DICE: Enabling Efficient SIMT Execution with Coarse-Grained Reconfigurable Arrays (CGRAs)

STUDENTS: Jiayi Wang, Ang Da Lu
Pn Computer Engineering Lab (PNCEL)

PNCEL

Motivation

GPGPU streaming multiprocessors' (SMs) inefficiencies:

- Frequent register file (RF) accesses
- Complex scheduling/control logics
- Performance loss due to control divergence

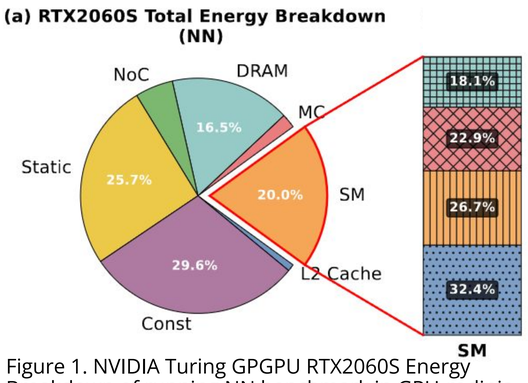


Figure 1. NVIDIA Turing GPGPU RTX2060S Energy Breakdown of running NN benchmark in GPU-rodinia.

DICE replaces the SMs' SIMD backend with CGRAs

- **Less** register file accesses thanks to direct communication between processing elements (PEs)
- **Lower** control overhead thanks to static configuration instead of instruction fetch/decode
- Control divergence handled with **selective dispatch**

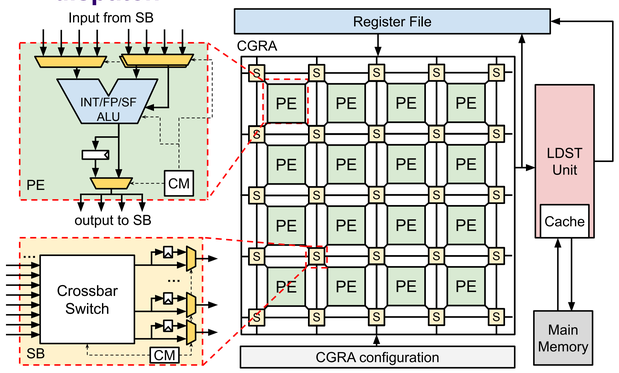


Figure 2. CGRA architecture in DICE.

CGRA vs. SIMD

CGRA = coarse-grained reconfigurable array
SIMD = single-instruction, multiple-data

RF accesses per thread:
- **SIMD**: 12 reads+7 writes
- **CGRA**: 4 reads

CUDA

```
E[tid] = A[tid] * B[tid]
       + C[tid] * D[tid];
```

Assembly

```
LD %r0, [&A[tid]]; //ld0
LD %r1, [&B[tid]]; //ld1
LD %r2, [&C[tid]]; //ld2
LD %r3, [&D[tid]]; //ld3
MUL %r4, %r0, %r1; //mul0
MUL %r5, %r2, %r3; // mul1
ADD %r6, %r4, %r5; //add
ST %r6, [&E[tid]]; //st
```

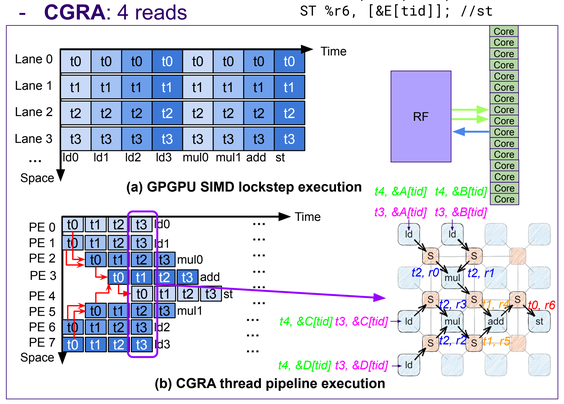


Figure 3. GPGPU SIMD execution vs. CGRA thread pipeline execution

Challenges

- How to support long programs?
  - ➔ **Partition** programs into **pipeline-able** blocks *(p-graph)*
- How to hide memory latency and handle control flow?
  - ➔ **Separate** memory load-to-use to hide memory latency
  - ➔ Cut branches or convert to **predicated** execution

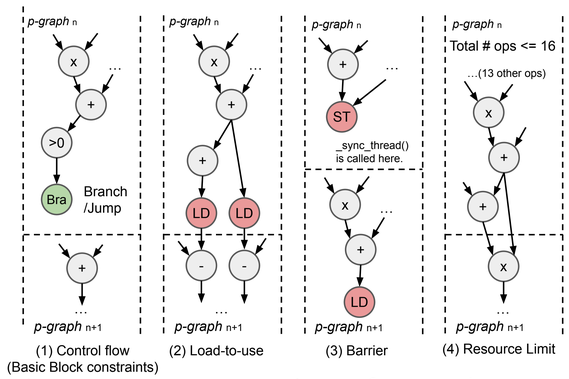


Figure 4. Four constraints of p-graph partitioning

Execution Flow & Architecture

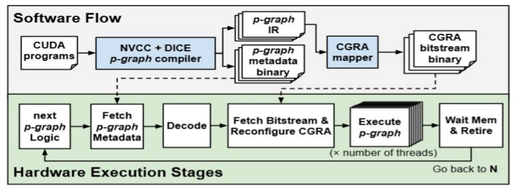


Figure 5. Software flow and hardware execution stages of DICE

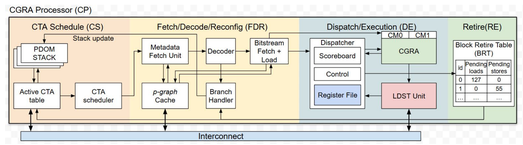


Figure 6. CGRA core Architecture

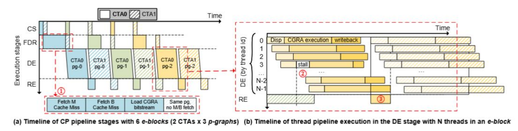


Figure 7. CGRA Processor (CP) pipeline execution example (M: metadata, B: CGRA bitstream, pg: p-graph, Disp: dispatch).

Optimizations

Unrolling small p-graphs

PE utilization increase from 3 to 12 with unrolling = 4.

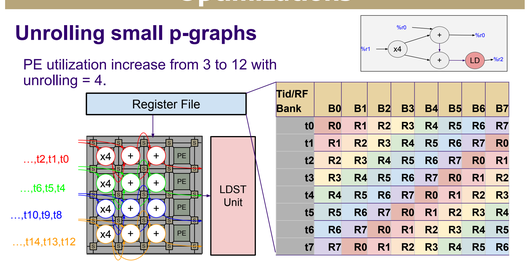


Figure 8. Thread Unrolling and swizzled register file

Temporal memory request coalescing

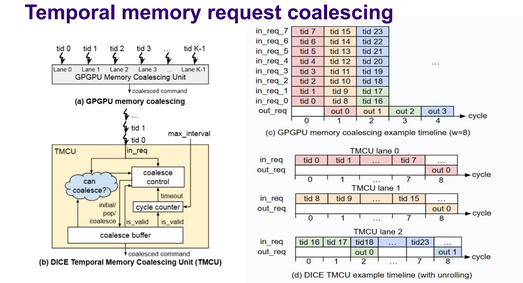


Figure 9. Temporal Memory Coalescing Unit (TMCU)

Results

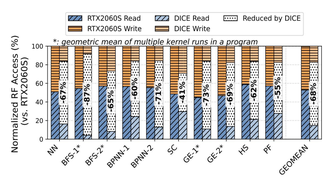


Figure 10. Normalized RF accesses (%)

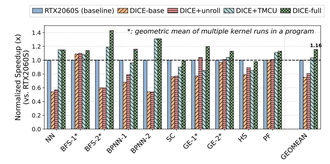
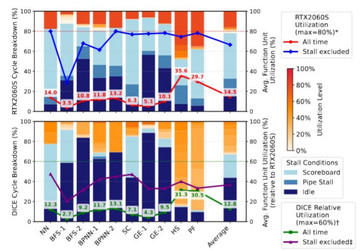


Figure 11. Normalized Speedup



*Maximum GPU utilization is 80% due to dispatcher constraints: each RTX2060S SM contains 160 functional units (64 INT, 64 FP, 16 LD/ST, 16 SFU) but can only dispatch up to 128 threads per cycle.
†DICE relative utilization is capped at 60%: each CGRA cluster has 96 functional units (4 CPs × 24 functional units) compared to RTX2060S's 160 units per SM.
‡Also represents the average *p-graph* size including memory operations.

Figure 12. Cycle breakdown and average function unit utilization

DICE achieves **better performance** due to:
- Reduced data movement instructions (e.g., mov in PTX)
- Selective dispatch when control flow diverges (branch)
- Long memory latency hidden by pipelined thread execution

DICE's lower spatial utilization is offset by significantly higher temporal utilization

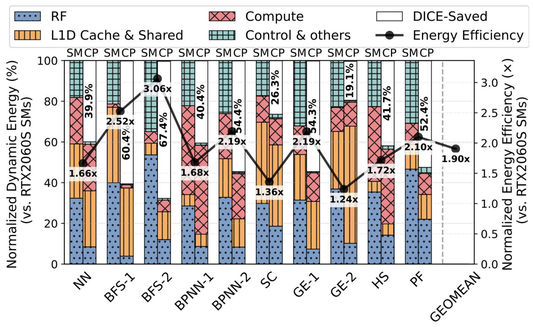


Figure 13. Normalized Energy Consumption (DICE CP vs. RTX2060S SM)

DICE achieves **higher energy efficiency** due to:
- Less register file accesses
- Scheduling by CTA (thread blocks) instead of warps

Summary

With equivalent compute units and memory, Compared to NVIDIA RTX2060S, DICE achieves:
- **1.16x** speedup
- **1.90x** dynamic energy efficiency
- **42%** dynamic power reduction

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

ADVISOR: Prof. Ang Li