ValueExpert: Exploring Value Patterns in GPU-accelerated Applications

Keren Zhou*[1], Yueming Hao*[2]

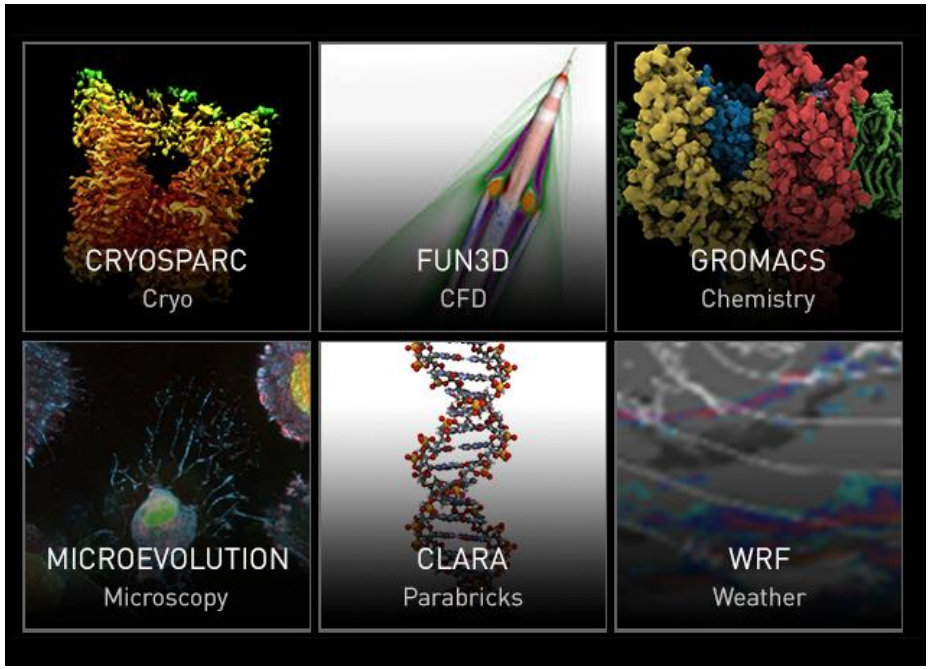
John Mellor-Crummey[1], Xiaozhu Meng[1], Xu Liu[2]

[1]Rice University

[2]North Carolina State University

*Co-first authors

GPUs are Extensively Used for Acceleration



DEEP GRAPH LIBRARY

MATLAB

Microsoft Cognitive Toolkit

mxnet

PaddlePaddle

PyTorch

TensorFlow

torch

Wolfram Language

intel

NVIDIA

AMD

Performance is important!

Image source:
https://blogs.nvidia.com/blog/2021/06/25/industrial-hpc-revolution/
Intel's Nvidia GPU licensing deal ends next month - Legal - News - HEXUS.net

GPU Performance Tools

- Existing tools apply *hotspot* analysis

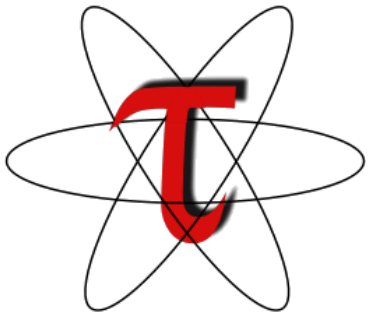
API Statistics

Reset | Export to CSV

Enter filter, e.g. $('Name') == 'MyAPIName'

Name	Number of Calls	Total Duration	Average Duration	Minimum Duration	Maximum Duration
cudaMemcpy	1	71.41us	71.41us	71.41us	71.41us
cudaMalloc	2	123.224ms	61.612ms	41.294us	123.183ms
cudaGetLastError	1	931ns	931ns	931ns	931ns
cuModuleGetFunction	4	12.106us	3.0265us	931ns	9.003us
cuMemcpyHtoD_v2	1	29.806us	29.806us	29.806us	29.806us
cuMemAlloc_v2	2	428.154us	214.077us	9.935us	418.219us
cuInit	1	191.568ms	191.568ms	191.568ms	191.568ms
cuDriverGetVersion	1	8.693us	8.693us	8.693us	8.693us
cuDeviceTotalMem_v2	3	34.152us	11.384us	8.693us	15.524us
cuDevicePrimaryCtxRetain	1	120.4ms	120.4ms	120.4ms	120.4ms
cuDeviceGetUuid	3	1.55us	516.667ns	310ns	620ns
cuDeviceGetName	3	1.861us	620.333ns	310ns	931ns
cuDeviceGetCount	1	4.657us	4.657us	4.657us	4.657us
cuDeviceGetAttribute	279	2.88614ms	10.3446us	0ns	498.013us

An example profile from Nsight Compute

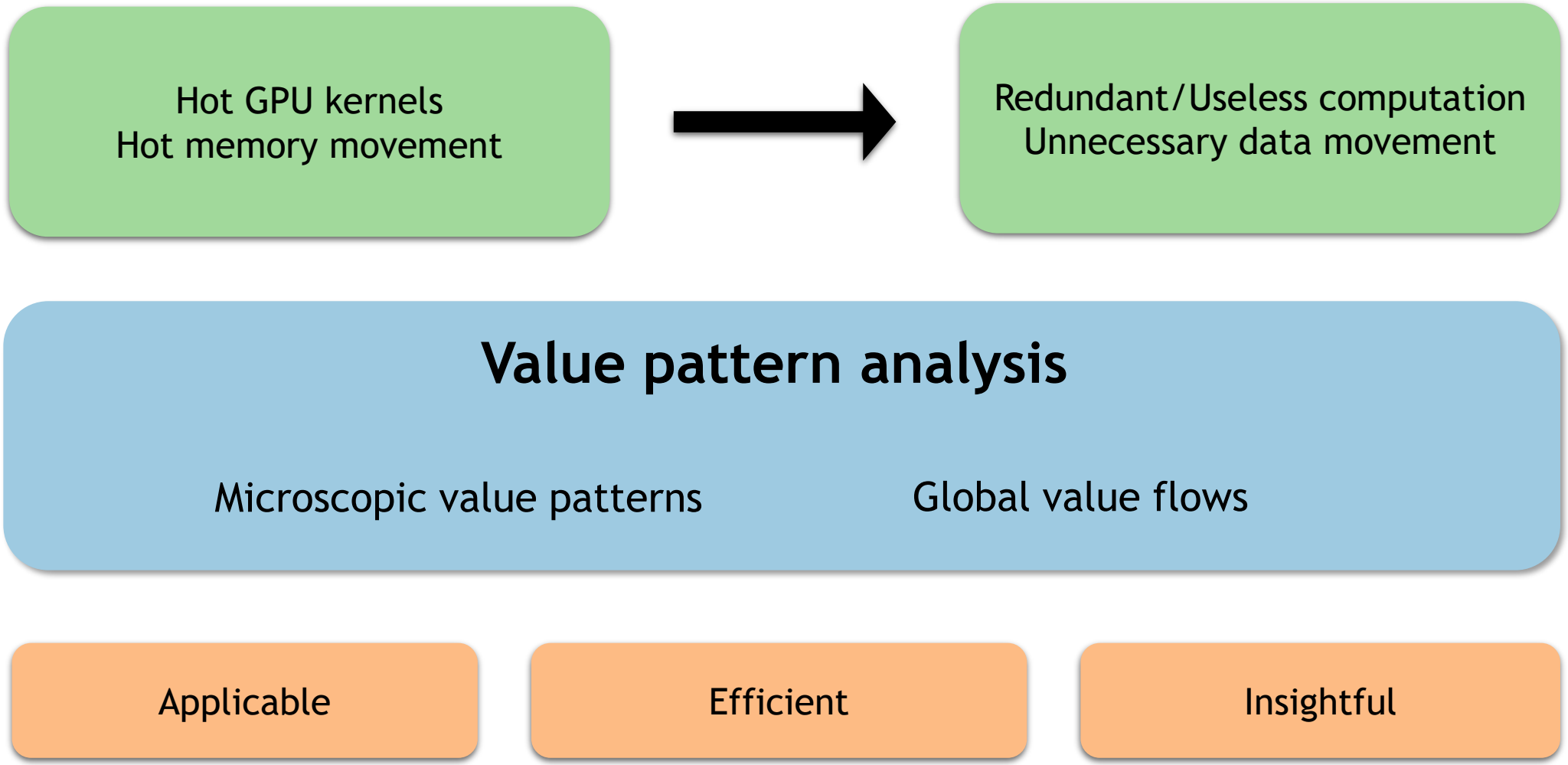


A Motivating Example from PyTorch

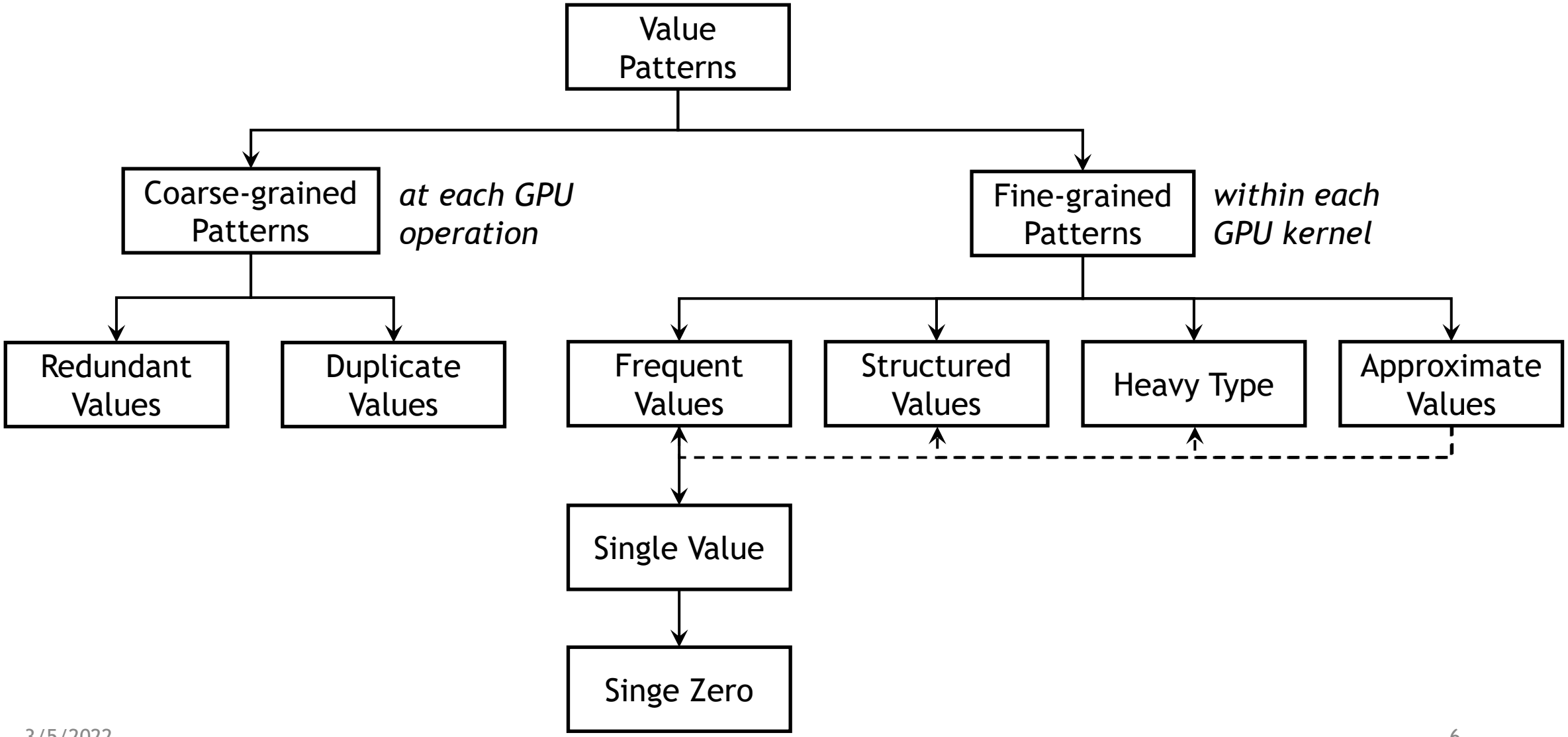
```
void replication_pad3d_backward_out_cuda_template(...) {
  gradInput.resize_as_(input);
  gradInput.zero_();
  ...
}
Tensor replication_pad3d_backward_cuda(...) {
  auto gradInput = at::zeros_like(input, LEGACY_CONTIGUOUS_MEMORY_FORMAT);

  replication_pad3d_backward_out_cuda_template(gradInput,
    gradOutput, input, paddingSize);
}
```

- Redundant value updates on the **gradInput** array
  - Replace *zeros_like* with *empty_like* improves the operator by **1.08×**

ValueExpert Design Principle

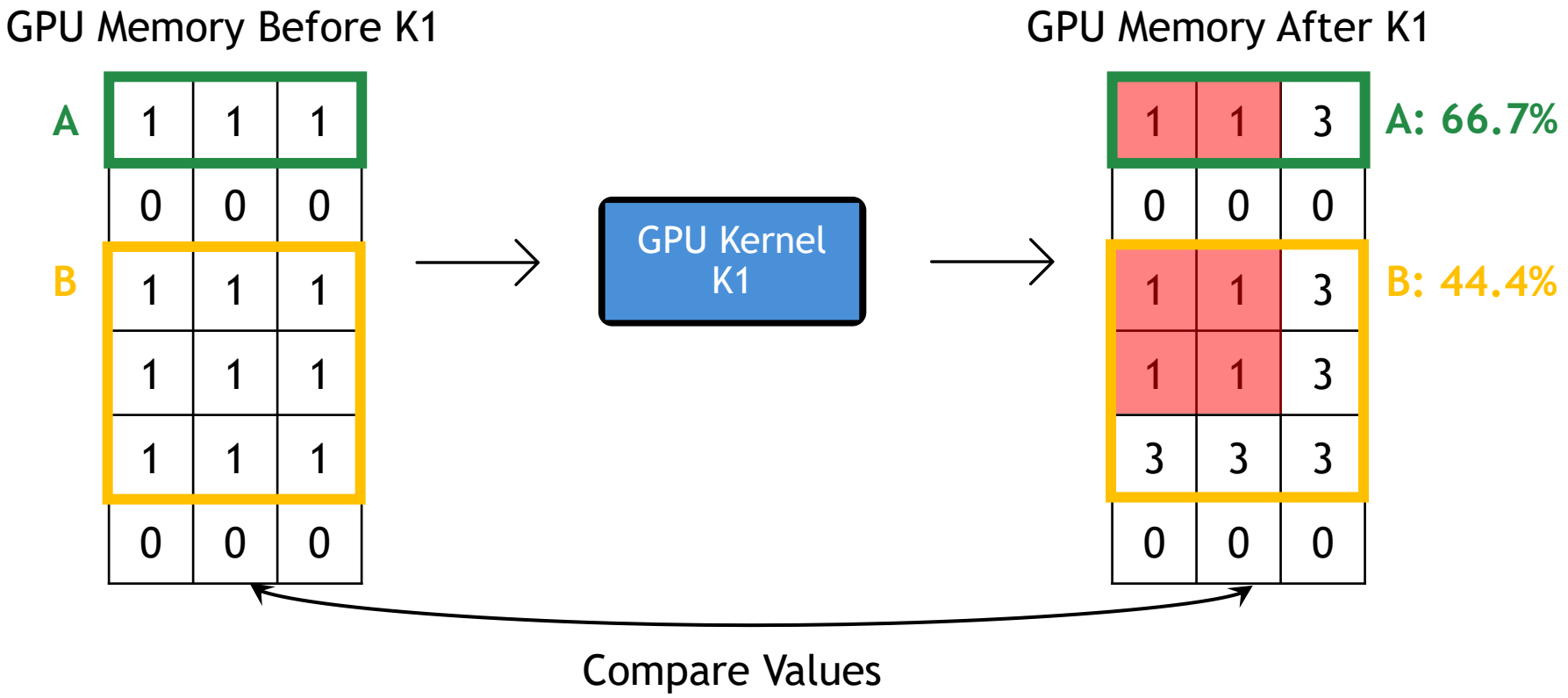


Value Pattern Categorization



Redundant Values

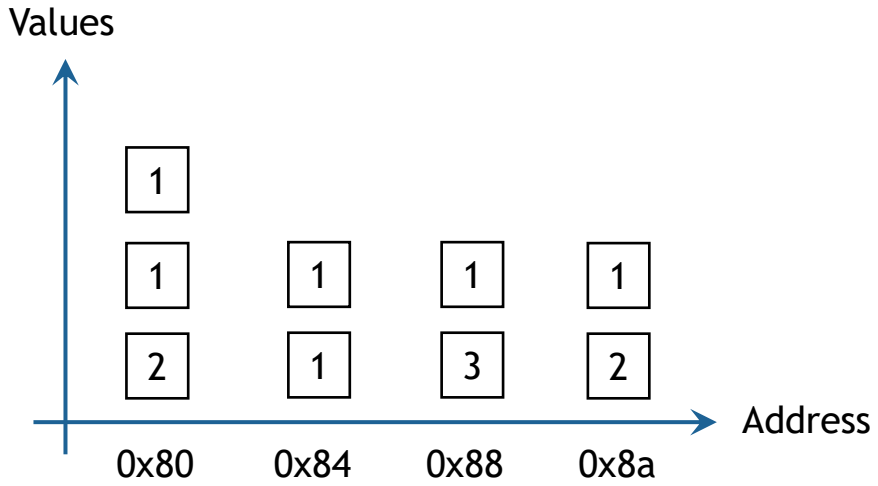
- Intercept memory allocation APIs to track memory objects
- Record value changes before and after a GPU Kernel



Frequent Values

- Some values are accessed more frequently than others

Value Accesses



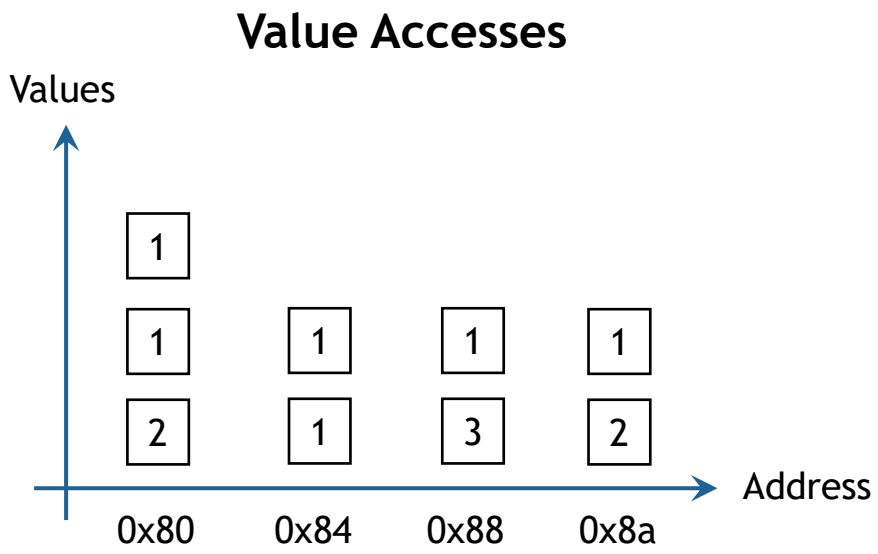
Value Distribution

Value	Count	Ratio
1	6	66.7%
2	2	22.2%
3	1	11.1%

> 50%

Heavy Type

- Values can be represented by narrowed data types



Value Distribution

Value	Count	Ratio
1	6	66.7%
2	2	22.2%
3	1	11.1%

int8: $[-2^{-7}, 2^7 - 1]$

Value Patterns are Pervasive

- ***8*** value patterns in ***10*** benchmarks and ***9*** applications

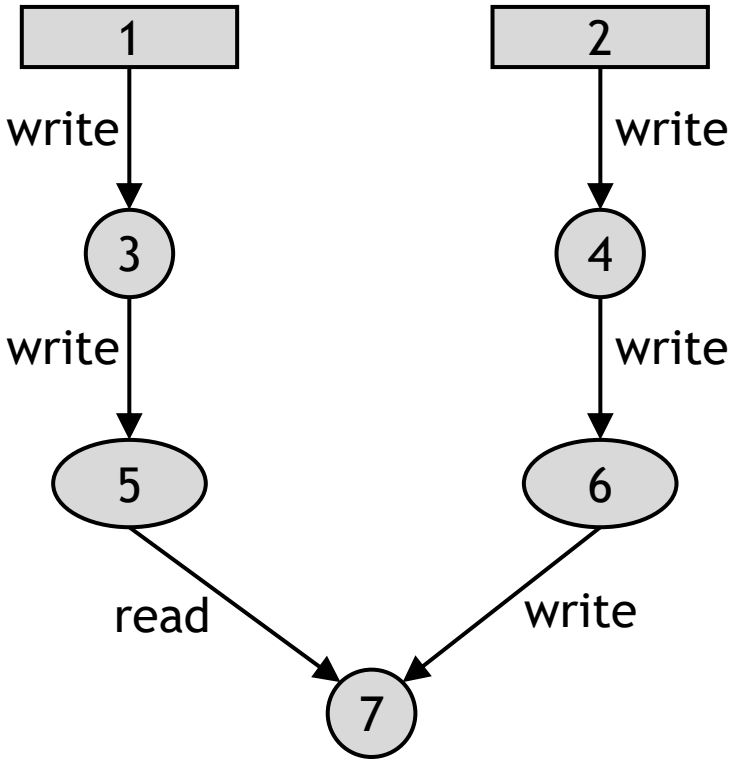
Applications	Redundant Values	Duplicate Values	Dense Values	Single Value	Single Zero	Heavy Type	Structured Values	Approximate Values
Rodinia/bfs	✓		✓	✓		✓		
Rodinia/backprop	✓	✓			✓			
Rodinia/sradv1		✓	✓	✓		✓	✓	
Rodinia/hotspot			✓					✓
Rodinia/pathfinder	✓		✓			✓		
Rodinia/cfd	✓		✓					
Rodinia/huffman	✓	✓	✓	✓		✓		
Rodinia/lavaMD	✓					✓		
Rodinia/hotspot3D								✓
Rodinia/streamcluster	✓							
Darknet	✓	✓	✓	✓	✓			
QMCPACK	✓							
Castro	✓							
BarraCUDA	✓		✓					
PyTorch-Deepwave	✓			✓	✓			
PyTorch-Bert	✓							
PyTorch-Resnet50	✓			✓	✓			
NAMD	✓				✓	✓		
LAMMPS	✓		✓					

Construct Value Flow Graph

Program

```
1  cudaMalloc(&A_dev, N * sizeof(int));
2  cudaMalloc(&B_dev, N * sizeof(int));
3  cudaMemset(A_dev, 0, N * sizeof(int));
4  cudaMemset(B_dev, 0, N * sizeof(int));
5  set_zeros<<<1, N>>>(A_dev, N/4);
6  set_zeros<<<1, N>>>(B_dev, N/4);
7  cudaMemcpy(B_dev, A_dev, N * sizeof(int),
   cudaMemcpyDeviceToDevice);
```

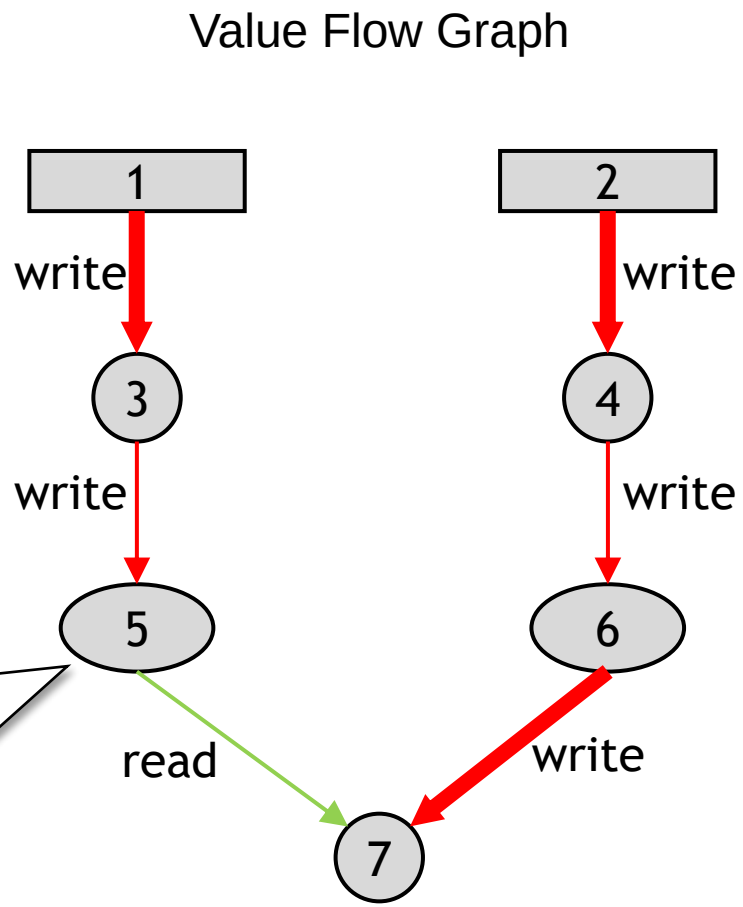
Value Flow Graph



Annotate Information on Graph

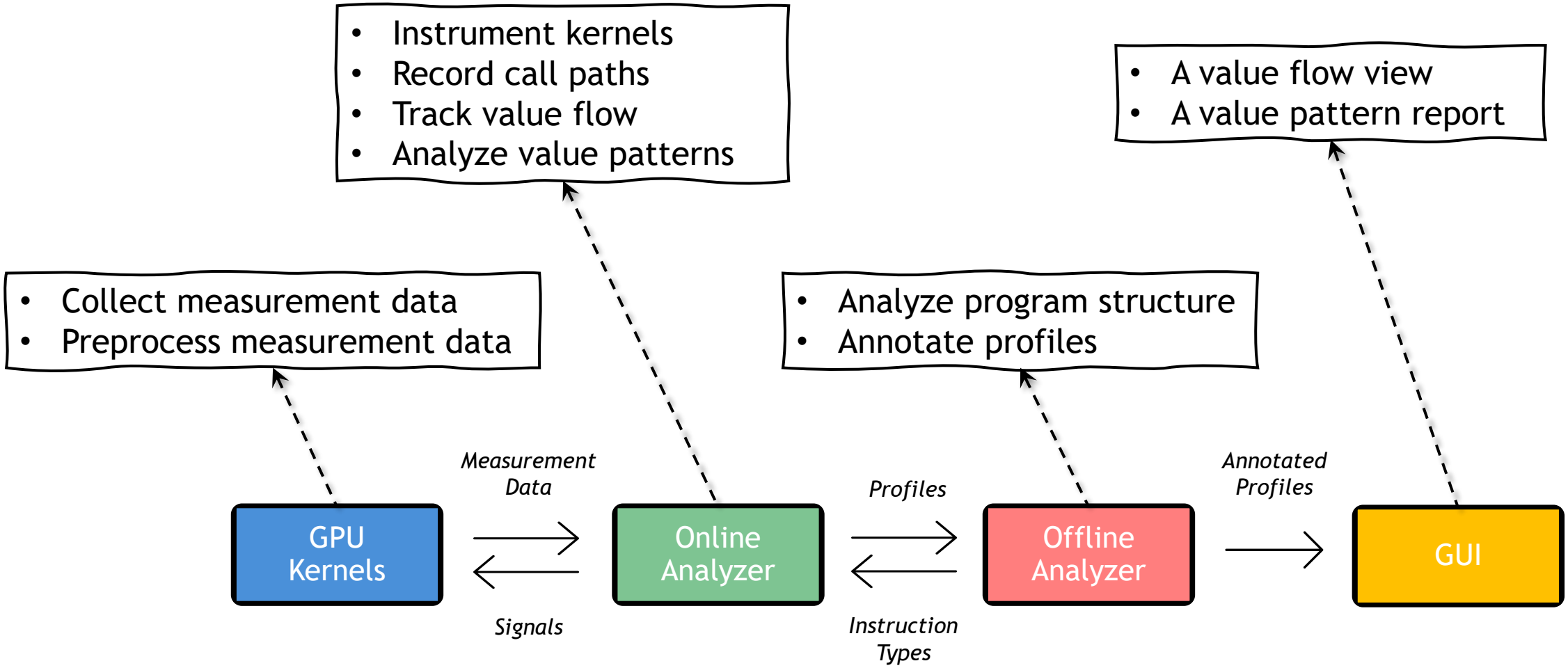
- Call path at each GPU operation
- Coarse-grained patterns
  - Edge width/color
  - Vertex size/color
- Fine-grained patterns
  - Use vertex ID to lookup value patterns

main.cu: 20	foo(void)
main.cu: 10	bar(void) [inline]
main.cu: 5	set_zeros(int *, unsigned long)
cudafe1.stub.c: 13	cudaLaunchKernel<char>(…)
cuda_runtime.h: 210	cudaLaunchKernel



All values are zeros

ValueExpert Workflow



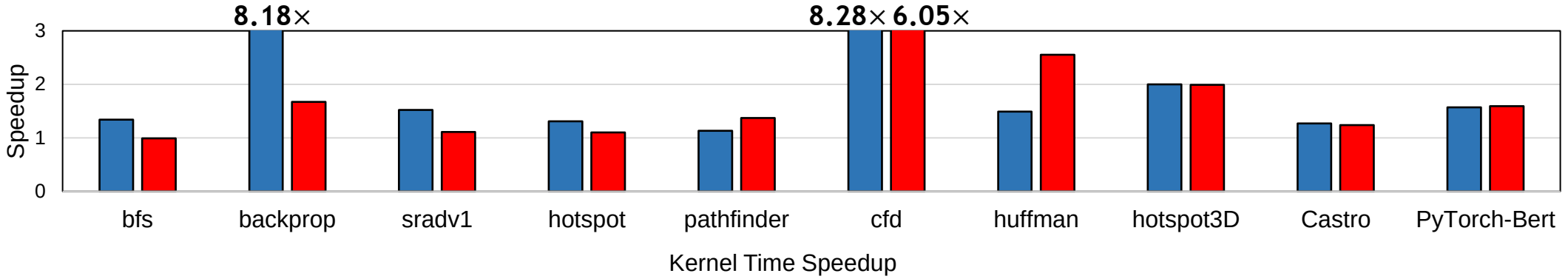
Making ValueExpert Practical

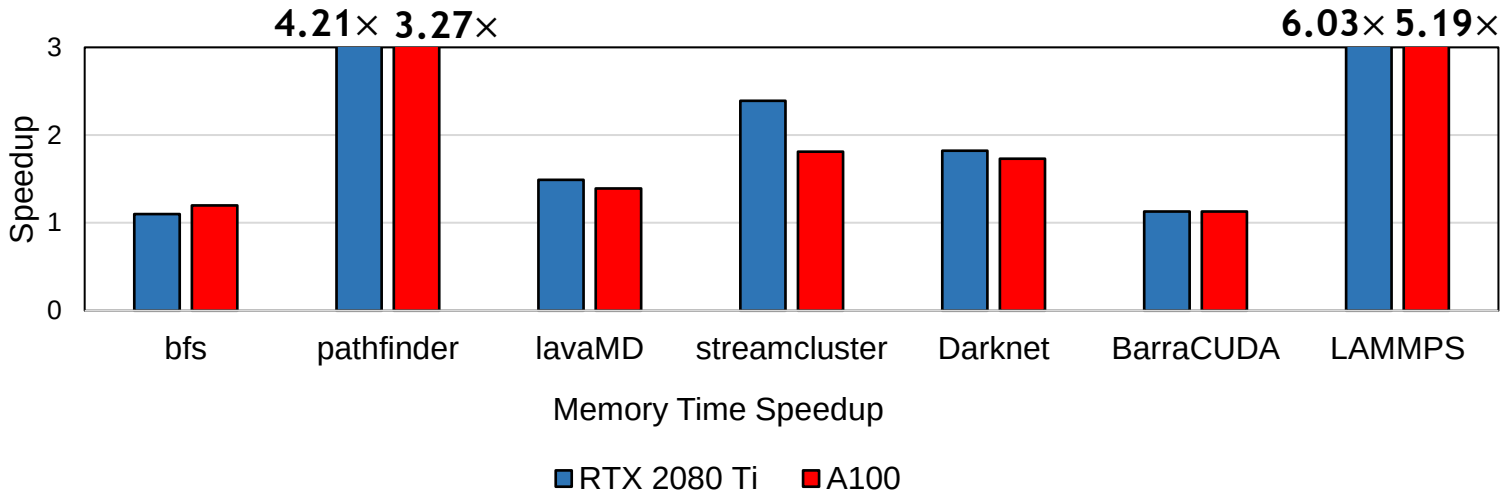
- Accelerating analysis using GPUs
- Employing several sampling mechanisms
- Visualizing large profiles

Evaluation Platforms

- AMD CPU + NVIDIA A100 GPU (A100)
  - CUDA/11.2
  - GCC/8.3.0
- Intel CPU + NVIDIA GTX 2080 Ti GPU (RTX 2080 Ti)
  - CUDA/11.2
  - GCC/9.3.0

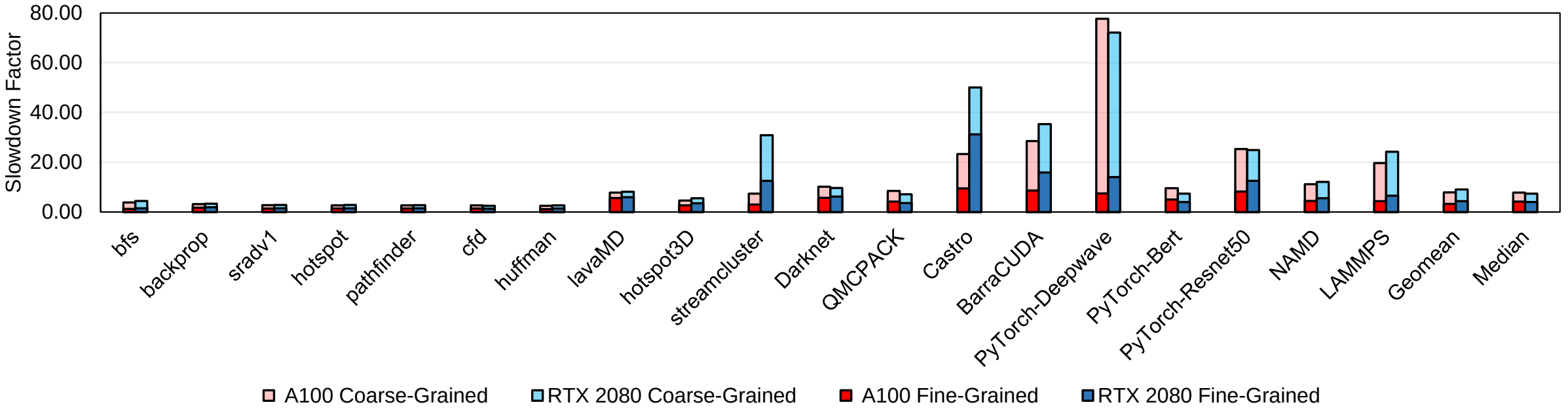
Speedups with Optimizations Guided by ValueExpert



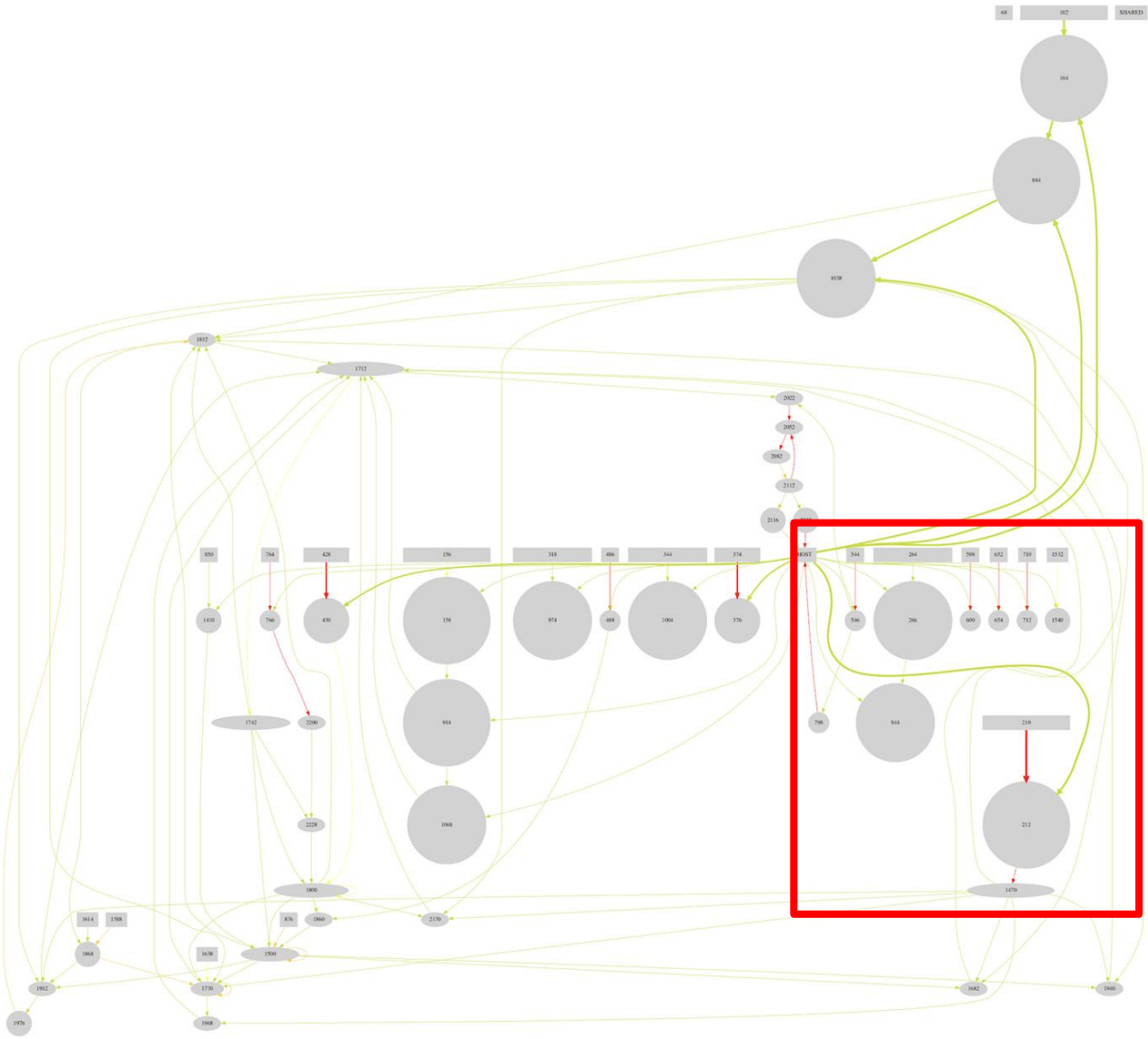


Overhead

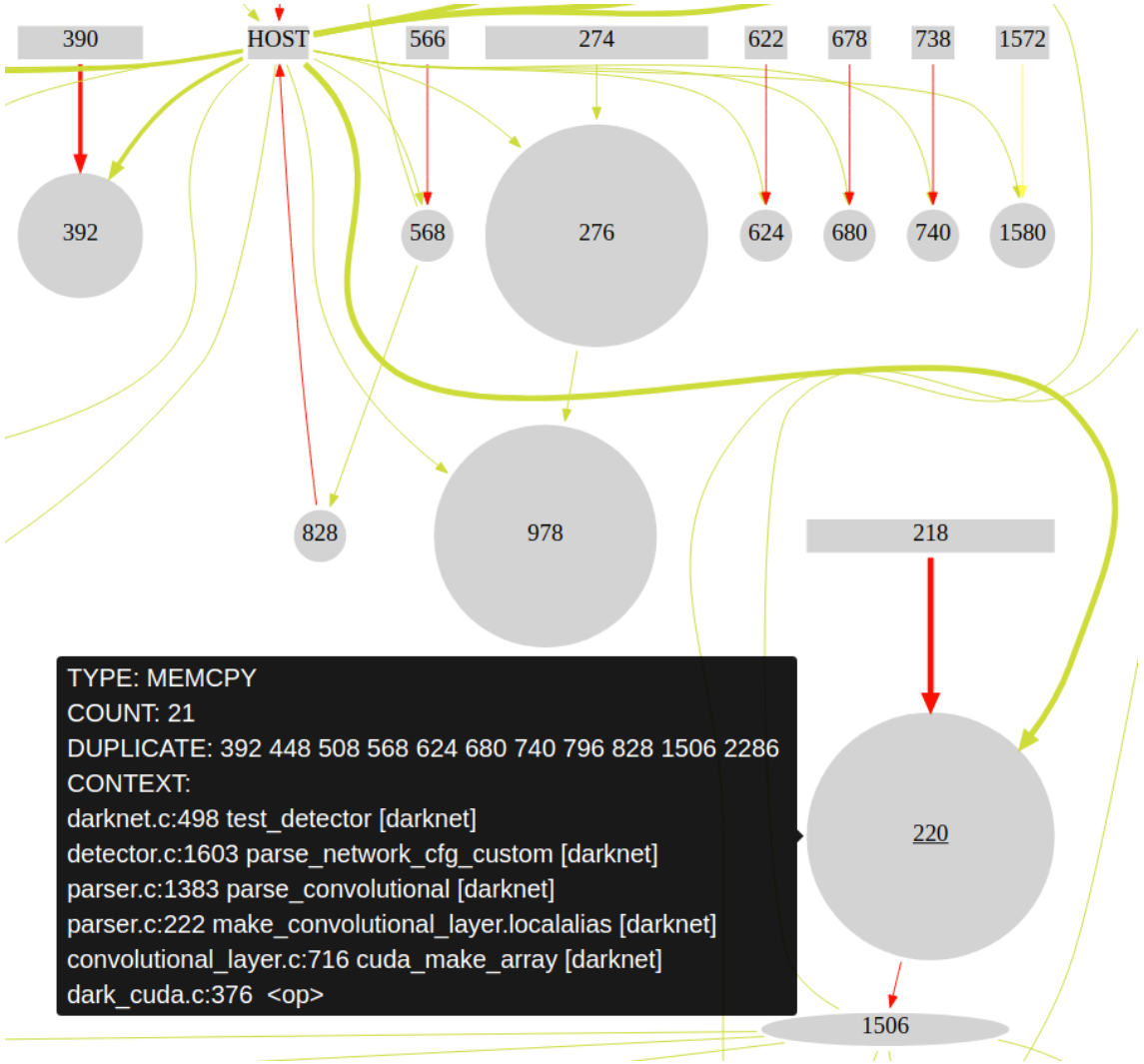
- Median overhead
  - **7.35**× on RTX 2080 Ti
  - **7.81**× on A100
- Factors that affect overhead
  - #GPU kernels
  - #Non-coalesced memory accesses



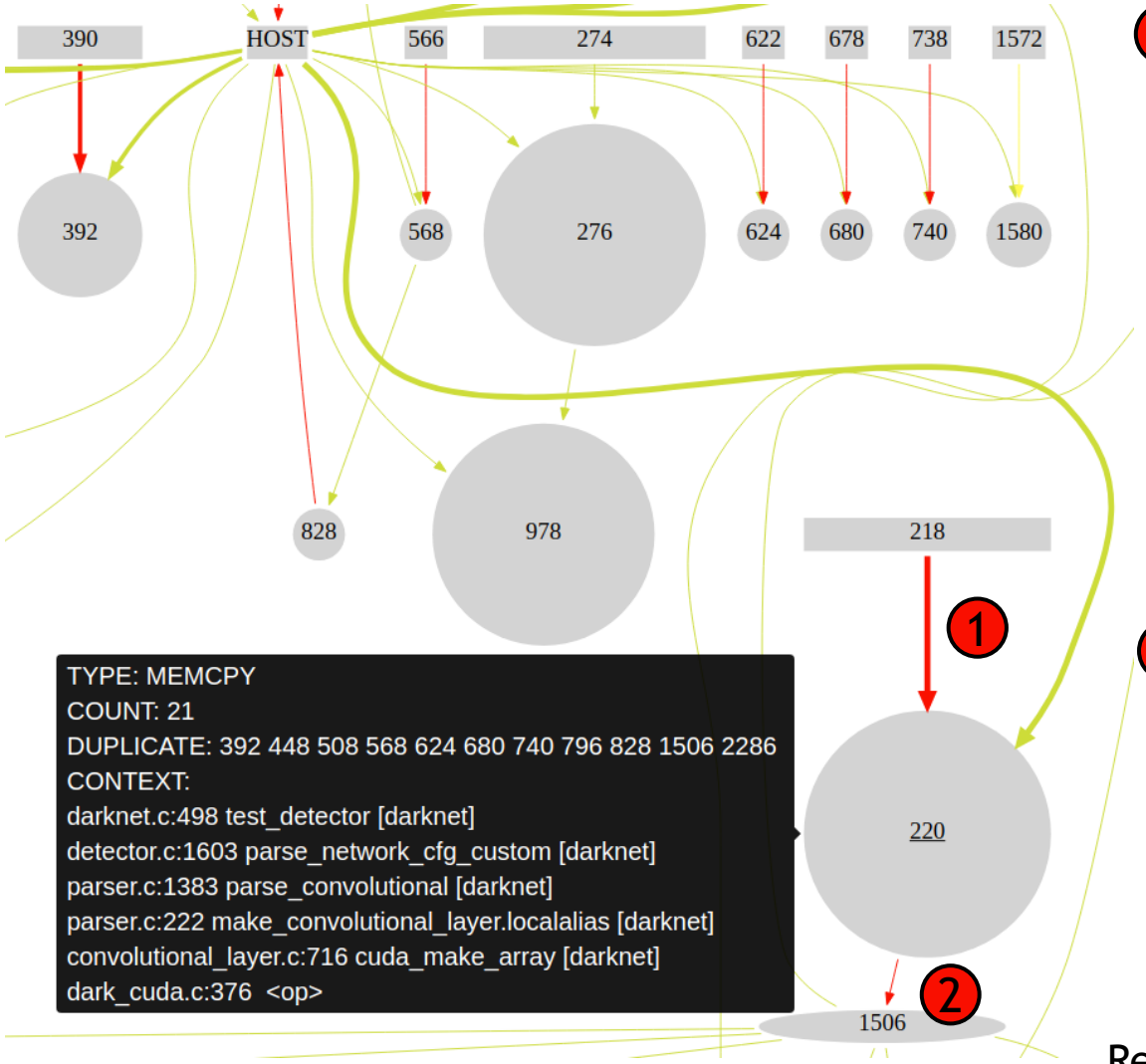
Darknet: Initial Value Flow Graph



Darknet: Graph Pruning



Darknet: Optimizations



① Unnecessary CPU-GPU data transfer

Reduce **84.2%** CPU-GPU memory traffic

② Redundant GPU instructions

Reduce **4.1%** load instructions and **10.6%** store instructions

Summary

- ValueExpert: the first tool analyzing value patterns
  - Applicable
  - Efficient
  - Insightful
- Code available at https://github.com/GVProf/GVProf