GPGPU Performance Modeling using Microbenchmarks

Hyoungjoo Kim (kimhj1473@snu.ac.kr), Professor: Jangwoo Kim, TA: Joonsung Kim

Introduction

- ✓ GPU internal structure is important
  - ➢ Architecture research
  - ➢ Application performance
  - ➢ Simulator accuracy
- ✓ Understanding such detail is difficult
  - ➢ Manufacturer secret
  - ➢ Complexity
  - ➢ Rapidly changing
- ➢ **Systematic, Generic GPU Reverse-Engineering Tool?**

Background

GPU Programming Model

- ○ Kernel: GPU code
  - B threads in a block
  - G blocks in a grid
- ○ Warp: SIMT unit
  - w threads in a warp
- ○ MultiProcessor (Indep.)

Here, we define

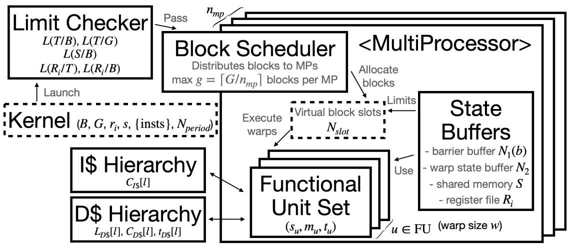
$b = \lceil B/w \rceil,\ g = \lceil G/n_{mp} \rceil$

Related Works

vs. Microbenchmark works
- ➢ Also focused on HW resources
- ➢ Portable among Nvidia, AMD
- ➢ Warp scheduling policy

vs. Modeling works
- ➢ Parametrized
- ➢ Models HW, not workload
- ➢ Functional unit formula

GPGPU Modeling



- ○ **Limit Checker**: Checks kernel size

  $B \le L_{TB}, BG \le L_{TG}, s \le L_{SB}, r_i \le L_{R_iT}, br_i \le L_{R_iB}$
- ○ **Block Scheduler**: Allocate blocks

  $T(g,b) = \left\lfloor \frac{g}{N_{slot}} \right\rfloor fu(bN_{slot}) + fu\big(b(g\%N_{slot})\big)$
- ○ **State Buffers**: Warp slots, RF, sharedM

  $N_{slot} = \min\left(N_1, \left\lfloor \frac{N_2}{b} \right\rfloor, \left\lfloor \frac{S}{\lceil s, s_{min} \rceil} \right\rfloor, \left\lfloor \frac{R_i}{\lceil r_i, r_{i,min} \rceil b} \right\rfloor\right)$
- ○ **Functional Units**: Alloc & exec warps

  $fu(c) = \max\left(1, \left\{ \frac{\Sigma t_u}{m_u P} \left\lceil \frac{c}{s_u} \right\rceil \middle| u \in FU \right\}\right)$
- ○ **Cache Hierarchy**

Microbenchmarks

- ○ **Limit Checker**

  Try & Find Max
- ○ **Functional Units**

```
measure_fu(G, B):
  thread_barrier()
  start = clock()
  #unroll rep(N):
    asm(instruction)
  thread_barrier()
  return clock() - start
```

(G,B) = (1,1): Latency
(1,w..bw): Hardware Width
(1,n), inst=clock(): Sched. Policy

- ○ **State Buffers**

```
measure_sb(G, B, *sync):
  if(tid=0) atomicAdd(sync, 1)
  while(*sync < G)
    check_if_timeout()
  return
  use_resources(s, r_i)
```

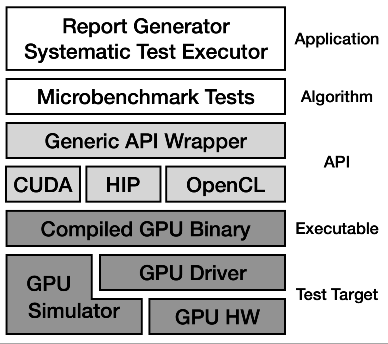
This deadlocks if $\frac{G}{n_{mp}} > N_{slot}$

(G,B) = (1..,c): Number of MP
(1..,1..): Warp state buffer size
(1..,1..), s=1..: SharedM size
(1..,1..), r=1..: RF size

- ○ **Cache Hierarchy**

  Fine-grained p-chase, portable

Implementation



Verification Method

- ○ **Tested environments**

  (A) gpgpusim (Turing, PTX, CUDA)
  (B) gem5-apu (gfx803, GCN3, HIP)
  (C) AMD V520 (gfx1011, RDNA1, HIP)
- ○ **Parameter value comparison**

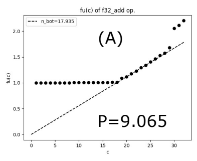
  Simulators: Config values and source code
  Hardware: Official spec, if available

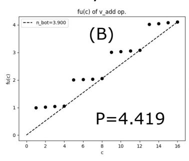
Verification Result

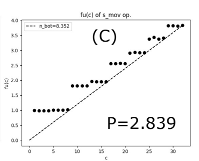
- ○ **Simple discrete parameters**

  Limits, number of MPs, cache linesize, …
  Measured accurately
- ○ **Functional unit width**

  Match with the model, little error

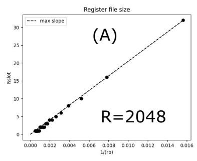


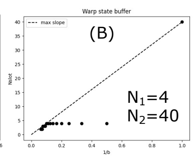


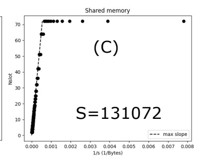


- ○ **State buffer size**

  Match with the model, some differs in (C)

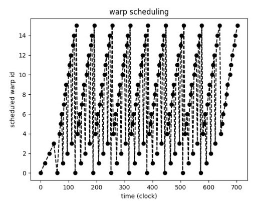


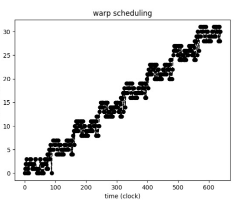




Case Studies

Warp scheduling policy

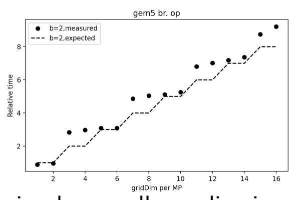




(gem5 Oldest First) (AMD V520)

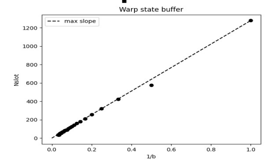
- ➢ gem5 can't simulate the sched policy of real AMD hardware, which is not RR
- ➢ Right graph's shape is similar to gpgpusim Greedy Than Oldest policy

Block allocation inefficiency



- ➢ For simple reg alloc policy in gem5, b≥2 kernel execution time at g=4n+3,4 is larger than expected
- ➢ Due to the gem5 code nature
- ➢ Should use dynamic policy for accurate simulation

RDNA warp state buffer



- ➢ Differ slightly from model
- ➢ Total warp slot number is not multiple of n_{mp}
  - ➢ may be asymmetric?
- ➢ gem5 needs other thread slot model to support RDNA

Conclusion

- ✓ GPU Model
  - ✓ Generic
  - ✓ Parametrized
- ✓ Microbenchmarks
- ✓ Implemented
  - ✓ One-Click Runnable
- ✓ Verified
  - ✓ Simulator & HW
- ✓ Found gem5 faults
- – Anomalies in FU data
- + Can go deeper
- + RDNA support
- + Intel, ARM GPUs?