DrGPU: A Top-Down Profiler for GPU

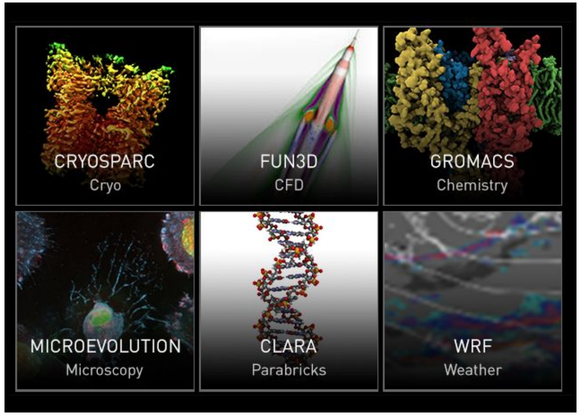
Yueming Hao[1], Nikhil Jain[2], Rob Van Der Wijngaart[2], Nirmal Saxena[2], Yuanbo Fan[3], Xu Liu[1]

[1]North Carolina State University

[2]NVIDIA Corporation

[3]Tenstorrent Incorporated

GPUs are Broadly Used for Acceleration



GPU Programming

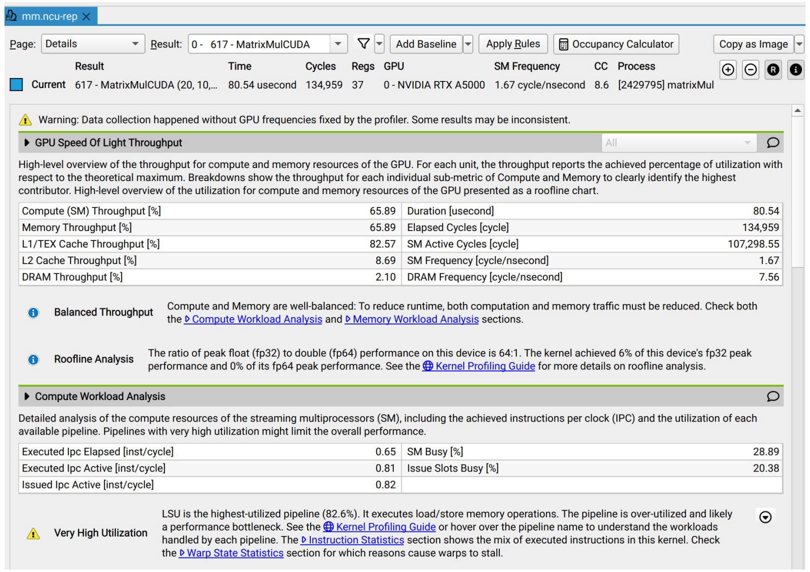
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    N = 1024
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

GPU Kernel

# of GPU threads

1 warp = 32 threads

Existing GPU Performance Tools



mm.ncu-rep

Page: Details | Result: 0 - 617 - MatrixMulCUDA | Add Baseline | Apply Rules | Occupancy Calculator | Copy as Image

	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	617 - MatrixMulCUDA (20, 10,...	80.54 usecond	134,959	37	0 - NVIDIA RTX A5000	1.67 cycle/nsecond	8.6	[2429795] matrixMul

Warning: Data collection happened without GPU frequencies fixed by the profiler. Some results may be inconsistent.

GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	65.89	Duration [usecond]	80.54
Memory Throughput [%]	65.89	Elapsed Cycles [cycle]	134,959
L1/TEX Cache Throughput [%]	82.57	SM Active Cycles [cycle]	107,298.55
L2 Cache Throughput [%]	8.69	SM Frequency [cycle/nsecond]	1.67
DRAM Throughput [%]	2.10	DRAM Frequency [cycle/nsecond]	7.56

Balanced Throughput Compute and Memory are well-balanced: To reduce runtime, both computation and memory traffic must be reduced. Check both the Compute Workload Analysis and Memory Workload Analysis sections.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved 6% of this device's fp32 peak performance and 0% of its fp64 peak performance. See the Kernel Profiling Guide for more details on roofline analysis.

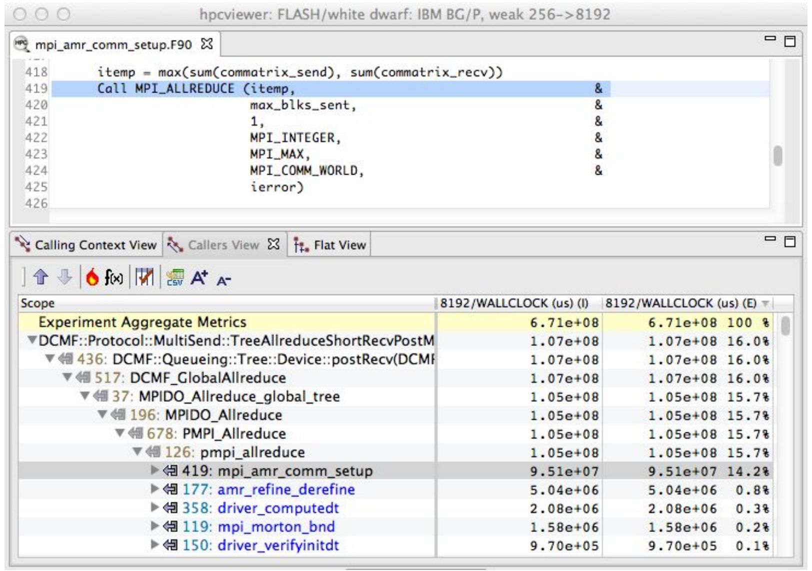
Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	0.65	SM Busy [%]	28.89
Executed Ipc Active [inst/cycle]	0.81	Issue Slots Busy [%]	20.38
Issued Ipc Active [inst/cycle]	0.82		

Very High Utilization LSU is the highest-utilized pipeline (82.6%). It executes load/store memory operations. The pipeline is over-utilized and likely a performance bottleneck. See the Kernel Profiling Guide or hover over the pipeline name to understand the workloads handled by each pipeline. The Instruction Statistics section shows the mix of executed instructions in this kernel. Check the Warp State Statistics section for which reasons cause warps to stall.

NVIDIA Nsight Compute



hpcviewer: FLASH/white dwarf: IBM BG/P, weak 256->8192

mpi_amr_comm_setup.F90

```
418        itemp = max(sum(commatrix_send), sum(commatrix_recv))
419        Call MPI_ALLREDUCE (itemp,                                  &
420                            max_blks_sent,                          &
421                            1,                                      &
422                            MPI_INTEGER,                            &
423                            MPI_MAX,                                &
424                            MPI_COMM_WORLD,                         &
425                            ierror)
426
```

Calling Context View | Callers View | Flat View

Scope	8192/WALLCLOCK (us) (I)	8192/WALLCLOCK (us) (E)
Experiment Aggregate Metrics	6.71e+08	6.71e+08 100 %
▼DCMF::Protocol::MultiSend::TreeAllreduceShortRecvPostM	1.07e+08	1.07e+08 16.0%
▼436: DCMF::Queueing::Tree::Device::postRecv(DCMF	1.07e+08	1.07e+08 16.0%
▼517: DCMF_GlobalAllreduce	1.07e+08	1.07e+08 16.0%
▼37: MPIDO_Allreduce_global_tree	1.05e+08	1.05e+08 15.7%
▼196: MPIDO_Allreduce	1.05e+08	1.05e+08 15.7%
▼678: PMPI_Allreduce	1.05e+08	1.05e+08 15.7%
▼126: pmpi_allreduce	1.05e+08	1.05e+08 15.7%
▶419: mpi_amr_comm_setup	9.51e+07	9.51e+07 14.2%
▶177: amr_refine_derefine	5.04e+06	5.04e+06 0.8%
▶358: driver_computedt	2.08e+06	2.08e+06 0.3%
▶119: mpi_morton_bnd	1.58e+06	1.58e+06 0.2%
▶150: driver_verifyinitdt	9.70e+05	9.70e+05 0.1%

HPCToolkit

Existing tools apply high-level *hotspot* analysis

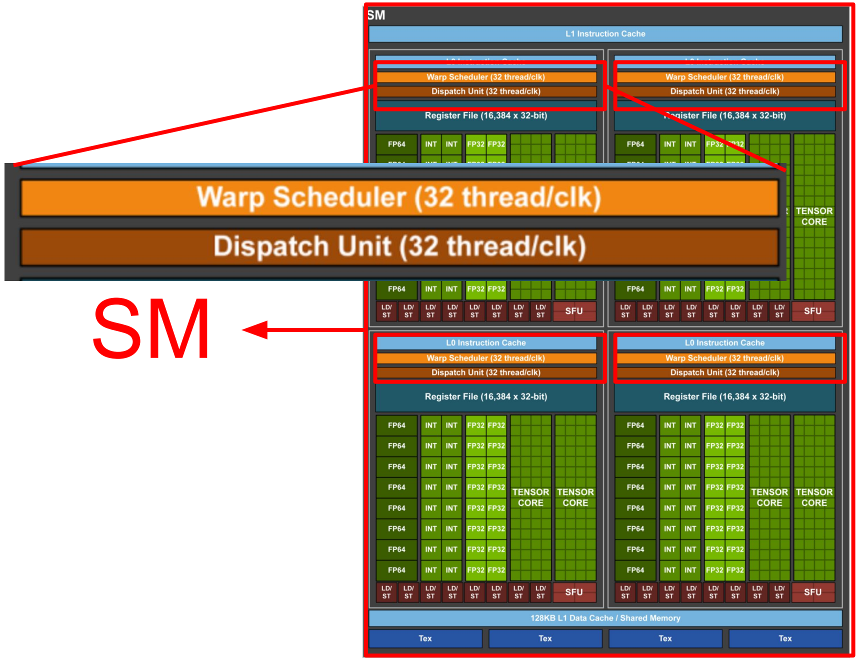
DrGPU Contribution

DrGPU

- ➢ tells you where GPUs waste on stalling by a top-down tree

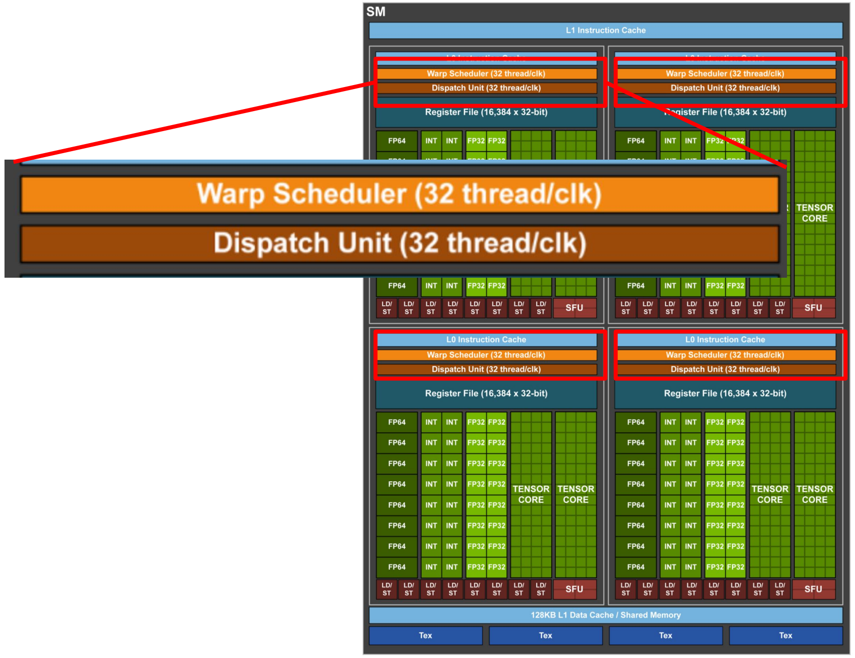
- ➢ provides analysis and optimization guidance for non-experts

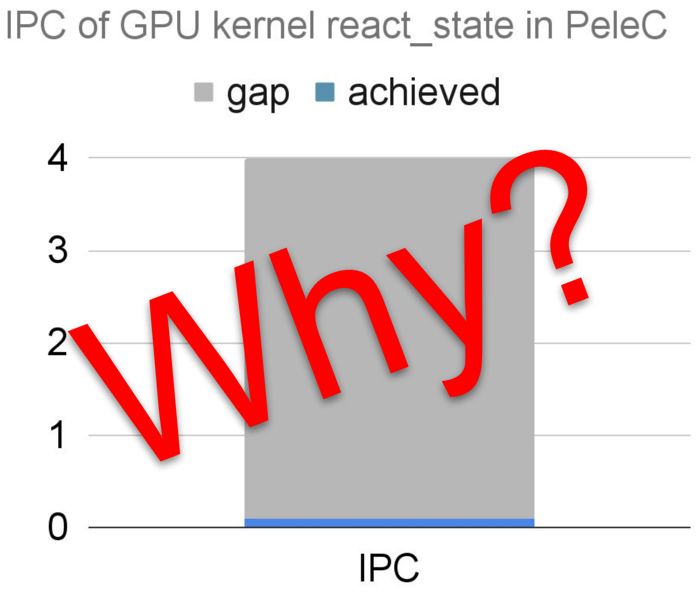
What are Bottlenecks of GPU?


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

1 instruction per cycle (**IPC**) per warp scheduler per SM
=>
Ideal instruction per cycle is **4**

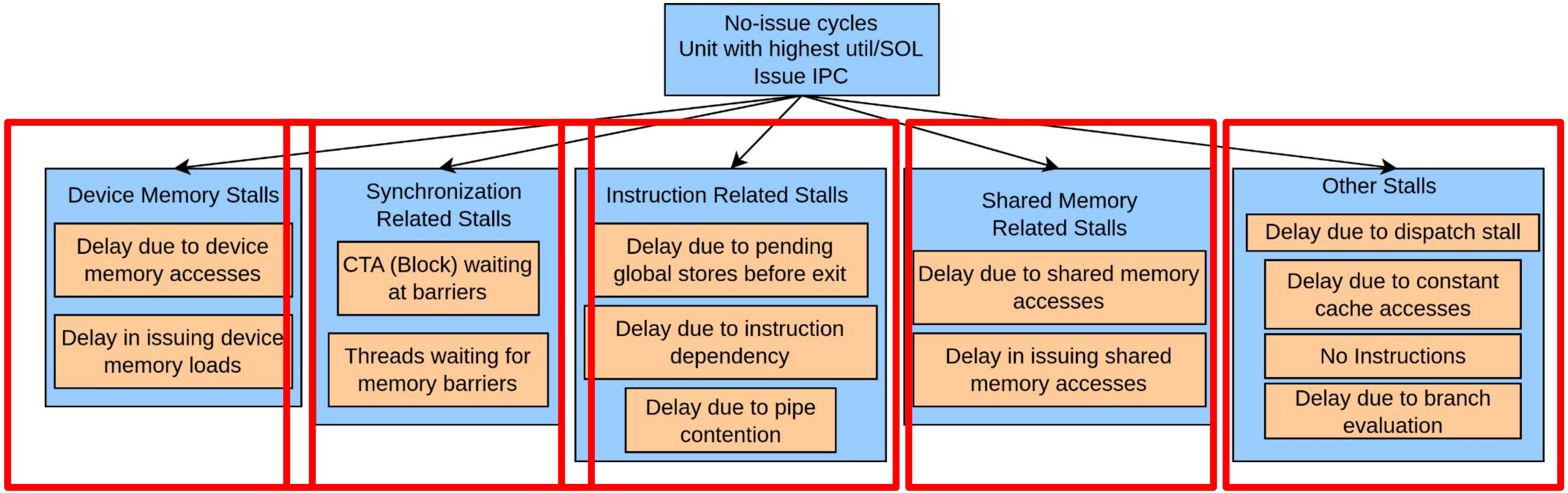
What are Bottlenecks of GPU?


SM
L1 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
L0 Instruction Cache
128KB L1 Data Cache / Shared Memory
Tex

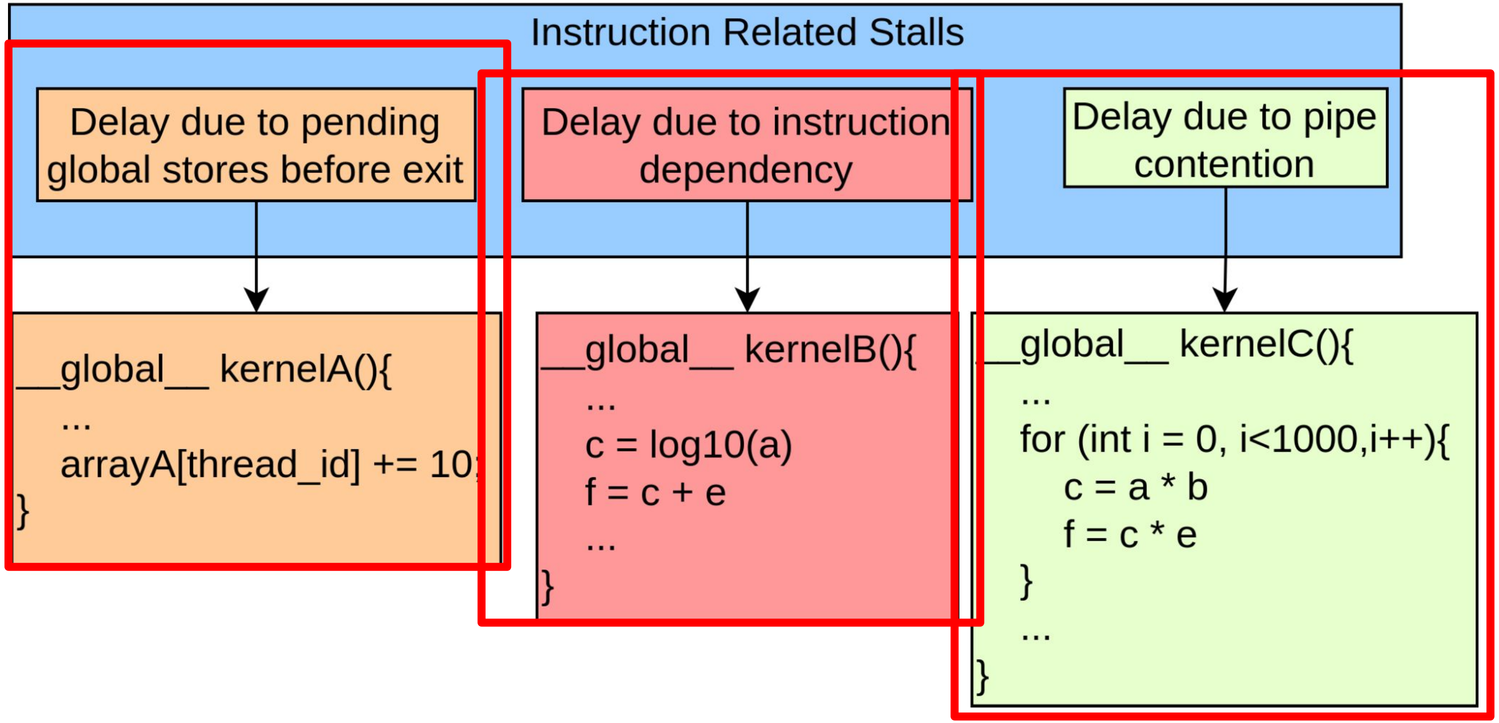


Gap is large between achieved IPC and ideal IPC!

Categories of Stall Reasons


No-issue cycles
Unit with highest util/SOL
Issue IPC
Device Memory Stalls
Delay due to device memory accesses
Delay in issuing device memory loads
Synchronization Related Stalls
CTA (Block) waiting at barriers
Threads waiting for memory barriers
Instruction Related Stalls
Delay due to pending global stores before exit
Delay due to instruction dependency
Delay due to pipe contention
Shared Memory Related Stalls
Delay due to shared memory accesses
Delay in issuing shared memory accesses
Other Stalls
Delay due to dispatch stall
Delay due to constant cache accesses
No Instructions
Delay due to branch evaluation

Instruction Related Stalls

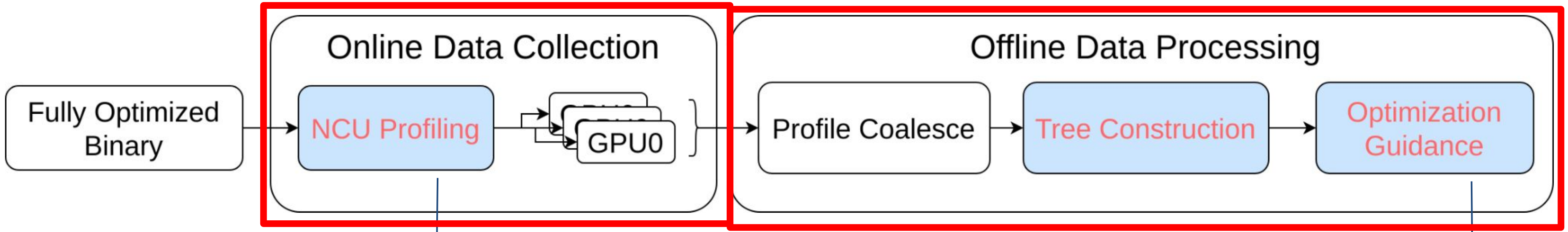

Instruction Related Stalls
Delay due to pending global stores before exit
Delay due to instruction dependency
Delay due to pipe contention

```
__global__ kernelA(){
    ...
    arrayA[thread_id] += 10;
}
```

```
__global__ kernelB(){
    ...
    c = log10(a)
    f = c + e
    ...
}
```

```
__global__ kernelC(){
    ...
    for (int i = 0, i<1000,i++){
        c = a * b
        f = c * e
    }
    ...
}
```

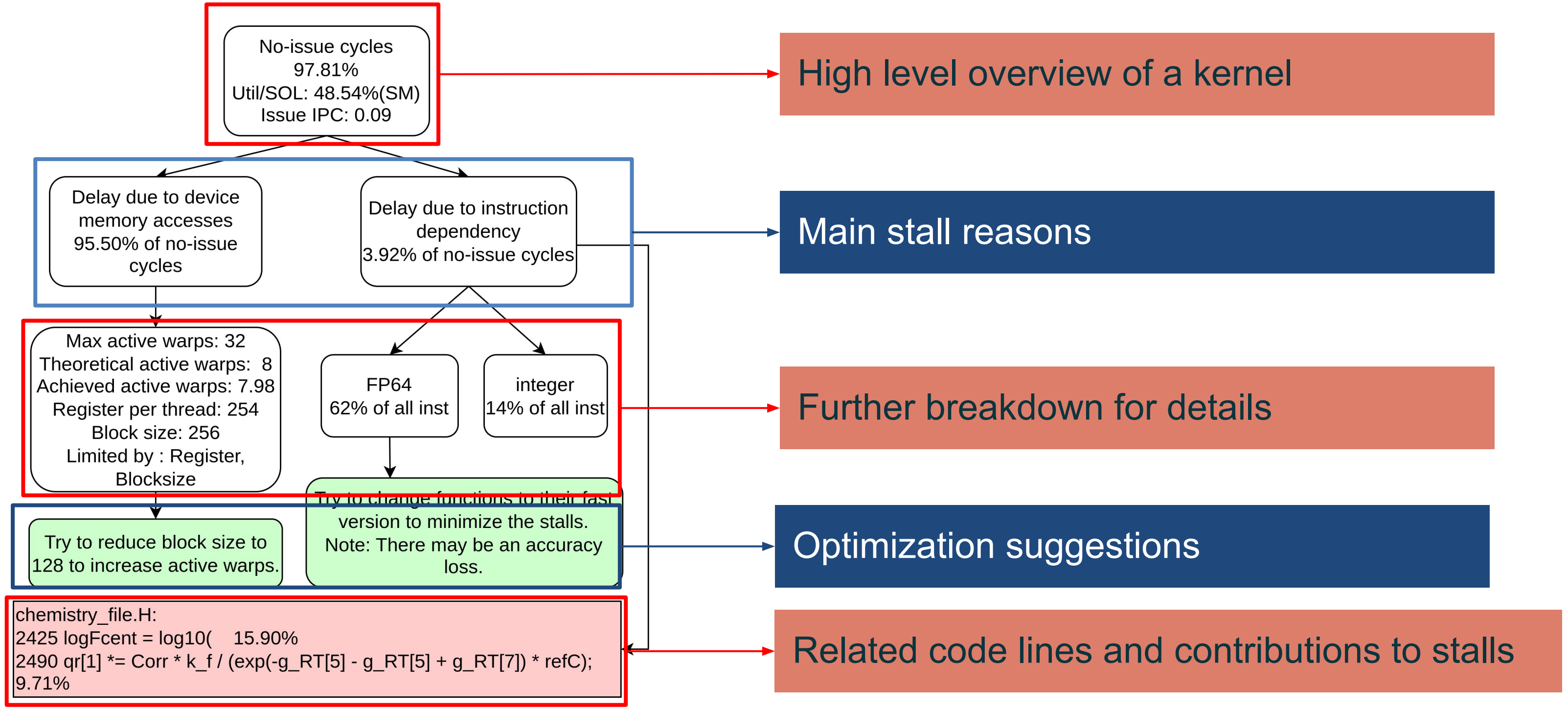
DrGPU Overview



DrGPU utilizes NCU(NVIDIA Nsight Compute) to profile GPU applications with customized GPU hardware counter list

DrGPU gives optimization suggestions based on collected hardware counters.

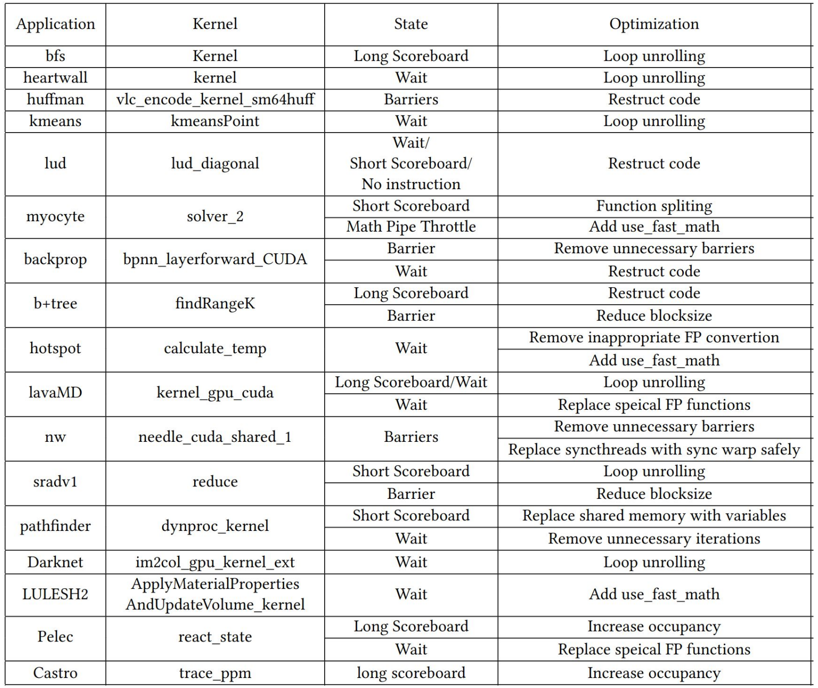
An example of Analysis Trees


No-issue cycles
97.81%
Util/SOL: 48.54%(SM)
Issue IPC: 0.09
Delay due to device memory accesses
95.50% of no-issue cycles
Delay due to instruction dependency
3.92% of no-issue cycles
Max active warps: 32
Theoretical active warps: 8
Achieved active warps: 7.98
Register per thread: 254
Block size: 256
Limited by : Register, Blocksize
FP64
62% of all inst
integer
14% of all inst
Try to reduce block size to 128 to increase active warps.
Try to change functions to their fast version to minimize the stalls.
Note: There may be an accuracy loss.
High level overview of a kernel
Main stall reasons
Further breakdown for details
Optimization suggestions
Related code lines and contributions to stalls

```
chemistry_file.H:
2425 logFcent = log10(    15.90%
2490 qr[1] *= Corr * k_f / (exp(-g_RT[5] - g_RT[5] + g_RT[7]) * refC);
9.71%
```

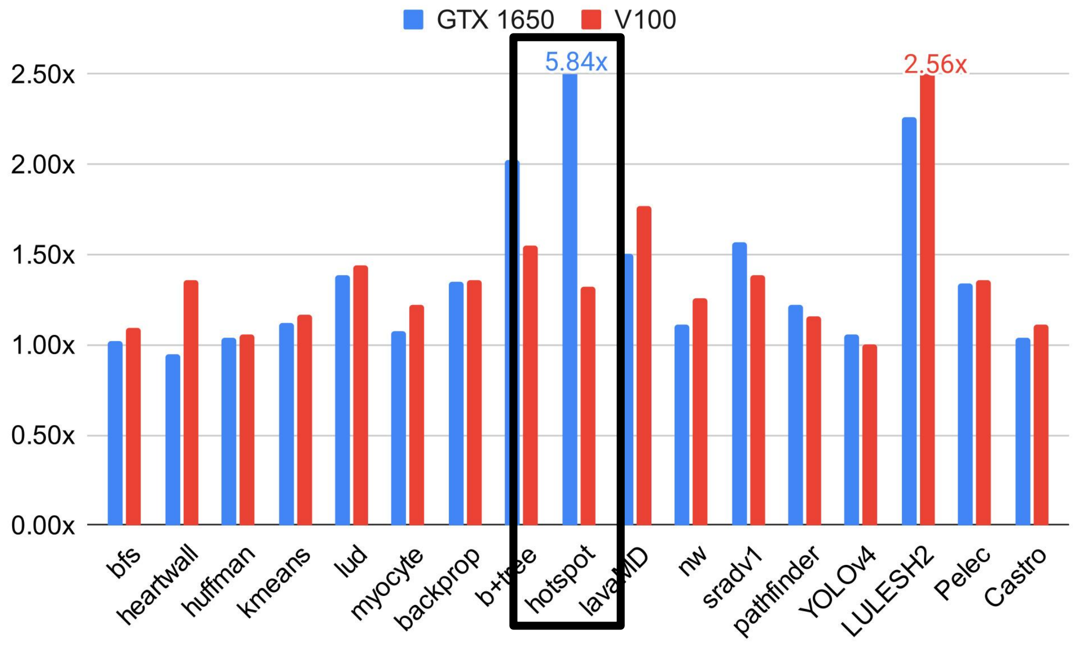
Evaluation Platforms

- GPU
  - V100 16GB
  - GTX 1650 4GB
- Applications
  - Rodinia benchmarks
  - YOLOv4 (Darknet)
  - LULESH2
  - PeleC
  - Castro



Application	Kernel	State	Optimization
bfs	Kernel	Long Scoreboard	Loop unrolling
heartwall	kernel	Wait	Loop unrolling
huffman	vlc_encode_kernel_sm64huff	Barriers	Restruct code
kmeans	kmeansPoint	Wait	Loop unrolling
lud	lud_diagonal	Wait/ Short Scoreboard/ No instruction	Restruct code
myocyte	solver_2	Short Scoreboard	Function spliting
		Math Pipe Throttle	Add use_fast_math
backprop	bpnn_layerforward_CUDA	Barrier	Remove unnecessary barriers
		Wait	Restruct code
b+tree	findRangeK	Long Scoreboard	Restruct code
		Barrier	Reduce blocksize
hotspot	calculate_temp	Wait	Remove inappropriate FP convertion
			Add use_fast_math
lavaMD	kernel_gpu_cuda	Long Scoreboard/Wait	Loop unrolling
		Wait	Replace speical FP functions
nw	needle_cuda_shared_1	Barriers	Remove unnecessary barriers
			Replace syncthreads with sync warp safely
sradv1	reduce	Short Scoreboard	Loop unrolling
		Barrier	Reduce blocksize
pathfinder	dynproc_kernel	Short Scoreboard	Replace shared memory with variables
		Wait	Remove unnecessary iterations
Darknet	im2col_gpu_kernel_ext	Wait	Loop unrolling
LULESH2	ApplyMaterialProperties AndUpdateVolume_kernel	Wait	Add use_fast_math
Pelec	react_state	Long Scoreboard	Increase occupancy
		Wait	Replace speical FP functions
Castro	trace_ppm	long scoreboard	Increase occupancy

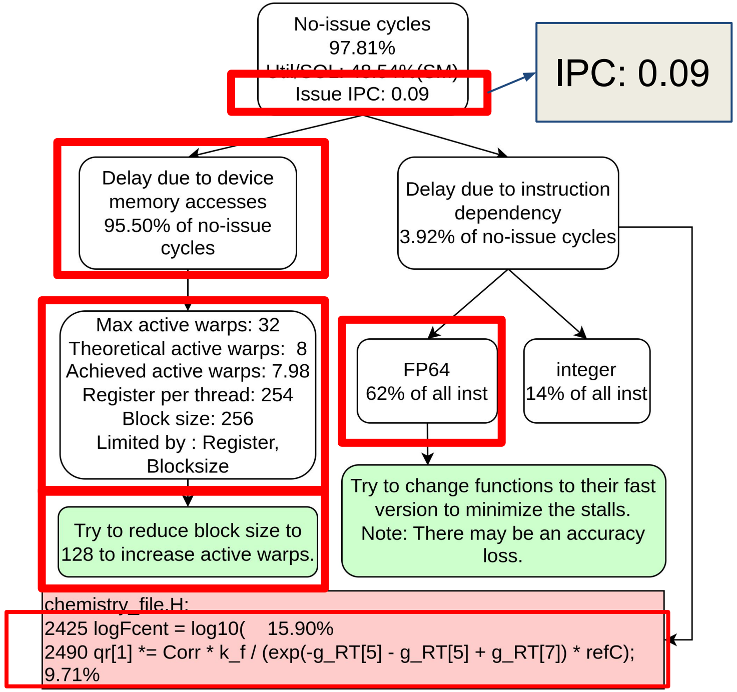
Speedups with Optimization Guided by DrGPU



1.58X on GTX 1650

1.36X on V100

PeleC


No-issue cycles
97.81%
Util/SOL: 48.54%(SM)
Issue IPC: 0.09
IPC: 0.09
Delay due to device memory accesses 95.50% of no-issue cycles
Delay due to instruction dependency 3.92% of no-issue cycles
Max active warps: 32
Theoretical active warps: 8
Achieved active warps: 7.98
Register per thread: 254
Block size: 256
Limited by : Register, Blocksize
FP64 62% of all inst
integer 14% of all inst
Try to reduce block size to 128 to increase active warps.
Try to change functions to their fast version to minimize the stalls. Note: There may be an accuracy loss.

```
chemistry_file.H:
2425 logFcent = log10(    15.90%
2490 qr[1] *= Corr * k_f / (exp(-g_RT[5] - g_RT[5] + g_RT[7]) * refC);
9.71%
```

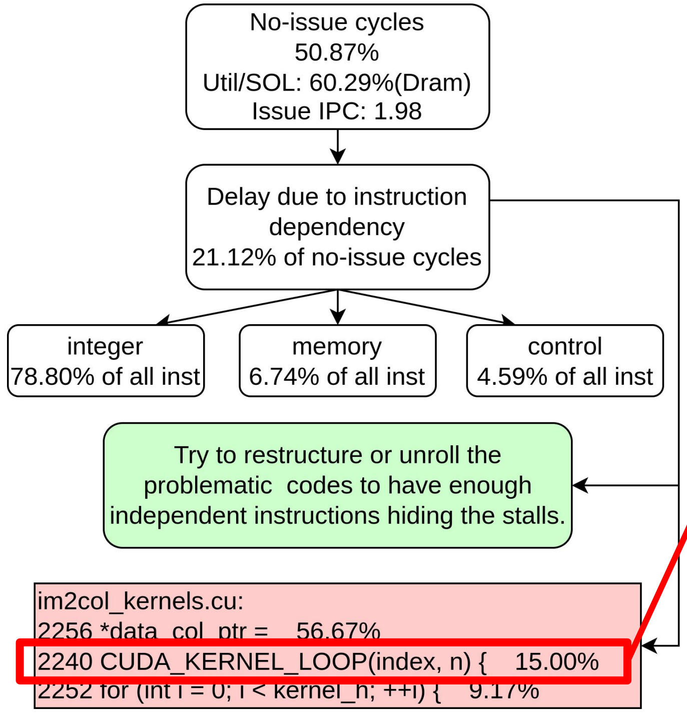
A portion of the analysis tree on GTX 1650

Optimizations

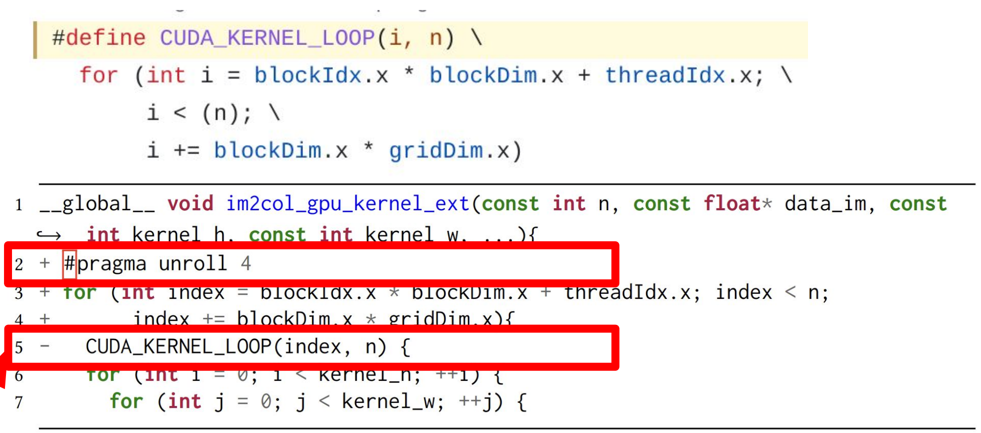
- Set blocksize to 128.
- Replace functions to their faster version. e.g., log10 -> log10f (0.1% precision loss)

1.34X speedup on GTX 1650
1.36X speedup on V100

YOLOv4


No-issue cycles
50.87%
Util/SOL: 60.29%(Dram)
Issue IPC: 1.98
Delay due to instruction dependency
21.12% of no-issue cycles
integer
78.80% of all inst
memory
6.74% of all inst
control
4.59% of all inst
Try to restructure or unroll the problematic codes to have enough independent instructions hiding the stalls.
im2col_kernels.cu:
2256 *data_col_ptr = 56.67%
2240 CUDA_KERNEL_LOOP(index, n) { 15.00%
2252 for (int i = 0; i < kernel_h; ++i) { 9.17%

A portion of the analysis Tree on GTX 1650



```
#define CUDA_KERNEL_LOOP(i, n) \
   for (int i = blockIdx.x * blockDim.x + threadIdx.x; \
          i < (n); \
          i += blockDim.x * gridDim.x)
```

```
  __global__ void im2col_gpu_kernel_ext(const int n, const float* data_im, const
↪   int kernel_h, const int kernel_w, ...){
+ #pragma unroll 4
+ for (int index = blockIdx.x * blockDim.x + threadIdx.x; index < n;
+         index += blockDim.x * gridDim.x){
-   CUDA_KERNEL_LOOP(index, n) {
      for (int i = 0; i < kernel_h; ++i) {
        for (int j = 0; j < kernel_w; ++j) {
```

Optimization

- Loop unrolling

1.06X speedup on GTX 1650

Conclusions

We propose DrGPU, a novel top-down profiler for GPU kernels.

- ➢ DrGPU quantifies stall cycles and decomposes them according to various hardware events for root causes.
- ➢ DrGPU generates performance analysis trees including source code location, root causes, and actionable guidance
- ➢ We optimized a number of applications with the insights provided by DrGPU with nontrivial speedups on both desktop and Summit NVIDIA GPUs
- ➢ Some of optimization suggestions proposed by DrGPU have been integrated to NVIDIA Nsight Compute

Code is available at: https://github.com/FindHao/drgpu

Thanks!