Ming Hsieh Department of Electrical Engineering

Lightweight Error Detection for GPGPU

USC Viterbi School of Engineering

Hyeran Jeon and Murali Annavaram

SCIP Research Group

Introduction

Why error detection is not used in GPGPU?

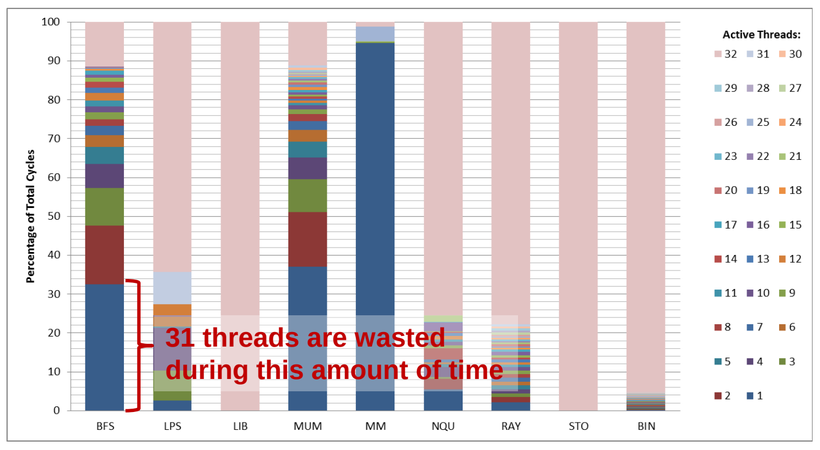
- For now, only memory components (RF,Cache,DRAM) are error protected by ECC
- GPU, the ancestor of GPGPU doesn't care about a few pixel errors that are not perceivable by human eyes

But GP(General Purpose)GPU?

- General Purpose applications(i.e. Scientific/Banking calculations)
  → **Accuracy matters**
- Hundreds of processing elements(cores) within a chip
  → **Likely to have H/W defects**

Goal : Building a Lightweight Error detection Method For GPGPU

Motivation



Not all threads/cores are needed all the time due to

- Control Flow Divergence
- Low parallelization of application itself

Can we use these Idle cores for another purpose?

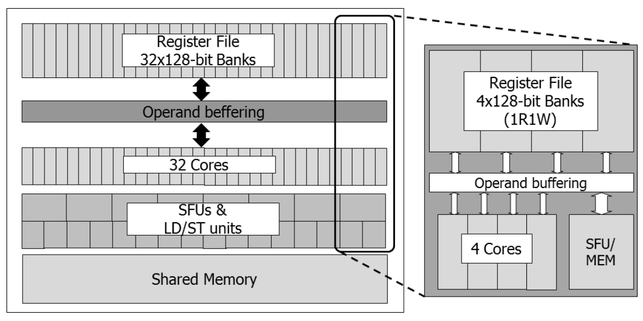
Background

GPGPU Architecture & SIMT execution

A batch of threads(***WARP***) run parallel in lock-step way

- threads in a warp share a PC while accessing individual register files

Streaming Multiprocessors in GPGPU support SIMT execution



Streaming Multiprocessor and SIMT cluster

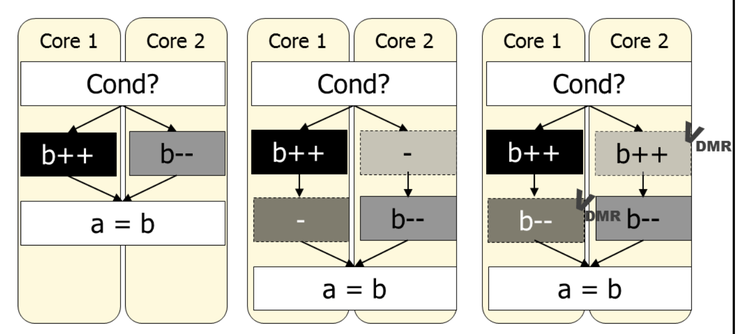
- Each of multiple cores and register banks supports a SIMT lane
- 4 128-bit wide register banks can feed 4 cores effectively for 2 or 3R and 1W (i.e. MULADD) instructions

2-way Error Checking Method

Intra-Warp Checking

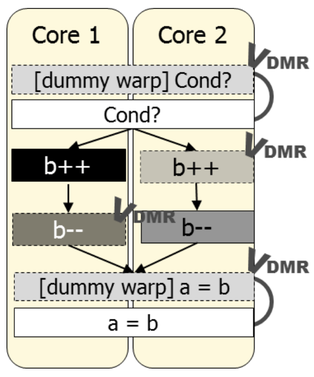
- For instructions i, *required threads for* i $<$ *SIMT lane capacity of system*
- Inactive threads within a warp duplicate active threads execution
- Error detection by comparing the computation result of the inactive and active threads

```
If(cond) {
    b++;
} else {
    b--;
}
a = b;
```



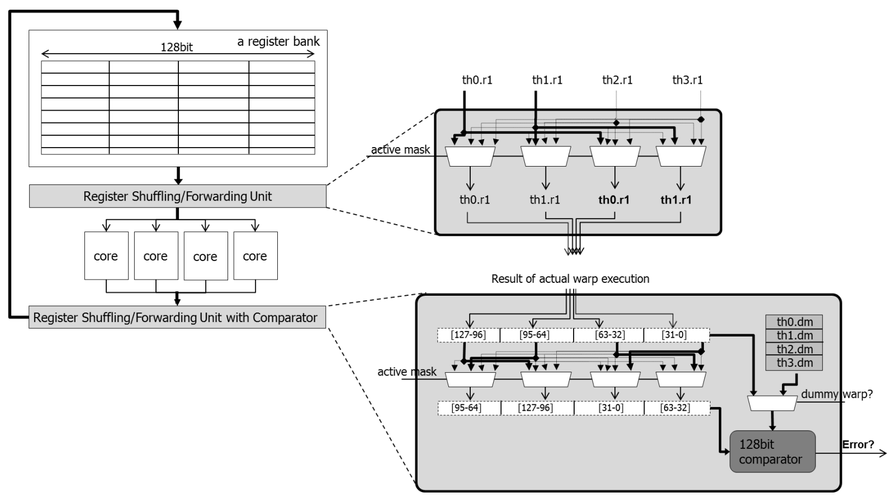
(a) code (b) dream (c) reality (d) proposed

Inter-Warp Checking



- For instructions i, *required threads for* i $=$ *SIMT lane capacity of system*
- Dummy warp duplicates an actual warp execution
- Error detection by comparing the execution result of actual and dummy warp

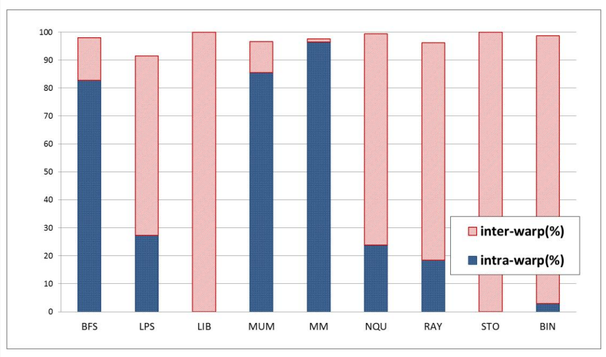
Modified Architecture



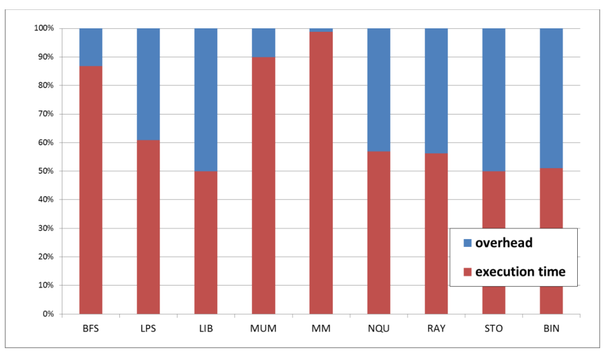
Register Shuffling/Forwarding Unit is added

- Before and after pipeline execution stage
- Registers of active threads are forwarded to inactive threads
- Registers are forwarded after shuffling to dummy warps
- 128-bit comparators for verifying computation results

Simulation Results



Error Coverage



Timing Overhead

Simulation Setting

- Benchmarks : NVIDIA CUDA SDK, Parboil, and ERCBench
- Simulator : GPGPU-Sim

Results

- **Average Error coverage is 97.61 %**
- **Average Timing overhead is 60.09 %**
  → Negligible when considering GPGPU's 10x~100x speedup over CPU

Ming Hsieh Institute
Ming Hsieh Department of Electrical Engineering