GPU-Trident: Efficient Modeling of Error Propagation in GPU Programs
November 19, 2020

Abdul Rehman Anwer
Guanpeng Li
Karthik Pattabiraman
Siva Hari
Michael Sullivan
Timothy Tsai
Soft Errors

The ionizing track left by secondary particles can cause an erroneous current pulse in one or multiple neighboring transistors.
Silent Data Corruption (SDC)

- Normal Execution
- Fault
- Error Propagation
  - Benign
  - Crash
  - SDC (Silent Data Corruption)
  - Incorrect Output
  - Correct Output
  - Exceptions, No Output

E.g. Amazon S3 Incident
• Error Correction Code (ECC)

• Hardware means

  • **Circuit hardening**
    Wastes power as gap between average and worst-case widens due to variations

  • **Guard-band**
    Hardware duplication (i.e., DMR) can result in 2X slowdown and/or energy consumption


Very expensive to deploy in practice!
Software protection techniques are more flexible and cost-effective!
$A = 0x00
$B = A + 0x04
$C = A + B

Golden Output

0 0 0 1 0 1 1 1

FI Output

0 0 1 0 1 1 1 1

Compare

Program

$A = 0x00
$B = A + 0x04
$C = A + B

Program

$A = 0x00
$B = A + 0x04
$C = A + B
FI injection – Overhead

- GPU-Qin [ISPASS’14, Fang], LLFI-GPU [SC’16, Li], SASSIFI [ISPASS’17, Hari]

- Highly inefficient, as it has to be repeated if application is updated

~7 hrs. for 100 faults per instruction

*Timings obtained from our experiments using LLFI-GPU, other works report similar timings
Trident – for CPU (DSN’18, Li)

Source Code
Program Input
Output Insn.

Insn. for Prediction

Profiling
Prediction

Inst. SDC prob.
Overall SDC prob.
• Execution of GPU applications is inherently multi threaded
• Threads frequently communicate with each other

Interleaving dependencies complicate error propagation

Thread 1
- store
- load

Shared Memory
- 0x00
- 0x04
- 0x08

Thread 2
- store
- load

[SC’16, Li]
No. of Threads

Thread

No. of Executions

CPU Program

GPU Program

- Average of ~5 years, Max 17 years
- > 5GB, data to be profiled in Circuit
- Inaccuracies in model accumulate
Challenges - Summary

1. Tracking error across threads
2. Huge amount of states to profile
3. Accumulation of inaccuracies
1. Tracking error across threads
2. Huge amount of states to profile
3. Accumulation of inaccuracies
Updating $f_m$

- $f_m$ constructs a memory dependency graph between instructions.
- We construct graph of whole kernel, instead of individual threads.

```c
__global__ void staticReverse(int *d, int n)
{
    __shared__ int s[64];
    int t = threadIdx.x;
    int tr = n-t-1;
    s[t] = d[t];
    __syncthreads();
    d[t] = s[tr];
}
```
Constructing the dependency graph

- Memory dependency, based on control-flow
- Limited possible control-flows

**Solution:** Sample threads with unique control-flows for profiling e.g. 3,840 out of 592,640 threads profiled for Pathfinder
Challenges

1. Tracking error across threads
2. Huge amount of states to profile
3. Accumulation of inaccuracies
Lucky Stores

- If a memory contains the same data we want to store in it
- Missing that store won’t result in any SDC

**Solution:** Update $f_c$ to modify propagation probability of comparisons dominating output stores.
1. Tracking error across threads

2. Huge amount of states to profile

3. Accumulation of inaccuracies

\[ f_m, f_s, f_c, f_S \]
Trident - Workflow

Profiling

Execution

Mem Profiling

$f_S$,

$f_c$,

$f_m$
GPU-Trident - Workflow

Parallelize this using CPU resources

Profiling → $f_S$ → Execution

Profiling → $f_c$ → Execution

Profiling → $f_m$ → Execution

Execution → Mem Profiling

Mem Profiling → $H$-Intra

Mem Profiling → $H$-Inter

$H$-Value
GPU-Trident: A set of LLVM passes, driven by python scripts

URL: https://github.com/DependableSystemsLab/GPU-Trident

Use LLFI-GPU for FI and use it as a baseline

Predict SDC probability with GPU-Trident, compare with FI

Evaluate GPU-Trident for 17 kernels (Rodinia and OSS HPC) at
  • Kernel level
  • Instruction level
$A = 0x00$

$B = A + 0x04$

$C = A + B$
Evaluating Kernel SDC probability

5,000 random FI trials per kernel
Evaluating Instruction SDC probability

\[ A = 0x00 \]
\[ B = A + 0x04 \]
\[ C = A + B \]

\[ A = 0x00 \]
\[ B = A + 0x04 \]
\[ C = A + B \]

\[ A = 0x00 \]
\[ B = A + 0x04 \]
\[ C = A + B \]

\[ A = 0x00 \]
\[ B = A + 0x04 \]
\[ C = A + B \]

\[ \ldots \]

\[ T1 \]
\[ T2 \]
\[ Tn \]

\[ 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \]

100 random FI trials per instruction
Mean absolute error for kernel SDC is 5.7% (Trident has error of 4.75%)

Pearson correlation coefficient for kernel SDC is 0.88 (Without outliers 0.99)

Average Pearson correlation coefficient for instructions is 0.83
• Kernel SDC probability

<table>
<thead>
<tr>
<th>FI trials</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>11x</td>
</tr>
<tr>
<td>3000</td>
<td>33x</td>
</tr>
<tr>
<td>5000</td>
<td>55x</td>
</tr>
</tbody>
</table>

• Instruction SDC probability
  • GPU-Trident is 2 orders of magnitude (~100x) faster than FI
  • FI takes 7 hrs, while GPU-Trident takes less than 5 minutes

GPU-Trident needs to construct model once, while each FI trials requires an application run
Software protection techniques are more flexible and cost-effective!
Selective Instruction Duplication

- Proposed by [DAC’09, Reddi], [ASPLOS’10, Feng], [CGO’16, Laguna]
Selective Instruction Duplication
 Modeling error propagation in GPU applications is challenging due to *scale* and *inaccuracy*.

- We develop heuristics, based on *memory access* and *data patterns*.

- Experiments show our techniques are *accurate* and *scalable*.


- For questions: arehman.anwer1@gmail.com