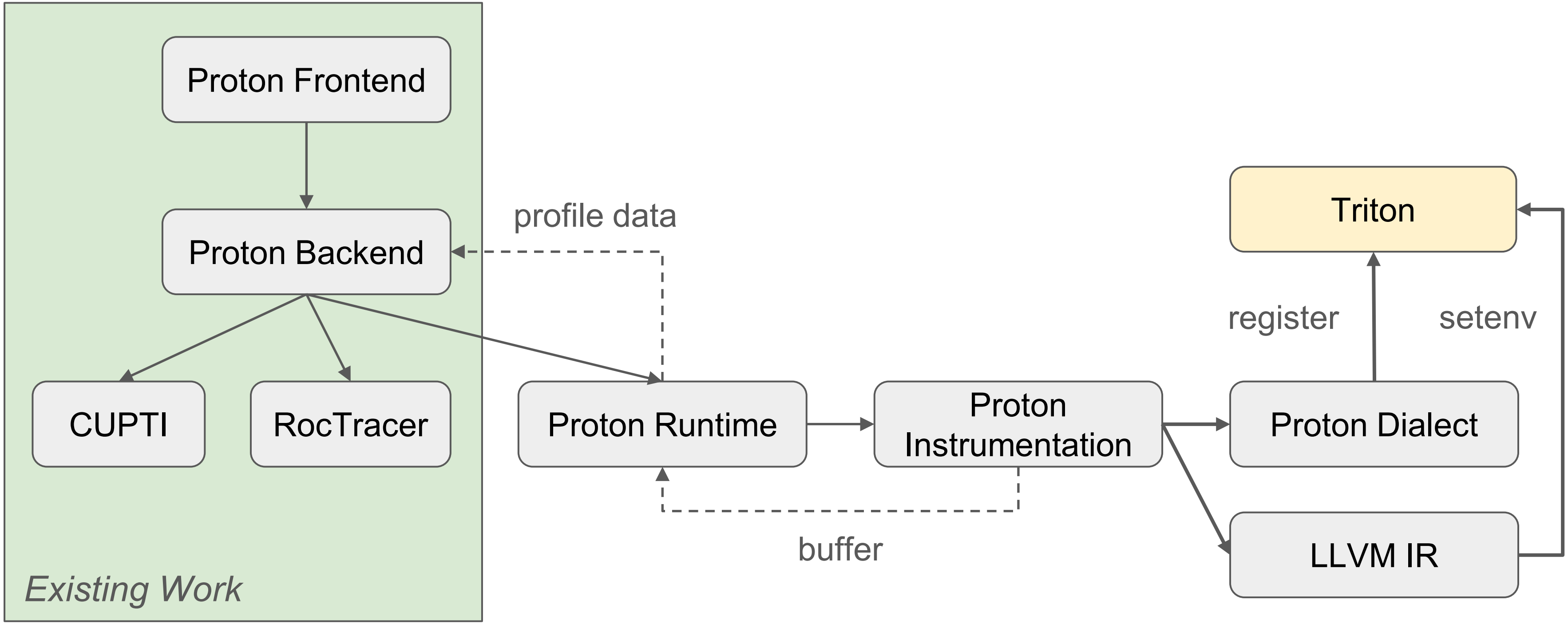
# Proton Instrumentation

- CUPTI and RocTracer are powerful but may not fully address our needs
- Why custom instrumentation?
  - Cross-Platform Support: One engine for multiple GPUs/accelerators
  - Reusable Client Interface: Simplify development across different platforms
  - Extended Metrics: Capture data unavailable through vendor tools
- Collaborating with Meta (Yuanwei Fang) and AMD (Corbin Robeck)

# Fine-grained GPU Trace



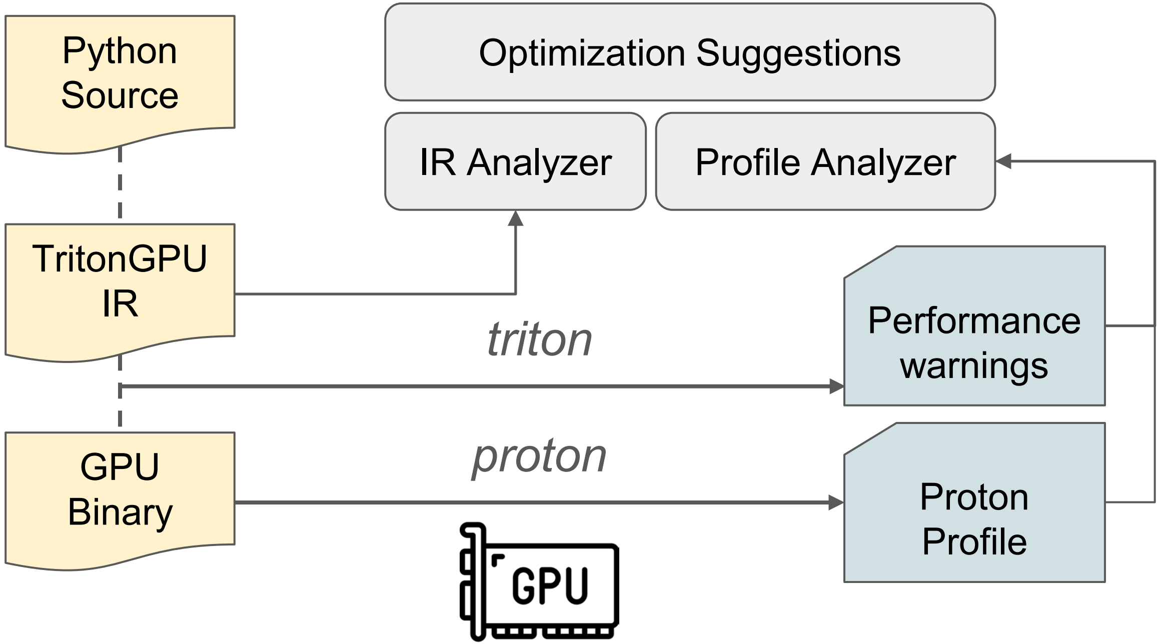
# Proposed Solution



# Performance Analyzer

- Incorporate multi-level IR analysis into proton
- Associate compile time warnings with runtime performance metrics
- Provide actionable optimizations for users
- Provide problem diagnostic insights for compiler developers

# Proposed Solution



# Summary

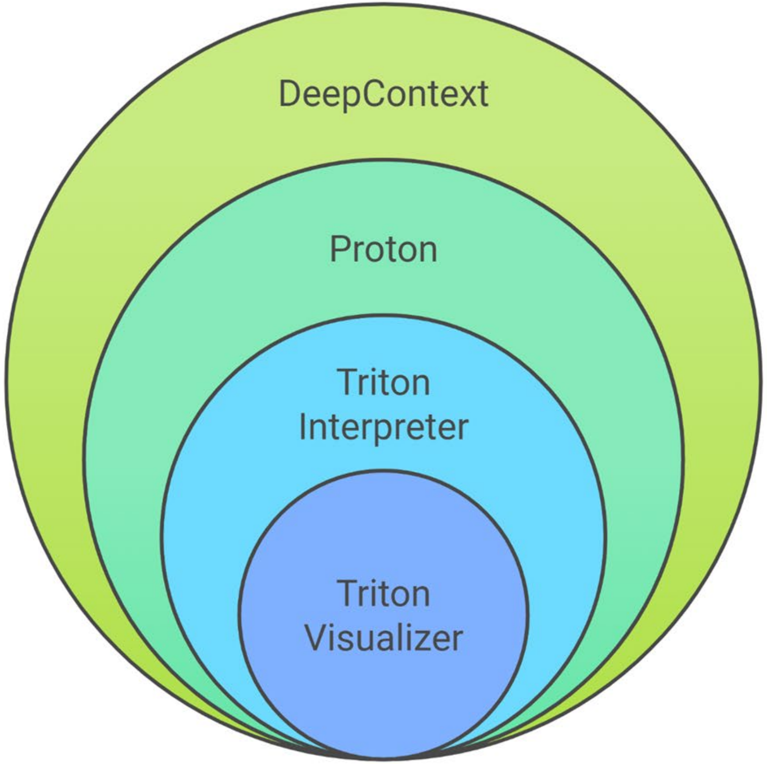
# Put it Together

Comprehensive framework profiler for end-to-end analysis

Operator profiler for detailed performance metrics

Advanced operator debugger for kernel-level operations

Simplified educational debugger and profiler



# Q&A