SIP: Autotuning GPU Native Schedules via Stochastic Instruction Perturbation

- Guoliang He, Eiko Yoneki
- University of Cambridge, UK
BACKGROUND: LLMS

- LLMs are remarked by their substantial computational demands
- To fully utilize the hardware resources, ML practitioner designs customized CUDA kernels, such as Flash Attention [1]
- In this work, we explore the optimization at GPU assembly level

BACKGROUND: CUDA COMPILATION

CUDA kernels compilation pipeline:

C++/CUDA → PTX → SASS → Cubin

- C++/CUDA: the common programming interface; CUDA kernels as a C++ function
- PTX: the lowest officially supported programming interface, usually embedded in CUDA/C++
- SASS: GPU native instructions. GPU architectures-dependent. (SIP optimizes at this level)
- Cubin: executable binary
• Latency Hiding: overlap memory I/O and computation units

• GPU executes static instruction schedule

• **Manually reordering** the instruction schedules is performed by prior works [2]. **Trial-and-error** is proposed to find the optimal sequences [3]

• Observation: large amount of time and effort are needed to optimize one CUDA kernel

MOTIVATIONS

Limitations:
• Manual scheduling is error-prone and requires in-depth CUDA expertise
• Cannot keep up with the development of new operators

Method:
• We aim to apply *automatic optimization* by defining a search space of possible instruction schedules and perform stochastic search
SIP

Search space:

• Original: full permutation of instructions; computationally intractable
• Pruning: consider only *global read and write* memory instructions, e.g. LDG, STG etc

Mutation:

• Randomly select an instruction and reorder an instruction above or below

Feedback signal:

• Assemble the mutated SASS and run on GPUs
Putting together:

• Use simulated annealing to form the control loop

• Simulated annealing is a metaheuristic stochastic optimization method to explore a discrete search space
IMPLEMENTATION

Integrate to OpenAI Triton:
• A MLIR-based compiler to write CUDA kernels
• Pytorch 2’s default backend
• 1 line of code change to use SIP

Workflow: search then look-up

Probabilistic Testing:
• The formal semantics of SASS is closed-source
• Probabilistic testing to evaluate end-to-end equivalency
• The compiler hint ret_ptr allows SIP to generate reference input/output

@triton.jit
def vector_add(x_ptr, out_ptr):
...

@sip.jit (ret_ptr=1)
def vector_add(x_ptr, out_ptr):
...
EVALUATIONS

• CUDA kernels: fused attention (flash-attention) and fused GEMM-leakyReLU
• GPU: NVIDIA A100 80GB
• Software: NVCC 12.2 and Triton v2.1.0
• Profiler: Nsight Compute
**Table 2.** Fused attention on A100. The input data have the format of [1, 4, 16384, 64], representing batch size, number of heads, sequence length, head dimension.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>SIP</th>
<th>Triton</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM Frequency</td>
<td>cycle/ns</td>
<td>1.51</td>
<td>1.49</td>
</tr>
<tr>
<td>SM Frequency</td>
<td>cycle/ns</td>
<td>1.06</td>
<td>1.05</td>
</tr>
<tr>
<td>Memory Throughput</td>
<td>%</td>
<td>33.19</td>
<td>31.38</td>
</tr>
<tr>
<td>DRAM Throughput</td>
<td>%</td>
<td>1.12</td>
<td>1.07</td>
</tr>
<tr>
<td>Duration</td>
<td>ms</td>
<td>1.29</td>
<td>1.37</td>
</tr>
<tr>
<td>L1/TEX Cache</td>
<td>%</td>
<td>44.46</td>
<td>44.69</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Cache</td>
<td>%</td>
<td>16.29</td>
<td>15.41</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute (SM)</td>
<td>%</td>
<td>45.87</td>
<td>43.39</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.** Fused GEMM LeakyReLU on A100. The input data have the format of [512, 512, 2048] (M, N, K).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>SIP</th>
<th>Triton</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM Frequency</td>
<td>cycle/ns</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>SM Frequency</td>
<td>cycle/ns</td>
<td>1.01</td>
<td>1.02</td>
</tr>
<tr>
<td>Memory Throughput</td>
<td>%</td>
<td>45.64</td>
<td>40.52</td>
</tr>
<tr>
<td>DRAM Throughput</td>
<td>%</td>
<td>9.14</td>
<td>8.14</td>
</tr>
<tr>
<td>Duration</td>
<td>μs</td>
<td>23.97</td>
<td>26.91</td>
</tr>
<tr>
<td>L1/TEX Cache</td>
<td>%</td>
<td>38.16</td>
<td>37.94</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Cache</td>
<td>%</td>
<td>45.64</td>
<td>40.52</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute (SM)</td>
<td>%</td>
<td>19.98</td>
<td>17.73</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TRANSFORMATION CORRECTNESS
DISCOVERING BETTER SCHEDULES

Listing 4. Triton

```plaintext
LDGSTS .E.BYPASS.128 [R219+0x4000], desc[UR16][R10.64], P0;
IMAD .WIDE R18, R9, 0x80, R10;
LDGSTS .E.BYPASS.128 [R219+0x4800], desc[UR16][R44.64], P0;
IMAD .WIDE.U32 R16, R222, 0x2, R64;
LDGSTS .E.BYPASS.128 [R219+0x5000], desc[UR16][R46.64], P0;
IMAD .WIDE.U32 R10, R222, 0x2, R60;
LDGSTS .E.BYPASS.128 [R219+0x5800], desc[UR16][R48.64], P0;
MOV R33, c[0x0][0x1b0];
LDGDEPBAR;
```

Listing 5. SIP

```plaintext
IMAD .WIDE R18, R9, 0x80, R10;
LDGSTS .E.BYPASS.128 [R219+0x4000], desc[UR16][R10.64], P0;
LDGSTS .E.BYPASS.128 [R219+0x4800], desc[UR16][R44.64], P0;
IMAD .WIDE.U32 R16, R222, 0x2, R64;
LDGSTS .E.BYPASS.128 [R219+0x5000], desc[UR16][R46.64], P0;
LDGSTS .E.BYPASS.128 [R219+0x5800], desc[UR16][R48.64], P0;
IMAD .WIDE.U32 R10, R222, 0x2, R60;
MOV R33, c[0x0][0x1b0];
LDGDEPBAR;
```
SUMMARY

SIP:
• Automatically optimize GPUs-native instruction schedules

TODOs:
• Apply static analysis to guarantee end-to-end equivalency
• Investigate better search algorithms

Questions? Email: gh512@cam.ac.uk

Thank you!