SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing

Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni

William & Mary
GPUs & Soft Errors

- GPUs are commonly deployed
- GPUs are prone to **soft errors**
  - High-energy radioactive particles (i.e., cosmic rays) cause **bit flips**
  - Commonly observed
  - Impact on long-running applications can be tremendous
    - **Masked** output: Correct
    - **Other** outputs: Crash, hang, ...
    - **Silent Data Corruption (SDC)** output: Incorrect
  - SDCs in critical applications can be dangerous


[http://global.atpinc.com/Memory-insider/what-is-soft-error-detection-sram-emmc](http://global.atpinc.com/Memory-insider/what-is-soft-error-detection-sram-emmc)
Reliability Research: Fault Injection

- Inject **single-bit errors** into different locations (**fault sites**) in applications

- Ground truth: **huge unreachable exhaustive fault sites!**

<table>
<thead>
<tr>
<th>2DCONV</th>
<th>Input size</th>
<th>Small</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of Elements</td>
<td>32 × 32</td>
<td>2048 × 2048</td>
<td></td>
</tr>
<tr>
<td>Num. of Threads</td>
<td>1024</td>
<td>4.19 × 10⁶</td>
<td></td>
</tr>
<tr>
<td>Num. of Fault Sites</td>
<td>1.90 × 10⁶</td>
<td>8.71 × 10⁹</td>
<td></td>
</tr>
<tr>
<td>Execution Time (Simulation)</td>
<td>1 sec</td>
<td>10 min</td>
<td></td>
</tr>
</tbody>
</table>

- Fault site selection:
  - **Random sampling** based on statistics: 1K runs, ±3% error
  - *(state-of-the-art)* **Fault site pruning***: 300~2K runs, ~1% error
  - Resilience proxy: Dynamic Instruction (DI) count

Fault site pruning

- 10 min × 440 = 1.2 h
- 1.2 h × ∞ inputs = ∞ h
- 1 sec × 440 = 7.3 min
- 7.3 min × ∞ inputs = 7.3 min

---

GPGPU Application Parallelization

*CTA: Cooperative Thread Array

GPGPU Application

Kernel 1

Kernel 2

Kernel n

CTA 1

CTA 2

... CTA m

Warp 1

Warp 2

... Warp k

Simplest kernel function: vectors $\mathbf{u} + \mathbf{v}$

Matrix operations:

CTA 1

CTA 2

CTA 3

CTA 4
**SUGAR Idea (Example: PathFinder)**

**DI patterns**

**Resilience patterns**

*DI: Dynamic Instruction*
Why?

• Code snippet of PathFinder

```c
1 int bx = blockIdx.x;
2 int tx = threadIdx.x;
3 int small_block_cols = BLOCK_SIZE-iteration*HALO*2; // Data chunk size
4 int blkX = small_block_cols*bx-border; // Chunk start position
5 int blkXmax = blkX+BLOCK_SIZE-1; // Chunk end position
6 int idx = blkX+tx; // Global thread ID
7 int validXmin = (blkX < 0) ? -blkX : 0; // Valid ch
8 int validXmax = (blkXmax > cols-1) ? \\
9 BLOCK_SIZE-1-(blkXmax-cols+1) : BLOCK_SIZE-1; /\n10 bool isValid = IN_RANGE(tx, validXmin, validXmax);
11 ....
12 for (int i=0; i<iteration; i++){
13    ....
14 if( IN_RANGE(tx, i+1, BLOCK_SIZE-i-2) && isValid){
15      // If valid, take the branch.
16    }
17  }
18 }
```

✓ DI Patterns —— Resilience Patterns

Input does **NOT** change branch divergence

DI-insensitive

✓ Small —— Large

Trend of repeating patterns

Calculate Boundary

Check Validation

Only valid threads perform computation & touch the data
What if input changes branch divergence?

- Code snippet of BFS K8

```c
int tid = blockIdx.x * MAX_THREADS_PER_BLOCK + threadIdx.x;
if (tid < no_of_nodes && g_updating_graph_mask[tid]) {
    g_graph_mask[tid] = true;
    g_graph_visited[tid] = true;
    *g_over = true;
    g_updating_graph_mask[tid] = false;
}
```

- Input data
- Input changes branch divergence
- DI-sensitive
- Check Validation
DI-sensitive Patterns (Example: BFS)

- DI patterns:
  - G1
  - G2

- Resilience patterns:
  - G1
  - G2

Small → +DI Profiling → Medium → +DI Profiling → Large
**SUGAR Workflow**

1. **Classify Application Type and CTA Pattern**
   - Input changes branch condition?
   - DI-insensitive
   - 1D-trend Frequency ($P_i$)

2. **Form Input S**
   - Larger Input
   - Dominant Pattern

3. **Perform Experiments**

4. **Estimate Application Kernel Resilience**
   - Resilience Estimation Results

**Target Application** + **Target Input**
SUGAR Workflow

1. Classify Application Type and CTA Pattern
   - Input changes branch condition?
   - DI-insensitive
     - 1D-trend Frequency($P_i$)
     - 2D-trend Frequency($P_i$)

2. Form Input $S$
   - Larger Input
   - Dominant Pattern

3. Perform Experiments
   - Resilience Estimation Results

Target Application + Target Input

- HotSpot
SUGAR Workflow

1. Classify Application Type and CTA Pattern
   - Input changes branch condition?
     - N: DI-insensitive
     - Y: DI-sensitive

2. Form Input S

3. Perform Experiments
   - 1D-trend Frequency($P_i$)
   - 2D-trend Frequency($P_i$)
   - DI profiling Frequency($G_i$)

4. Estimate Application Kernel Resilience
   - Pattern Frequency
   - Resilience Estimation Results
SUGAR Workflow

1. Classify Application Type and CTA Pattern
   - Input changes branch condition?
     - N: DI-insensitive
     - Y: DI-sensitive

2. Form Input S
   - Random Sampling
   - Exploratory Runs
   - Input S

3. Perform Experiments
   - DI profiling
     - 1D-frequency ($F_{reqency}(P_i)$)
     - 2D-frequency ($F_{reqency}(P_i)$)

4. Estimate Application Kernel Resilience
   - Pattern Frequency

Target Application

Target Input

Random Sampling

Exploratory Runs

Smallest Size

Resilience Estimation Results

1D-trend $F_{reqency}(P_i)$

2D-trend $F_{reqency}(P_i)$
SUGAR Workflow

1. Classify Application Type and CTA Pattern
   - Input changes branch condition?
     - N: DI-insensitive
     - Y: DI-sensitive

2. Form Input S
   - Random Sampling
   - Exploratory Runs
   - Smallest Size

3. Perform Experiments
   - Fault Injection Campaign
     - DI-insensitive: Resilience($P_i$)
     - DI-sensitive: Resilience($G_i$)

4. Estimate Application Kernel Resilience
   - Pattern Frequency
   - Resilience Estimation Results
Random Sampling
Smallest Size

1. Classify Application Type and CTA Pattern

Input changes branch condition?

DI-insensitive

Y

DI-sensitive

N

Di profiling
Frequency \((G_i)\)

2. Form Input S

Random Sampling
Exploratory Runs

Input S

Smallest Size

3. Perform Experiments

Fault Injection Campaign
DI-insensitive: Resilience \((P_i)\)
DI-sensitive: Resilience \((G_i)\)

4. Estimate Application Kernel Resilience

Pattern Frequency

Pattern Resilience

Resilience (Kernel) = \[
\begin{cases} 
\sum_{i=1}^{n} \text{Resilience}(P_i) \times \text{Frequency}(P_i) & \text{DI-insensitive} \\
\sum_{i=1}^{n} \text{Resilience}(G_i) \times \text{Frequency}(G_i) & \text{DI-sensitive} 
\end{cases}
\]
SUGAR Workflow

1. Classify Application Type and CTA Pattern
   - Input changes branch condition?
     - N: DI-insensitive
     - Y: DI-sensitive

2. Form Input S
   - Random Sampling
   - Exploratory Runs
   - Smallest Size

3. Perform Experiments
   - Fault Injection Campaign
     - DI-insensitive: $Resilience(P_i)$
     - DI-sensitive: $Resilience(G_i)$

4. Estimate Application Kernel Resilience
   - DI profiling $Frequency(G_i)$
   - DI-insensitive: $Resilience(P_i)$
   - DI-sensitive: $Resilience(G_i)$

Target Application

Target Input

Resilience Estimation Results
Evaluation: Accuracy

- Baseline: *Fault Site Pruning (state-of-the-art)*

<table>
<thead>
<tr>
<th></th>
<th>Masked</th>
<th>SDC</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td>0.68%</td>
<td>0.64%</td>
<td>0.25%</td>
</tr>
<tr>
<td>Large</td>
<td>1.14%</td>
<td>1.07%</td>
<td>0.31%</td>
</tr>
</tbody>
</table>

Evaluation: Accuracy

- Baseline: **Fault Site Pruning (state-of-the-art)**
- Asymptote resilience estimation
- Resilience trend:

### Average Difference

<table>
<thead>
<tr>
<th></th>
<th>Masked</th>
<th>SDC</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td>0.68%</td>
<td>0.64%</td>
<td>0.25%</td>
</tr>
<tr>
<td>Large</td>
<td>1.14%</td>
<td>1.07%</td>
<td>0.31%</td>
</tr>
<tr>
<td>All</td>
<td>0.89%</td>
<td>0.83%</td>
<td>0.28%</td>
</tr>
</tbody>
</table>

Average speedup:
- Medium: 7.3x
- Large: 186.6x
SUGAR Summary

**SUGAR Idea (Example: PathFinder)**

- DI Patterns
- Resilience Patterns

- Small
- Medium
- Large

- DI-sensitive Patterns
- OD Polishing
- HD Polishing
- Large

- Robustness Patterns

**SUGAR Workflow**

1. Cloudy Application Type and DI Pattern
2. Performed Experiments
3. Determine Application-level DI Pattern
4. Resilience Evaluation

**Evaluation: Accuracy**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SUGAR</th>
<th>Baseline</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Difference</td>
<td>1.14%</td>
<td>1.03%</td>
<td>0.53%</td>
</tr>
</tbody>
</table>

**Magic? SUGAR**

- 1 sec × 440 = 7.3 min
- 7.3 min × ∞ inputs = 7.3 min

- DI Patterns → Resilience Patterns
- Small → Large

- Accurate
- Fast
Thank you :)  

**SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing**

Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni  
William & Mary

This work is supported by NSF grant CCF-1717532