DrGPU: A Top-Down Profiler for GPU

Yueming Hao\textsuperscript{1}, Nikhil Jain\textsuperscript{2}, Rob Van Der Wijngaart\textsuperscript{2}, Nirmal Saxena\textsuperscript{2}, Yuanbo Fan\textsuperscript{3}, Xu Liu\textsuperscript{1}

\textsuperscript{1}North Carolina State University
\textsuperscript{2}NVIDIA Corporation
\textsuperscript{3}Tenstorrent Incorporated
GPUs are Broadly Used for Acceleration
GPU Programming

```c
// 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);
    ...
}
```

1 warp = 32 threads

GPU Kernel

# of GPU threads
Existing GPU Performance Tools

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?

1 instruction per cycle (IPC) per warp scheduler per SM

=>

Ideal instruction per cycle is 4
What are Bottlenecks of GPU?

Gap is large between achieved IPC and ideal IPC!
Instruction Related Stalls

Delay due to pending global stores before exit

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

Delay due to instruction dependency

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

Delay due to pipe contention

```
__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

High level overview of a kernel

Main stall reasons

Further breakdown for details

Optimization suggestions

Related code lines and contributions to stalls

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.

versions to minimize the stalls. Note: There may be an accuracy loss.

chemistry_file.H:
2425 log10( 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 Details

<table>
<thead>
<tr>
<th>Application</th>
<th>Kernel</th>
<th>State</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs</td>
<td>kernel</td>
<td>Long Scoreboard</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>heartwall</td>
<td>kernel</td>
<td>Wait</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>huffman</td>
<td>vlc_encode_kernel_sm64huff</td>
<td>Barriers</td>
<td>Restruct code</td>
</tr>
<tr>
<td>kmeans</td>
<td>kmeansPoint</td>
<td>Wait</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>lud</td>
<td>lud_diagonal</td>
<td>Wait/Short Scoreboard/No instruction</td>
<td>Restruct code</td>
</tr>
<tr>
<td>myosocyte</td>
<td>solver_2</td>
<td>Short Scoreboard</td>
<td>Function splitting</td>
</tr>
<tr>
<td>backprop</td>
<td>bpn_layerforward_CUDA</td>
<td>Barrier</td>
<td>Remove unnecessary barriers</td>
</tr>
<tr>
<td>b+tree</td>
<td>findRangeK</td>
<td>Long Scoreboard</td>
<td>Restruct code</td>
</tr>
<tr>
<td>hotspot</td>
<td>calculate_temp</td>
<td>Wait</td>
<td>Remove inappropiate FP converion</td>
</tr>
<tr>
<td>lavaMD</td>
<td>kernel_gpu_cuda</td>
<td>Long Scoreboard/Wait</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>nw</td>
<td>needle_cuda_shared_1</td>
<td>Wait</td>
<td>Replace special FP functions</td>
</tr>
<tr>
<td>stradv1</td>
<td>reduce</td>
<td>Short Scoreboard</td>
<td>Loop unrolling</td>
</tr>
<tr>
<td>pathfinder</td>
<td>dynproc_kernel</td>
<td>Short Scoreboard</td>
<td>Replace shared memory with variables</td>
</tr>
<tr>
<td>Darknet</td>
<td>im2col_gpu_kernel_ext</td>
<td>Wait</td>
<td>Remove unnecessary iterations</td>
</tr>
<tr>
<td>LULESH2</td>
<td>ApplyMaterialProperties AndUpdateVolume_kernel</td>
<td>Wait</td>
<td>Add use_fast_math</td>
</tr>
<tr>
<td>PeleC</td>
<td>react_state</td>
<td>Long Scoreboard</td>
<td>Increase occupancy</td>
</tr>
<tr>
<td>Castro</td>
<td>trace_ppm</td>
<td>Long scoreboard</td>
<td>Increase occupancy</td>
</tr>
</tbody>
</table>
Speedups with Optimization Guided by DrGPU

1.58X on GTX 1650
1.36X on V100
PeleC

Optimizations

- Set blocksize to 128.
- Replace functions to their faster version. e.g., $\log_{10} \rightarrow \log_{10f}$ (0.1% precision loss)

1.34X speedup on GTX 1650
1.36X speedup on V100
YOLOv4

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!