GoPTX: Fine-grained GPU Kernel Fusion by PTX-level Instruction Flow Weaving

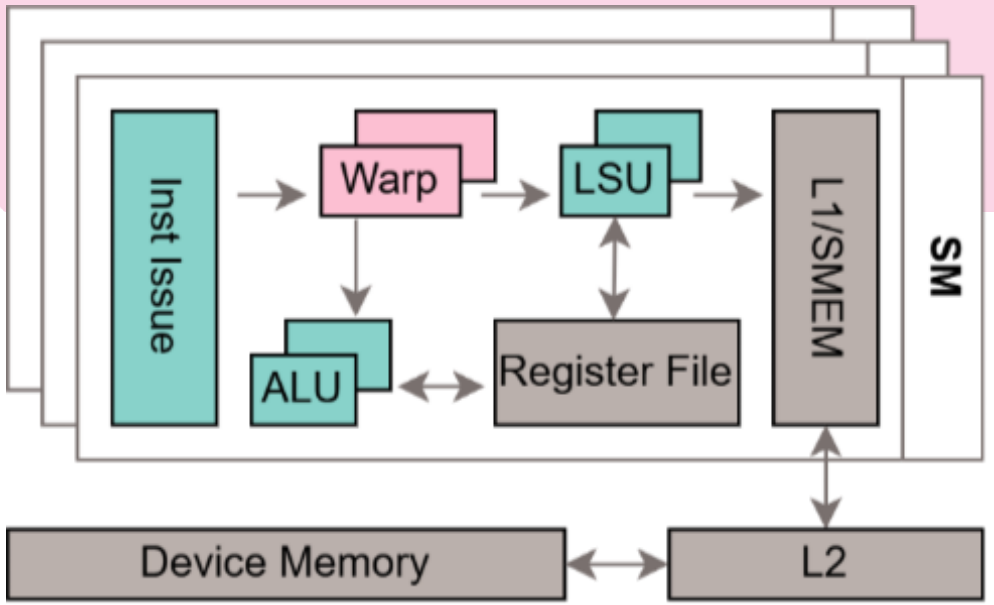
Kan Wu, Zejia Lin, Mengyue Xi, Zhongchun Zheng, Wenxuan Pan,
Xianwei Zhang*, Yutong Lu**
{zhangxw79*, luyutong**}@mail.sysu.edu.cn

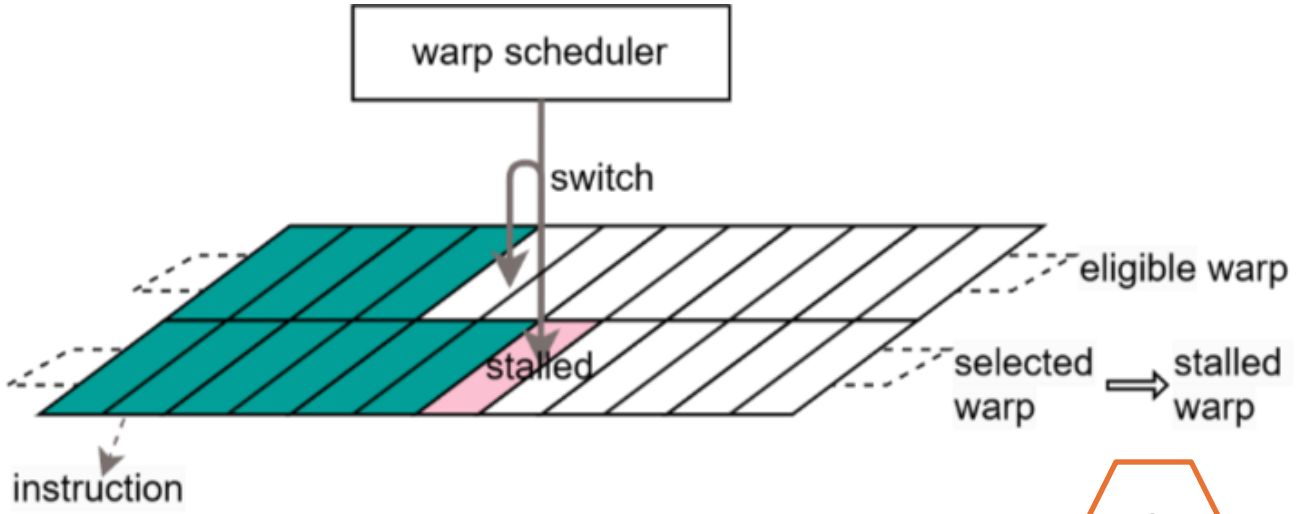
中山大學
SUN YAT-SEN UNIVERSITY

SPONSORED BY CEDA IEEE Council on Electronic Design Automation sigda acm

Background: GPU

- *Heavily utilized in HPC and AI domains*
- *Massive thread execution (Warp)*
- *Warp frequently encounters stalls*
- *Causing a switch to another thread resulting in "bubbles"*





Insight

- ***Mainly scoreboard stalls*** 🤔
  - ***Handle data hazards***
  - ***Enable out-of-order execution***
- ***Data dependencies can't be fully overlapped by instruction reordering***
- ***Fill bubbles with sth. else ...***

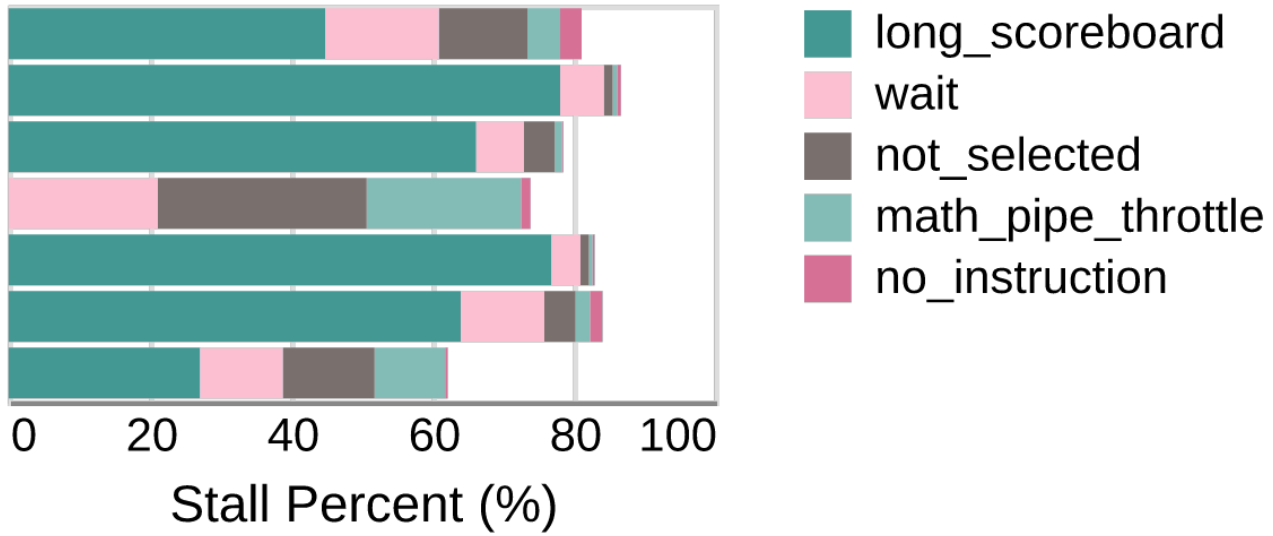


Fig. Stall reasons breakdown of 7 selected kernels.

Motivation

- *... Fill bubbles with another instruction flow* 😋
  - *Weaving instructions to increase data dependency length and thus ILP*
  - *Fine-grained intra-SM resource sharing across kernels to improve utilization*
- *Hardware level solution*
  - *Hyper-threading*
  - *Cost is too high for GPU*
- *Turn to software solution*

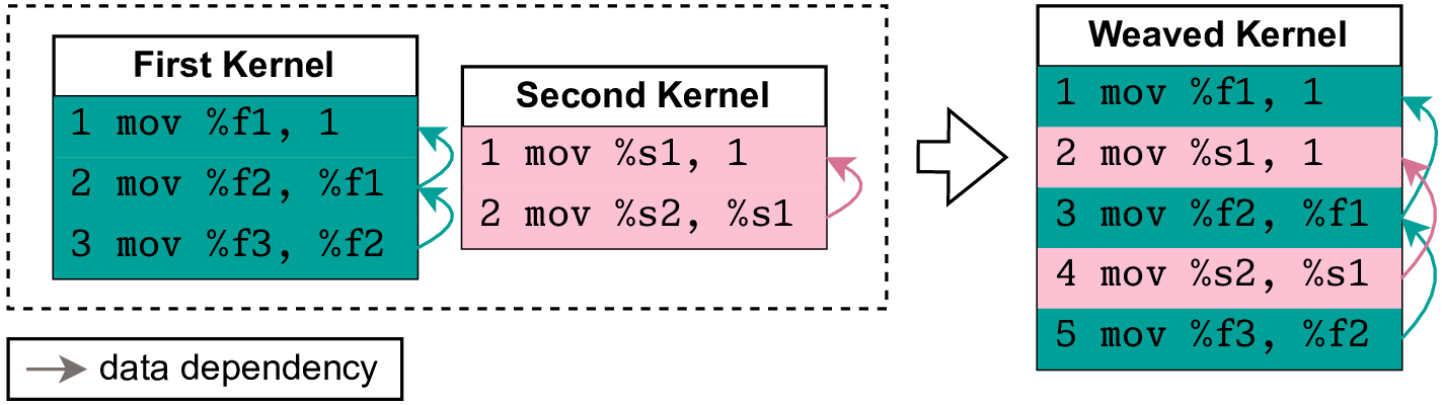
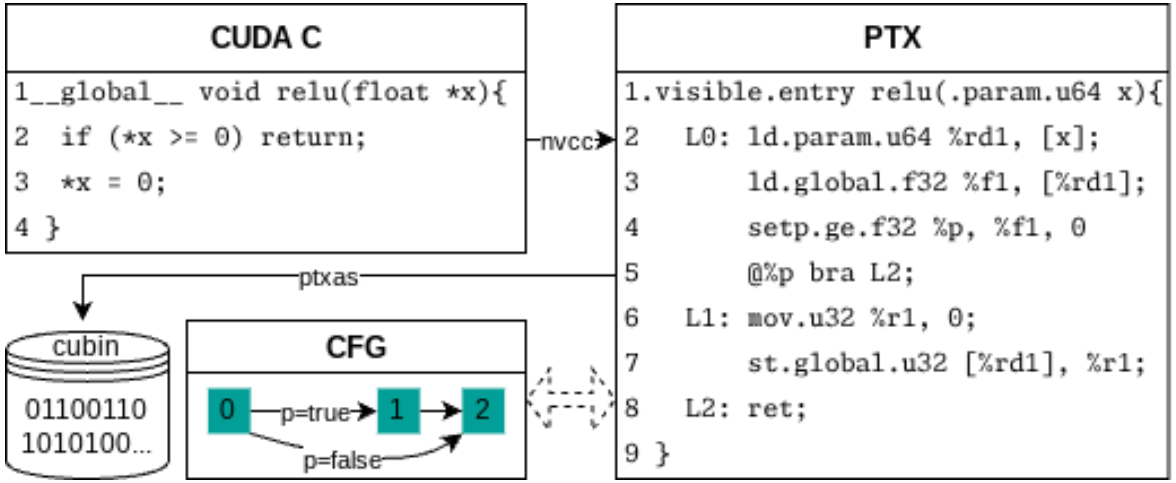


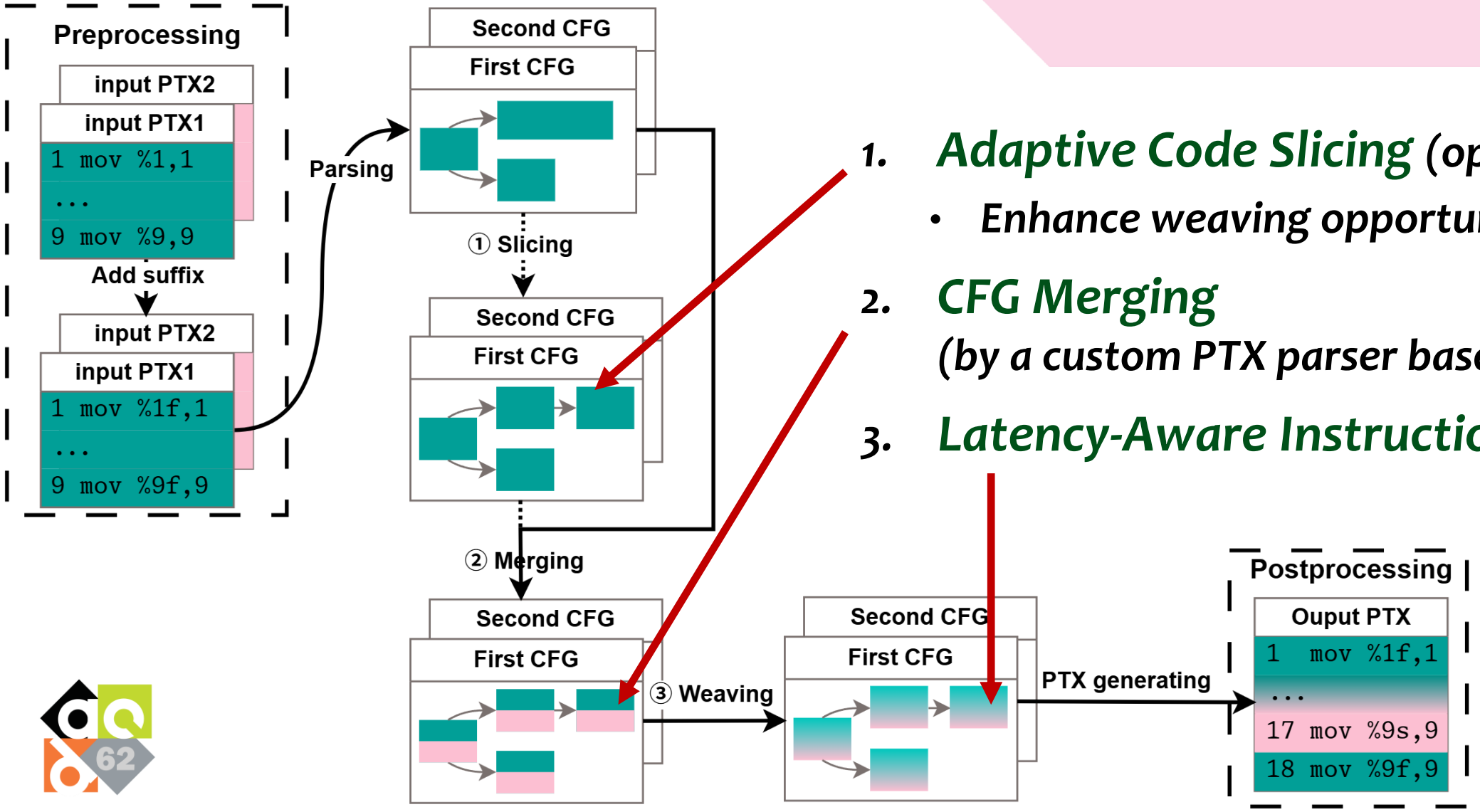
Fig. Increase dependency length by instruction weaving.

Challenge

- **Complex** Instruction Flows
  - *if, else, while, for, break, continue ...*
  - *turn to PTX-level control flow graphs (CFGs)*
- **Deadlocks** in multithreading
- Instruction **Scheduling** is NP-hard
- No official PTX processing tools provided by NVIDIA



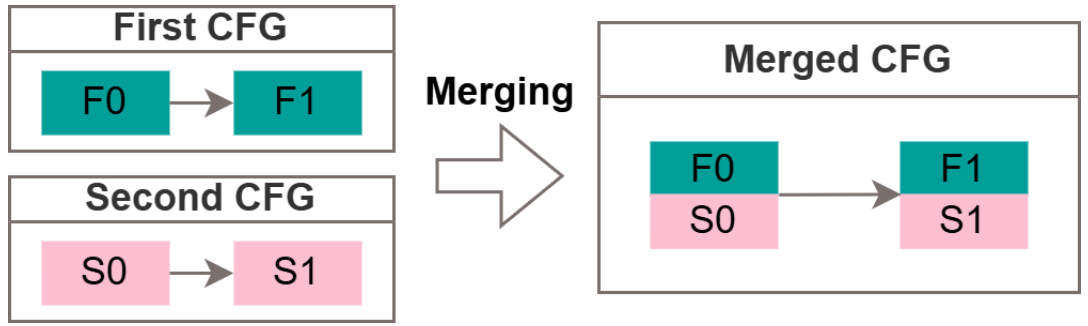
Design of GoPTX

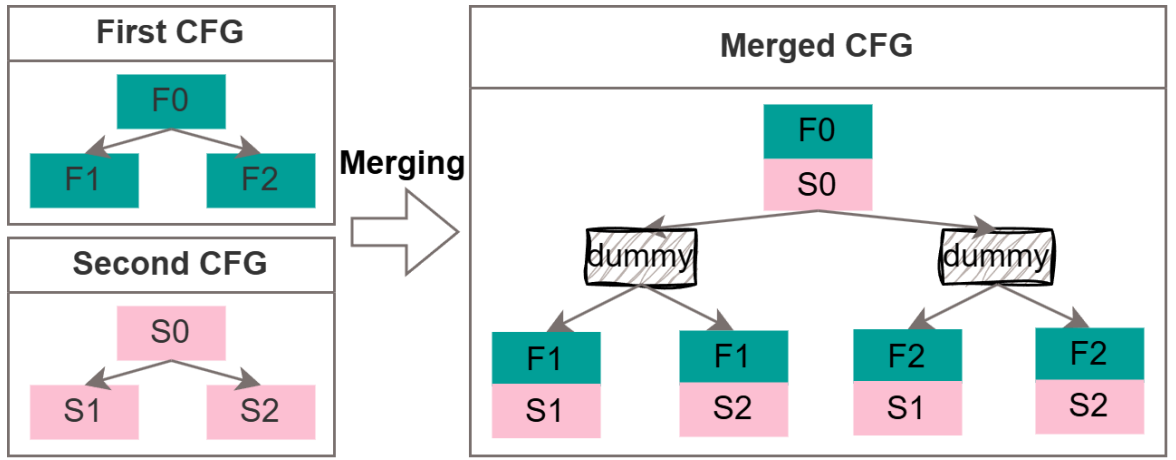


1. **Adaptive Code Slicing** (optional)
   - Enhance weaving opportunity
2. **CFG Merging**
   (by a custom PTX parser based on ANTLR4)
3. **Latency-Aware Instruction Weaving**

Control Flow Graph Merging

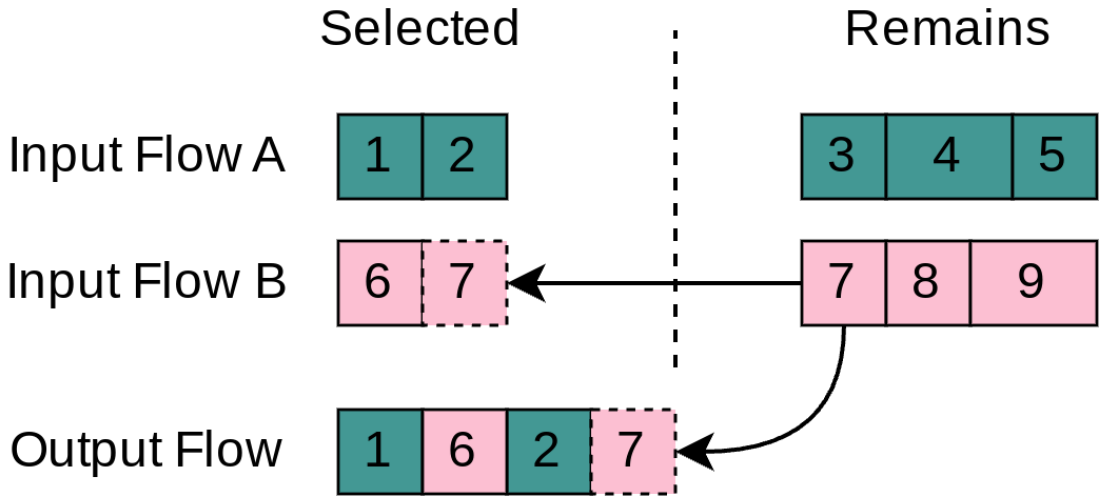
- **Merge kernels into one**
  - Preserve semantics of both kernel
  - Data dependencies, branching behavior, ...
- **Combine basic blocks (BBs)**
  - By traversalling all the possibilities
- **Build CFG**
  - Insert necessary dummy nodes for branch conditions





Latency-aware Instruction Weaving

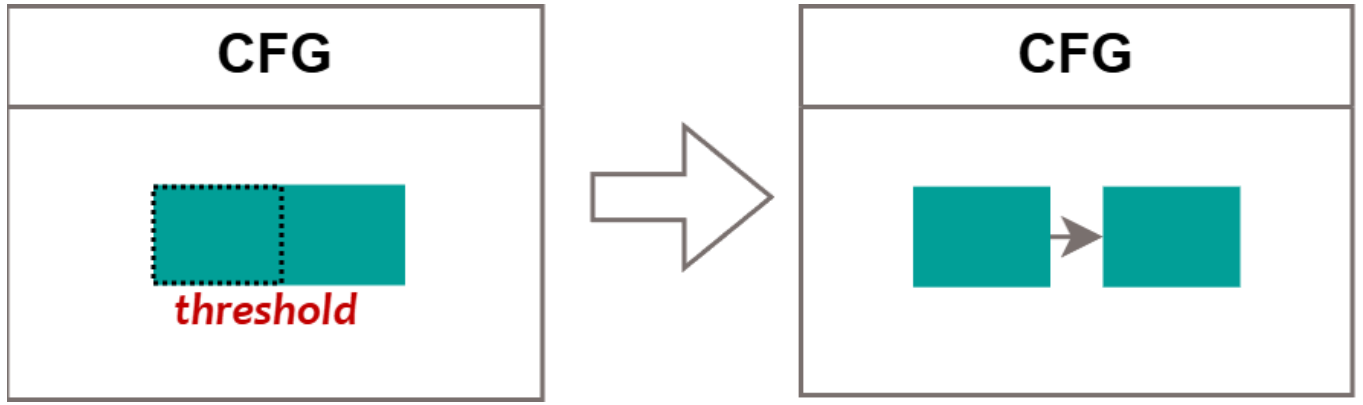
- **Instruction scheduling is NP-Hard** 🤯
  - Generate good heuristic input to ptxas backend
- **Greedily select instructions** 😎
  - From **flow** with the **lower** latency sum
- **Microbenchmark**-driven latency measurement



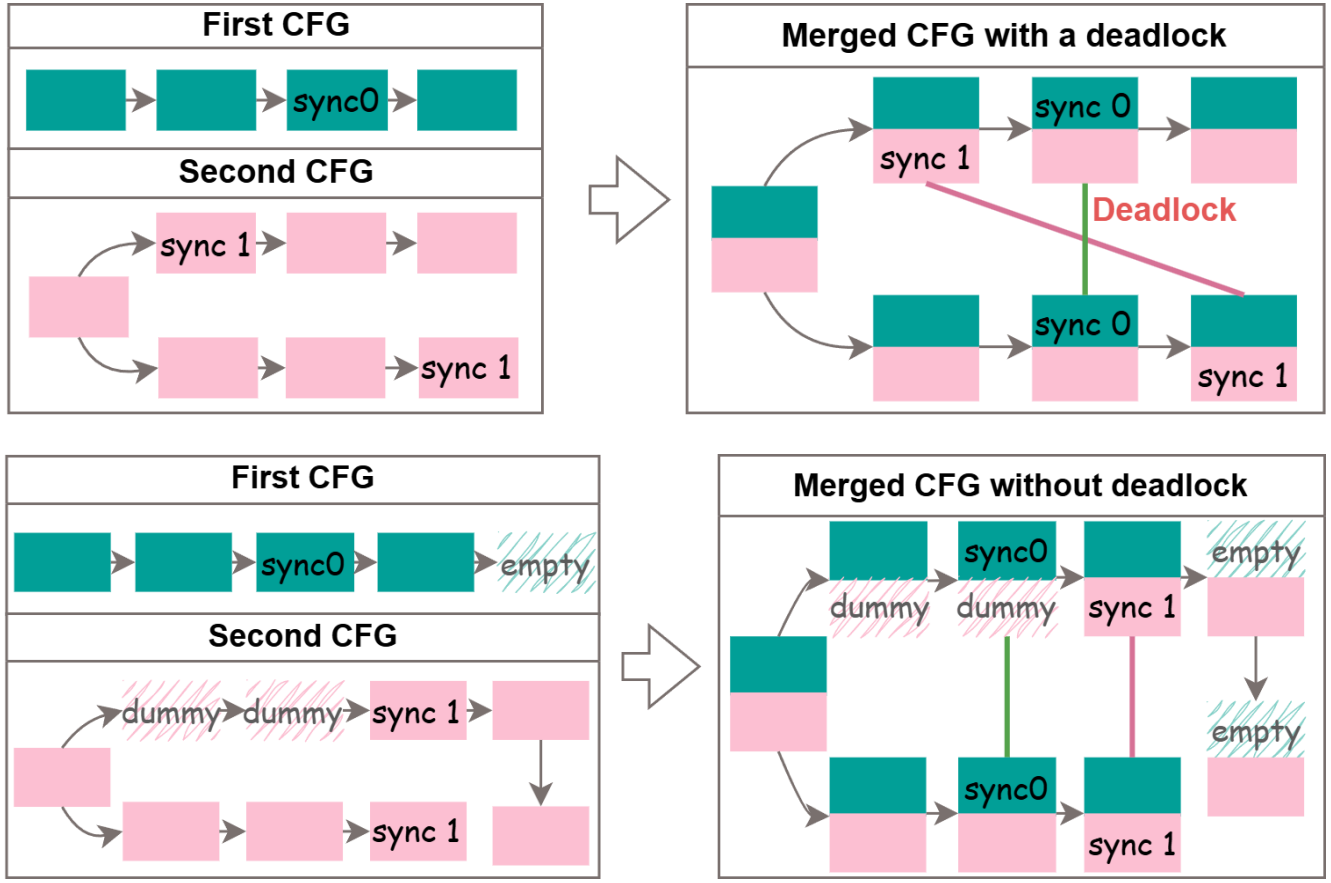
Adaptive Code Slicing

- *Potential scheduling Space in CFG Merging*
  - *Short BB may fail to hide the latency of the long instruction flow*
- *Limit each slice's execution time*

  to the Threshold: the average of execution cycles of all BBs



Deadlock Avoiding



- ***Race condition*** *occurs when* ***syncs*** *exist in both CFGs to merging*
- ***Inserts dummy BBs*** *to* ***delay*** *the first sync until another sync finish* 😋

Evaluation

- **Baseline**: launching two kernels in separate cuStreams
- **VFuse**: vertical fusion that concatenates two kernels sequentially
- **HFuse**: horizontal fusion that schedules into different warps
- NVIDIA A100 & 7x7=49 workloads from 7 kernels of cuda sample, onnx-runtime…

First Kernel

```
1  __global__ __launch_bounds__(32)
2  void first(float x) {
3    x1 = 1; x2 = 2;
4    if (x < 0) {
5      x2 = 1; x1 = 2;
6    }
7  }
```

Second Kernel

```
1  __global__ __launch_bounds__(32)
2  void second(float y) {
3    y1 = 1; y2 = 2;
4    if (y < 0) {
5      y2 = 1; y1 = 2;
6    }
7  }
```

VFuse Kernel

```
1  __global__ __launch_bounds__(32)
2  void vfuse(float x, float y) {
3    x1 = 1; x2 = 2;
4    if (x < 0) {
5      x2 = 1; x1 = 2;
6    }
7    y1 = 1; y2 = 2;
8    if (y < 0) {
9      y2 = 1; y1 = 2;
10   }
11 }
```

HFuse Kernel

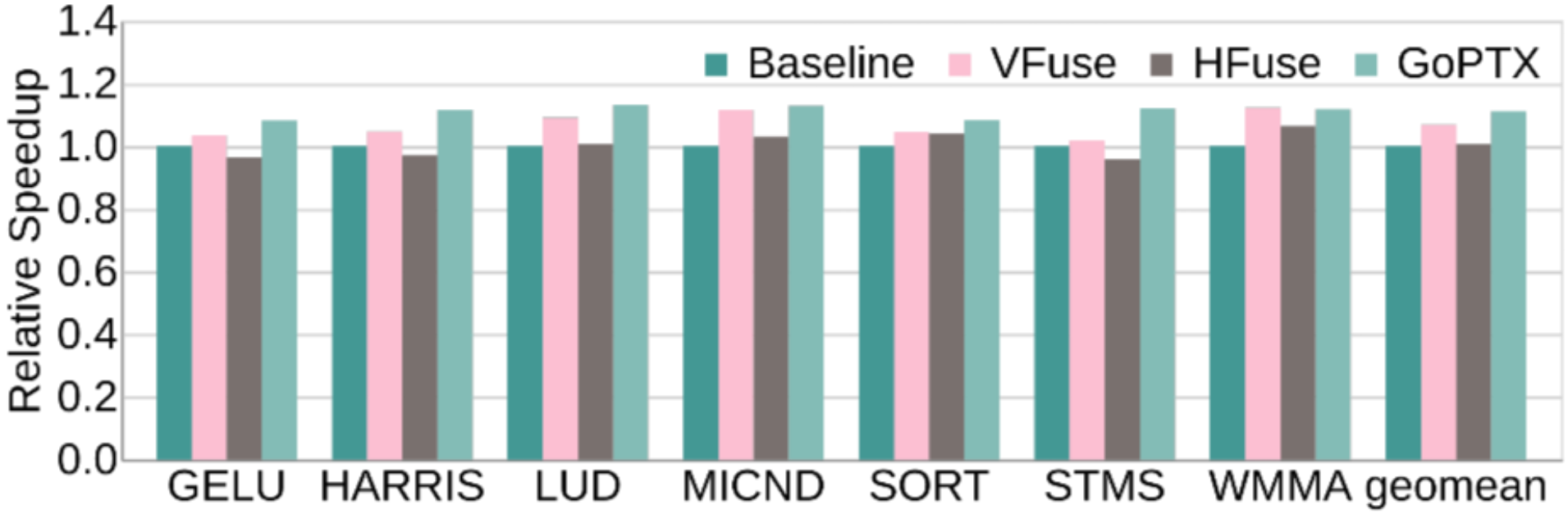
```
1  __global__ __launch_bounds__(64)
2  void hfuse(float x, float y){
3    if (threadIdx.x < 32) {
4      x1 = 1; x2 = 2;
5      if (x < 0) {
6        x2 = 1; x1 = 2;
7      }
8    } else {
9      y1 = 1; y2 = 2;
10     if (y < 0) {
11       y2 = 1; y1 = 2;
12     }
13   }
14 }
```

Desire Kernel

```
1  __global__ __launch_bounds__(32)
2  void desire(float x, float y) {
3    x1 = 1; y1 = 1;
4    x2 = 2; y2 = 2;
5    if (x < 0 && y < 0) {
6      x2 = 1; y2 = 1;
7      x1 = 2; y1 = 2;
8    } else if (x < 0) {
9      x2 = 1; x1 = 2;
10   } else if (y < 0) {
11     y2 = 1; y1 = 2;
12   }
13 }
```

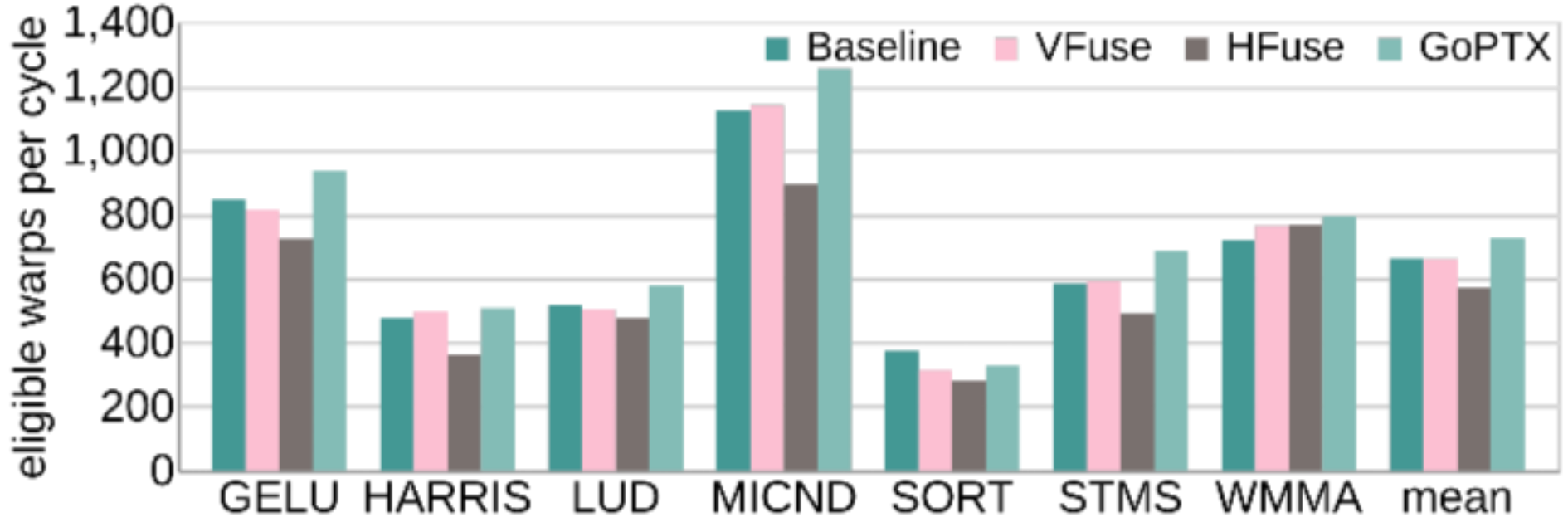
Speedup

- **GoPTX** *11.2% geomean speedup with a maximum of 23%*
- **VFuse** *6.4%, gains from compute-intensive workloads but less than GoPTX in other benchmarks*
- **HFuse** *1.0%, extensive requires profiling to determine kernel combine ratio*



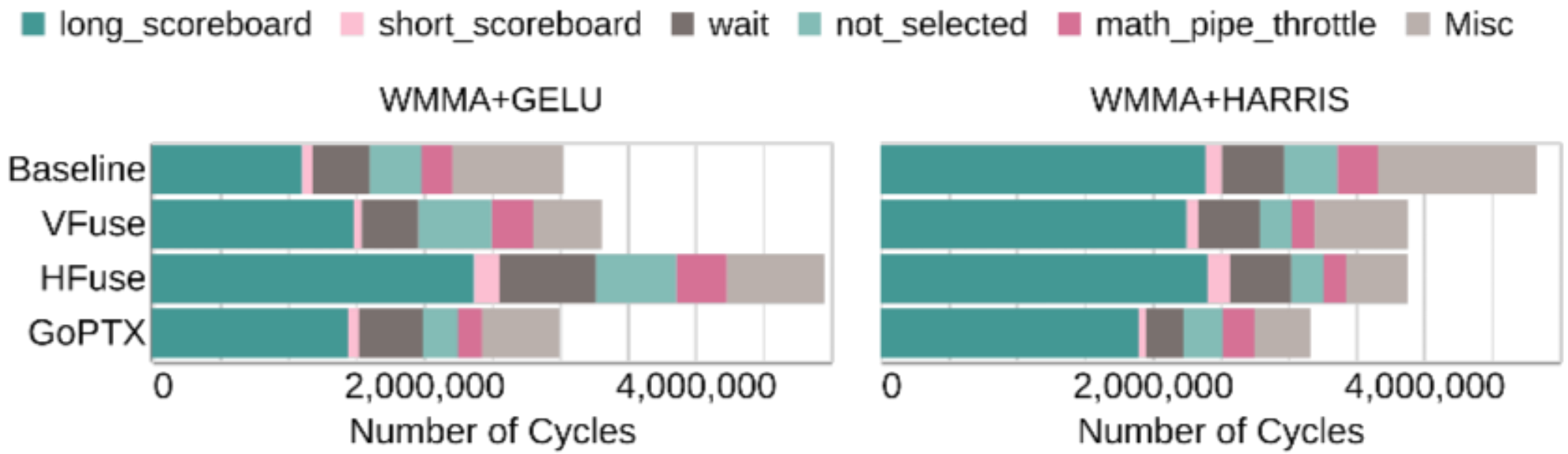
ILP as measured by eligible warps per cycle

- GoPTX increases EWPC by geomean of 5.5% and max 50%
- VFuse exhibits minimal changes due to sequential concatenation
- HFuse reduces EWPC by 14.4% in exchange for TLP improvement
- GoPTX improves hardware utilization (More in the paper 👉)



Case Study of stalls

- *Chose WMMA+GELU and WMMA+HARRIS*
  - *WMMA+GELU(both fp-16 bounded): speedup -2%, EWPC -35%, scoreboard +34%*
  - *WMMA+HARRIS: speedup +20%, EWPC +38% , scoreboard -20.5%*
- *GoPTX may exacerbate stalls (still less than VFuse and HFuse) in cases where kernels race for the same hardware resources*



Performance Breakdown

- **Merging only:** 9.7%
- **Merging+Slicing:** 9.1%, but more overheads occur
- **Merging+Weaving:** 10.0%, less opportunities to weave
- **GoPTX:** 11.2%

For more details, please refer to the paper.

Summary

- GPUs suffer from scoreboard warp stalls
- GoPTX fine-grained fuses concurrent kernels
  1. PTX-level CFG Merging to preserve the semantics of input kernels
  2. Latency-Aware Instruction Weaving to increase data dependency length
  3. Adaptive BLock Slicing to enhance weaving opportunity
- Experimental demonstrate that GoPTX effectively improves performance with higher ILP and utilization

Thank you!
Questions?

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation sigda acm