Analysis and Mitigation of Soft-Errors on High Performance Embedded GPUs

Luca Sterpone
Sarah Azimi
Corrado De Sio

Politecnico di Torino

Dipartimento di Automatica e Informatica

Filippo Parisi

PUNCH | Torino

Goals

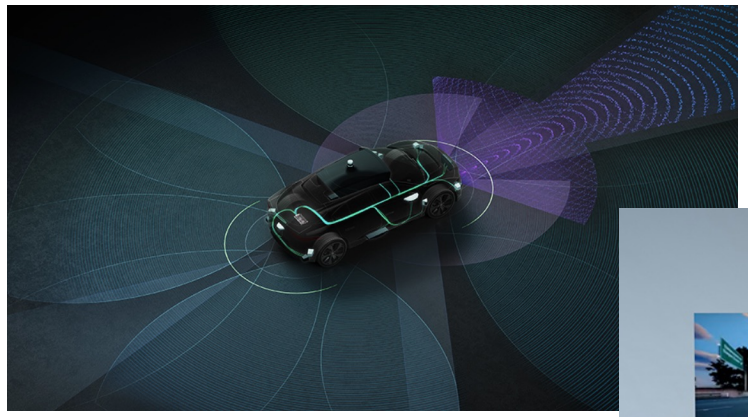
- **A reliability analysis methodology for embedded General Purpose GPUs (GPGPUs) adopting emulation-based fault injection**
  - Injection of Single Event Upsets (SEUs)
  - Instrumentation of the Streaming ASSembler (SASS)
- **A software mitigation method implementing dependable computing strategies for GPGPUs**
  - Redundancy with negligible computational time overhead

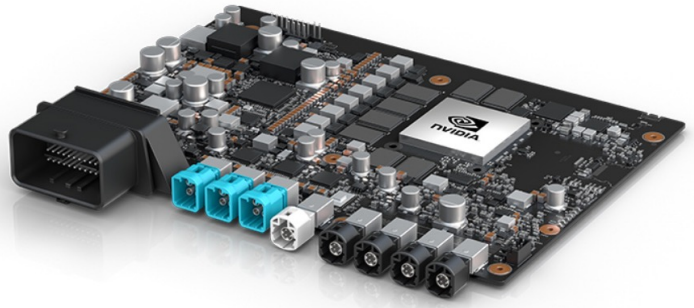
Outline

- Towards GPGPUs for embedded systems
- Space and terrestrial radiation environment
- Soft Errors effects and reliability analysis
- Embedded GPUs computing scenario
- Fault injection method and experimental results
- Software mitigation and experimental results
- Conclusions
- Future works

Towards GPGPUs for embedded systems

- GPGPU applications are nowadays expanding from the domain of High Performance Computing (HPC)
- Embedded GPUs are one of the main solutions for self-driving cars and autonomous vehicles

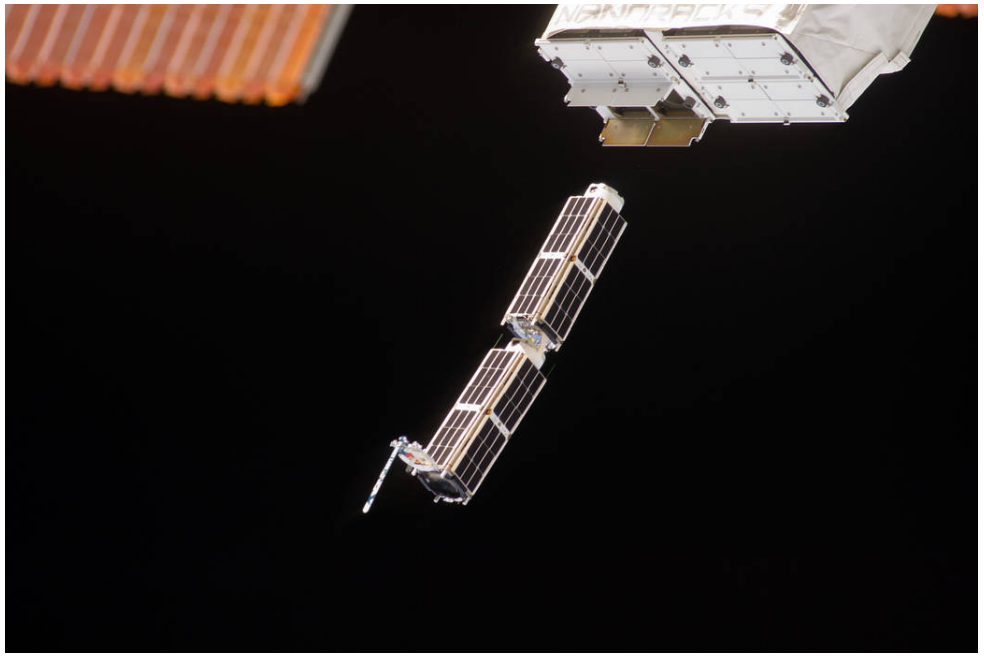




[NVIDIA courtesy]

Towards GPGPUs for embedded systems

- **Towards an Integrated GPU Accelerated SoC as a Flight Computer for Small Satellites (CubeSat) or more…**



[Nanoracks courtesy]

[NASA / ESA / CSA]

Space Radiation Environment

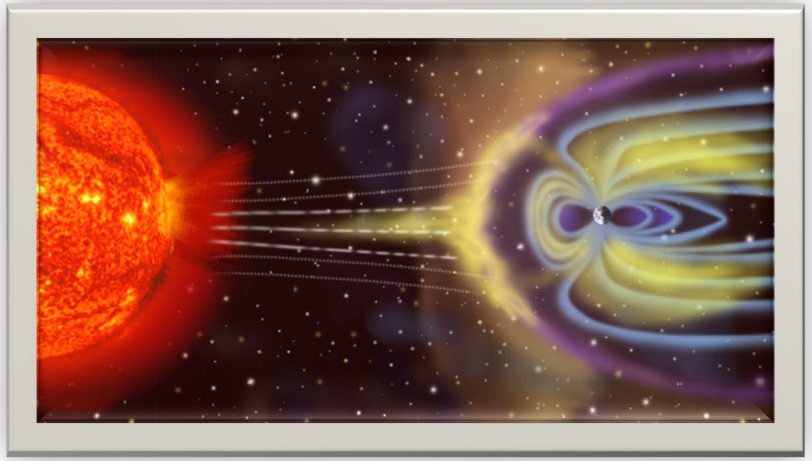
- **Towards safety and mission-critical applications**

SPACE Environment

Galactic Cosmic Ray
Protons
Heavy Ions

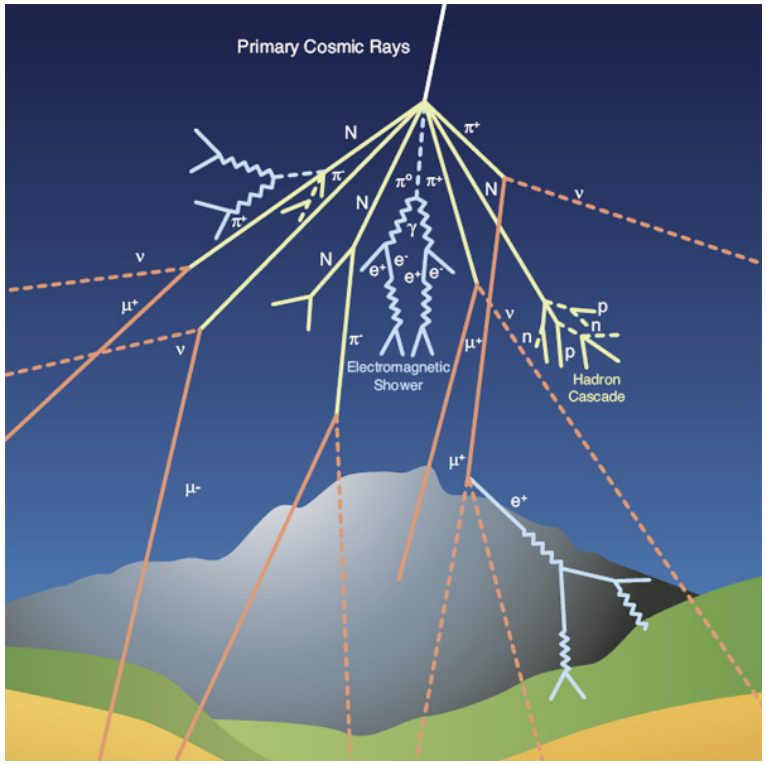
Supernova explosions
Planets collisions
Star collisions

Solar wind and flares
Protons
Heavy Ions

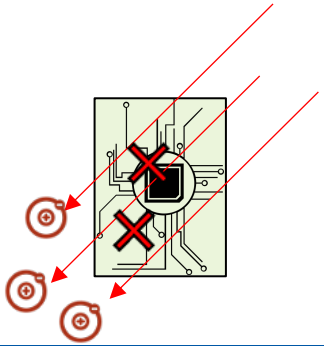


Van Allen Belt
Protons
Electrons

Terrestrial Radiation Environment

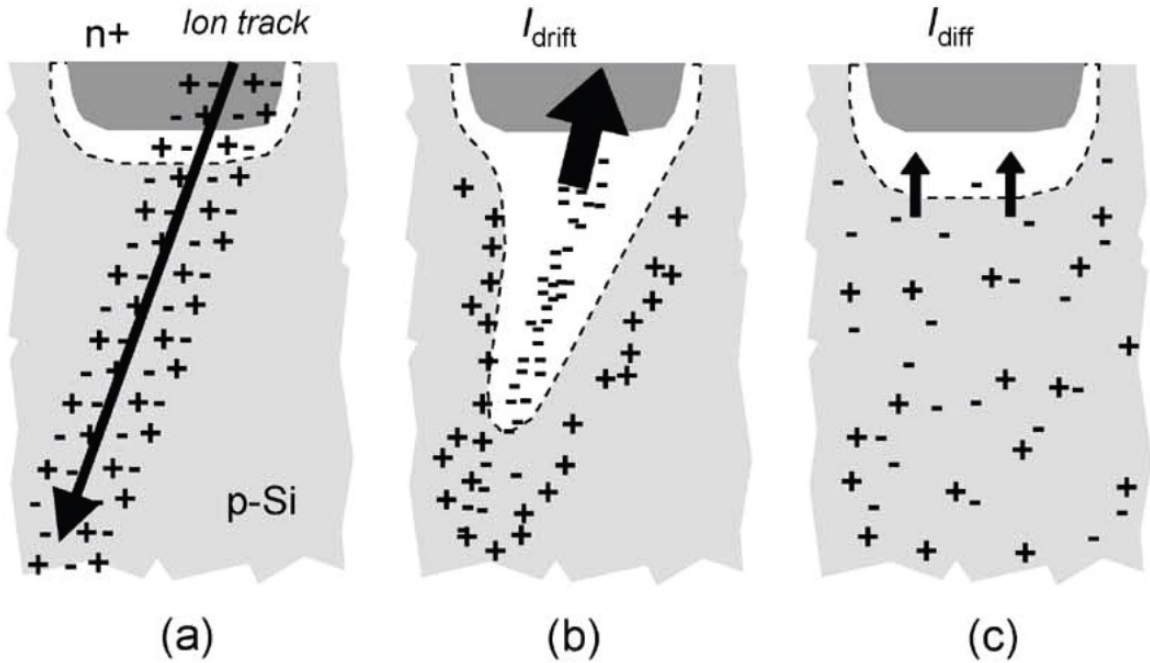


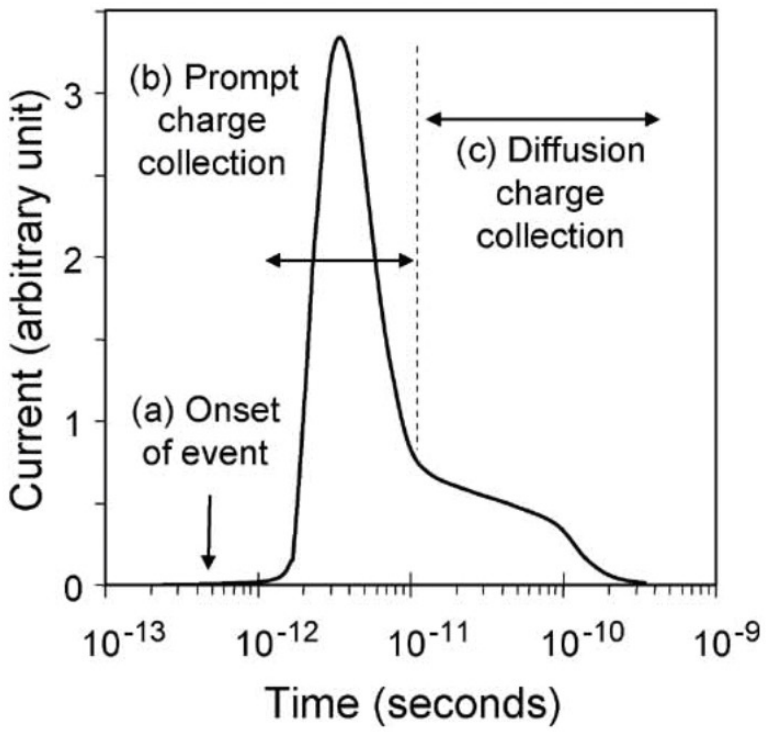
- **Galactic Cosmic Ray interacts with atmosphere**
- **Generation of a shower of energetic particles**
  - Especially **Neutrons**
  - JEDEC JESD89A Standard reports up to 13 n/(cm² · h) at sea level
- **Neutrons affect modern distributed computer systems and architectures**



Soft Error

- **A radiation particle can generate a Soft Error also known as Single Event Upset (SEU) if affecting a memory element: Data FF or RAM bit-flip**

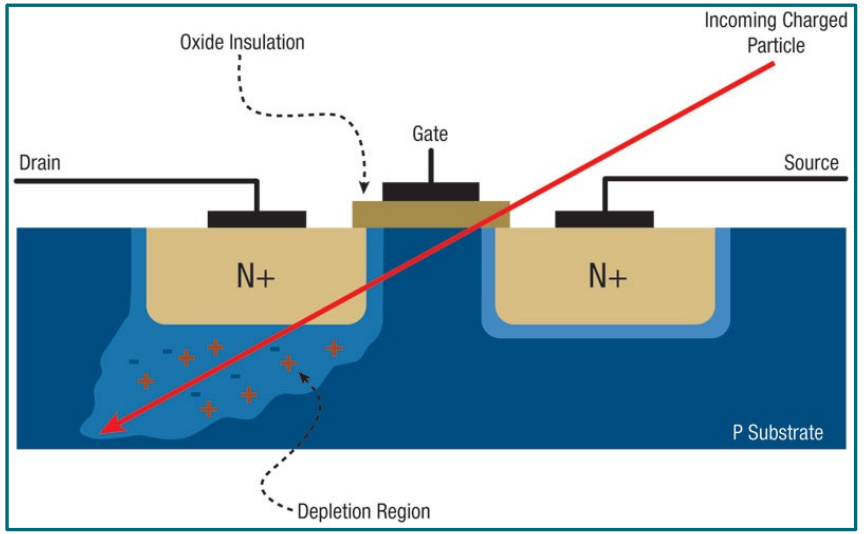




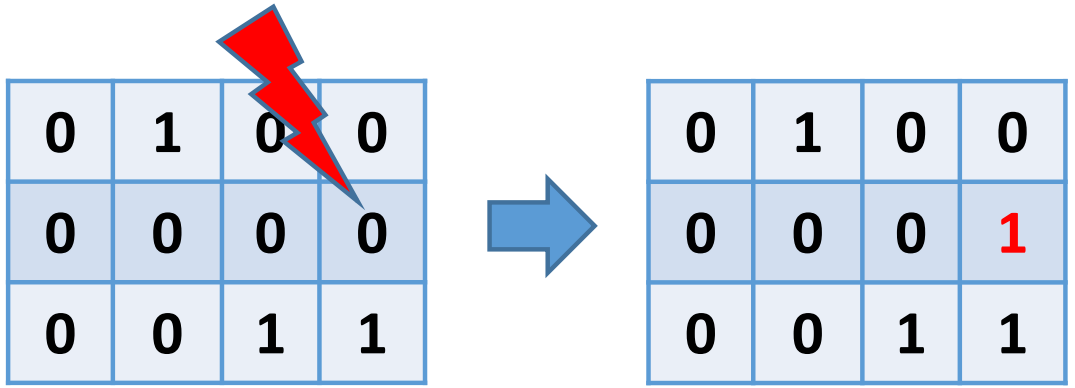
[Baumann 2005]

Soft Error

- **A radiation particle can generate a Soft Error also known as Single Event Upset (SEU) if affecting a memory element: Data FF or RAM bit-flip**



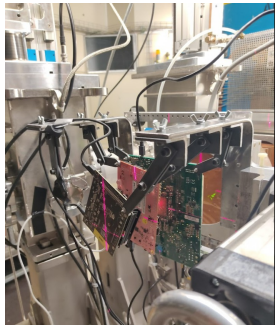
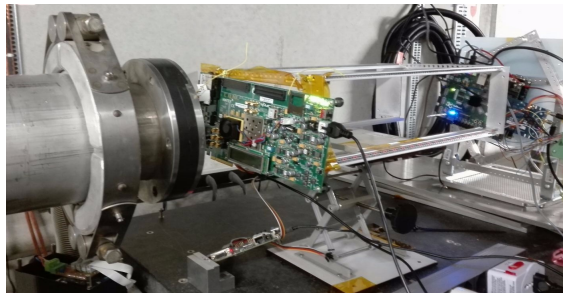
Single Event Upsets (SEUs)



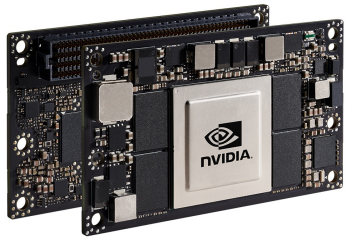
Reliability analysis methodologies

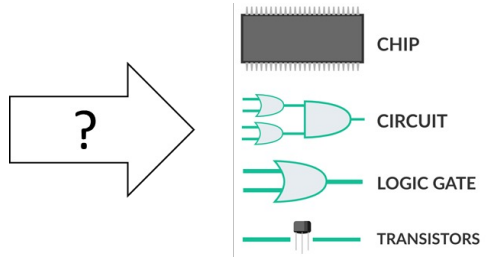
Radiation Testing

- Real Devices + Real Faults
- Costly (time, money, skills)
- Lack of Control

Fault Simulation or Emulation





- Emulate Specific Fault Model
- Good Granularity
- Slow
- Usually, not involving whole hw-level
  (→ not comprehensive)

Reliability analysis - Simulators

- **Several simulators have been developed for micro-architectural level analysis**
- **These tools mimic the parallelism implemented in GPGPUs**
  - Functional GPGPU simulator Barra
  - Heterogenous CPU-GPU simulator gem5-gpu
  - GPGPU-sim
- **Useful for early stage reliability investigations**
  - Fault injection on data variables and streams
- **Inaccurate for SEU injection in architectural resources**

Reliability analysis – Micro Architectural Fault Injections

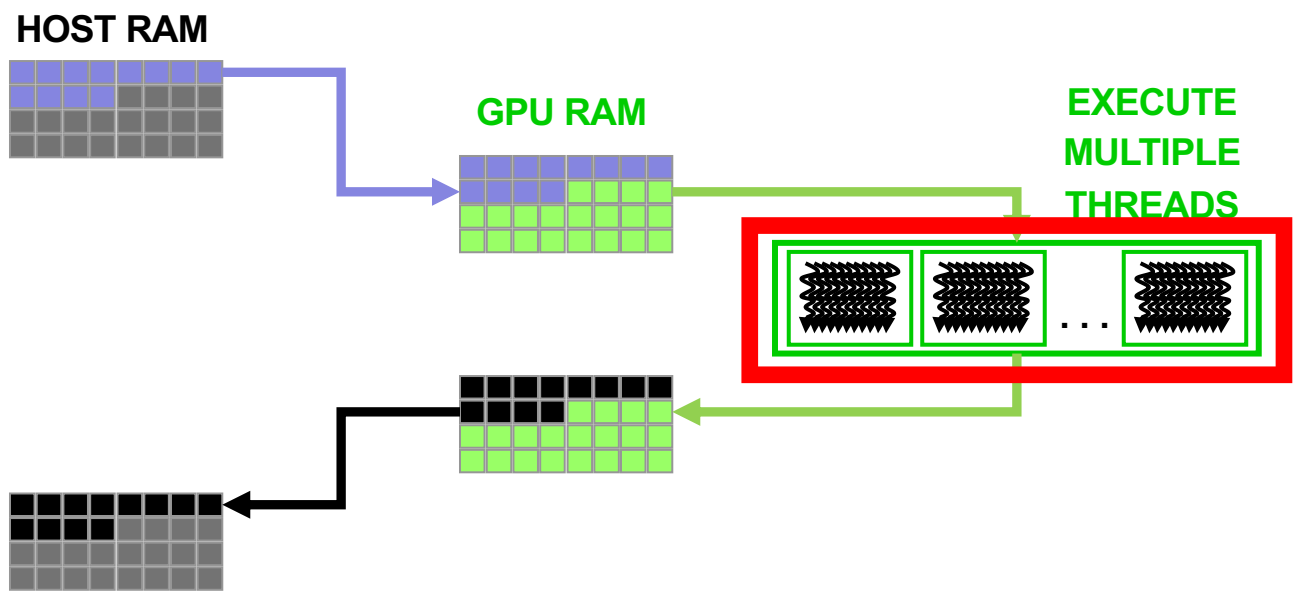
- **Development of approaches able to exploit the Streaming ASSembler (SASS) and to inject faults**
- **SASSIFI was proposed to inject faults in different locations**
  - Register files
  - Shared memories
  - Instruction operands
- **NVBitFI was proposed to intercept dynamic kernels call and insert error without modifying the source code**
  - Without affecting instruction scheduling or register allocations of the target program

Embedded GPUs computing scenario

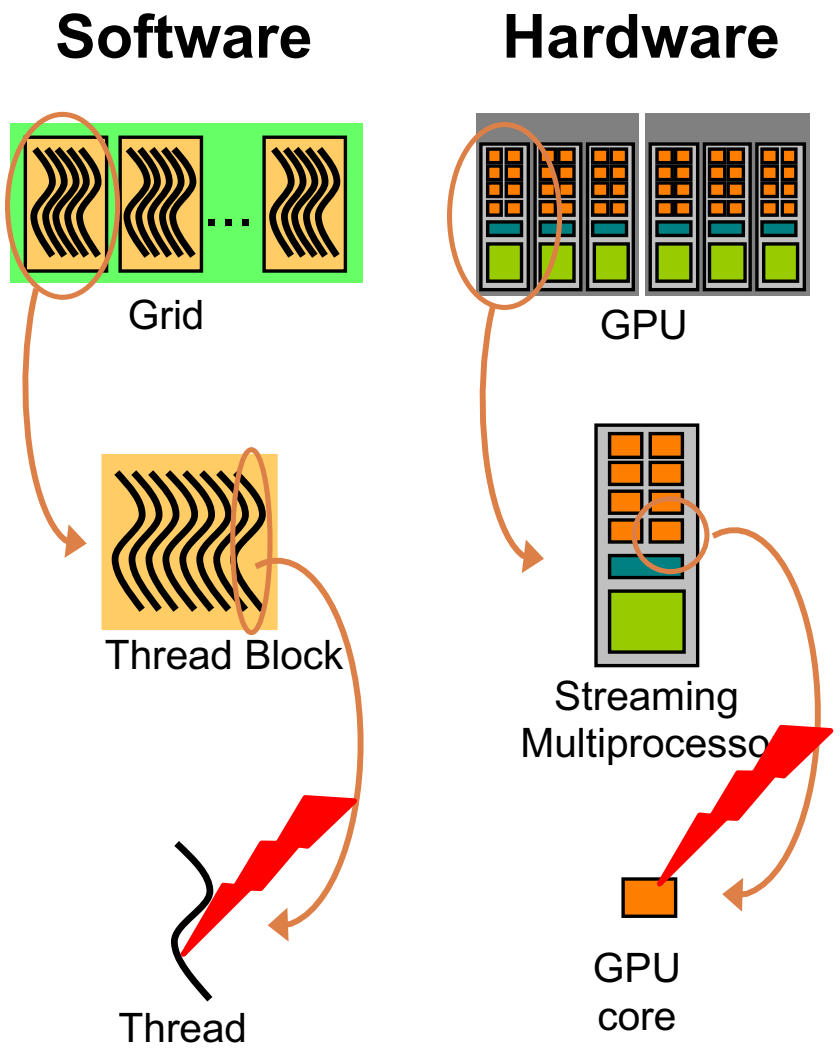
- **GPGPU devices employ a Single Instruction Multiple Threads (SIMT) model**
  - The same instructions are executed by many cores inside the device
- **Each core maintains its own execution flow as a thread**
  - Threads may diverge to perform different tasks
  - Threads may converge via synchronization or barrier instructions

Embedded GPUs computing scenario

- Data must be moved from HOST to DEVICE memory in order to be processed on the GPU
- When data is processed, and no more needed on the GPU, it is transferred back to HOST



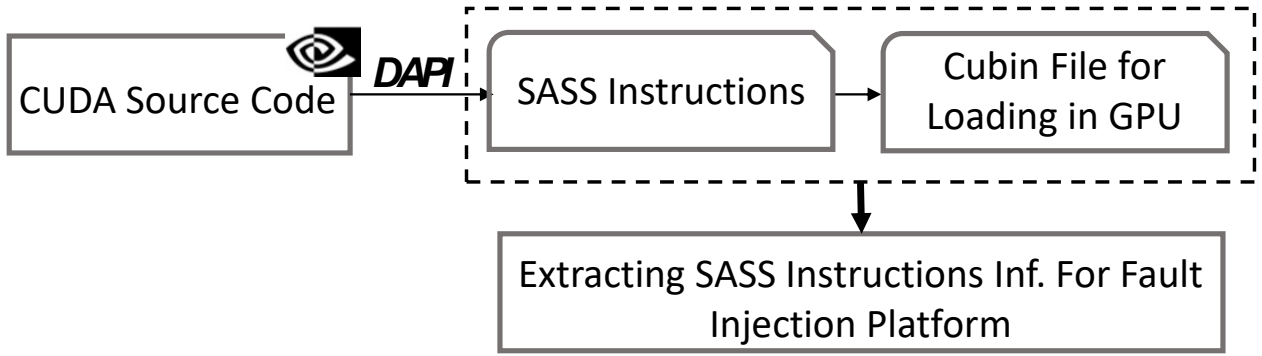
Embedded GPUs computing scenario



- Once the CUDA application is launched, users could organize threads in Blocks and Grids
- Tasks are not always perfectly distributed in threads
- The same fault in terms of corrupted bits and instructions in different threads could yield different results
- The context management is performed by the GPU runtime system

Compilation of source code

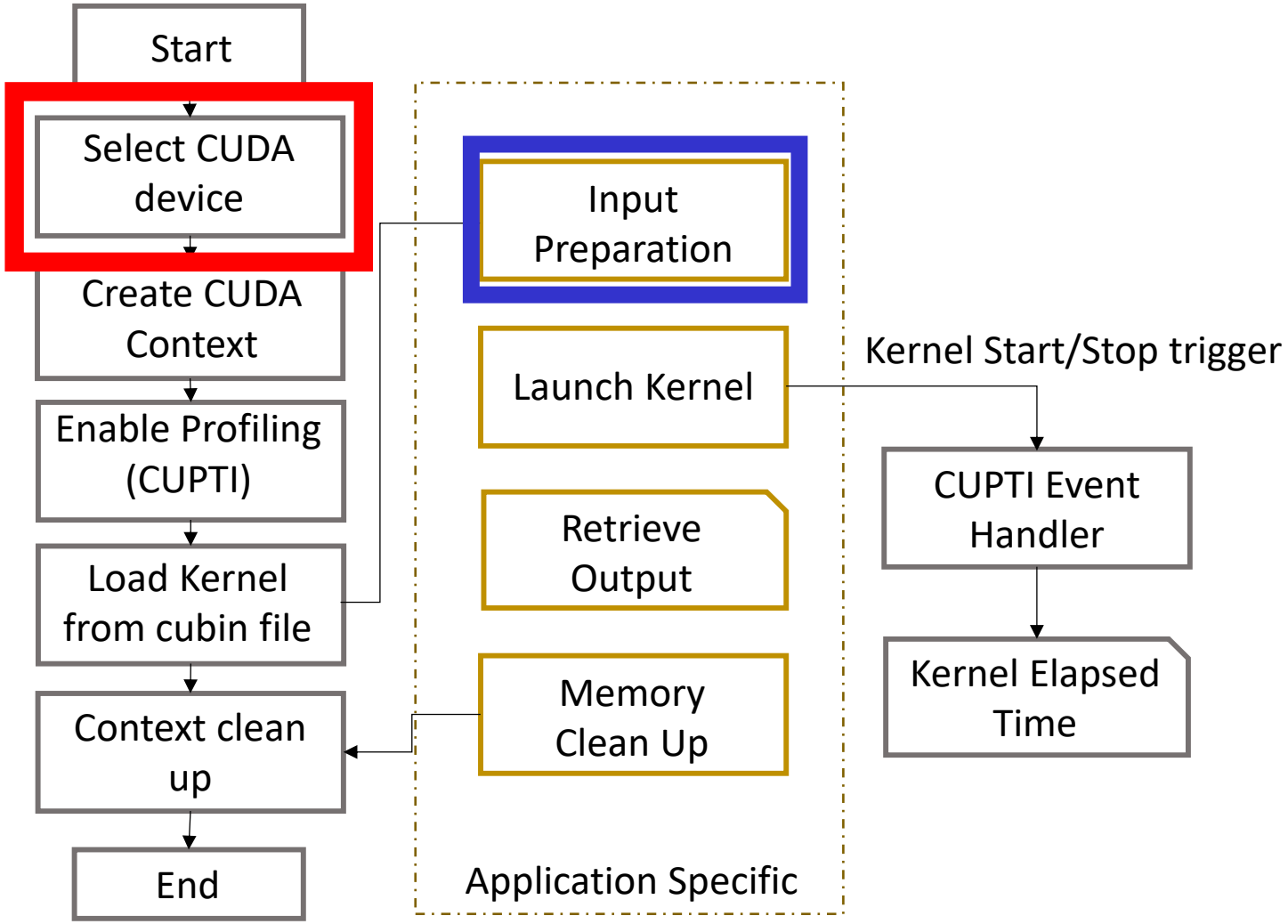
- **NVIDIA CUDA runtime environment (runtime APIs) provides high-level features**
  - Context management
- **Kernel invokes syntax extension and PTX for Just-In-Time (JIT) compilation**
- **JIT compiler transforms PTX code to device specific SASS**
- **We exploited NVIDIA Driver API (DAPI)**



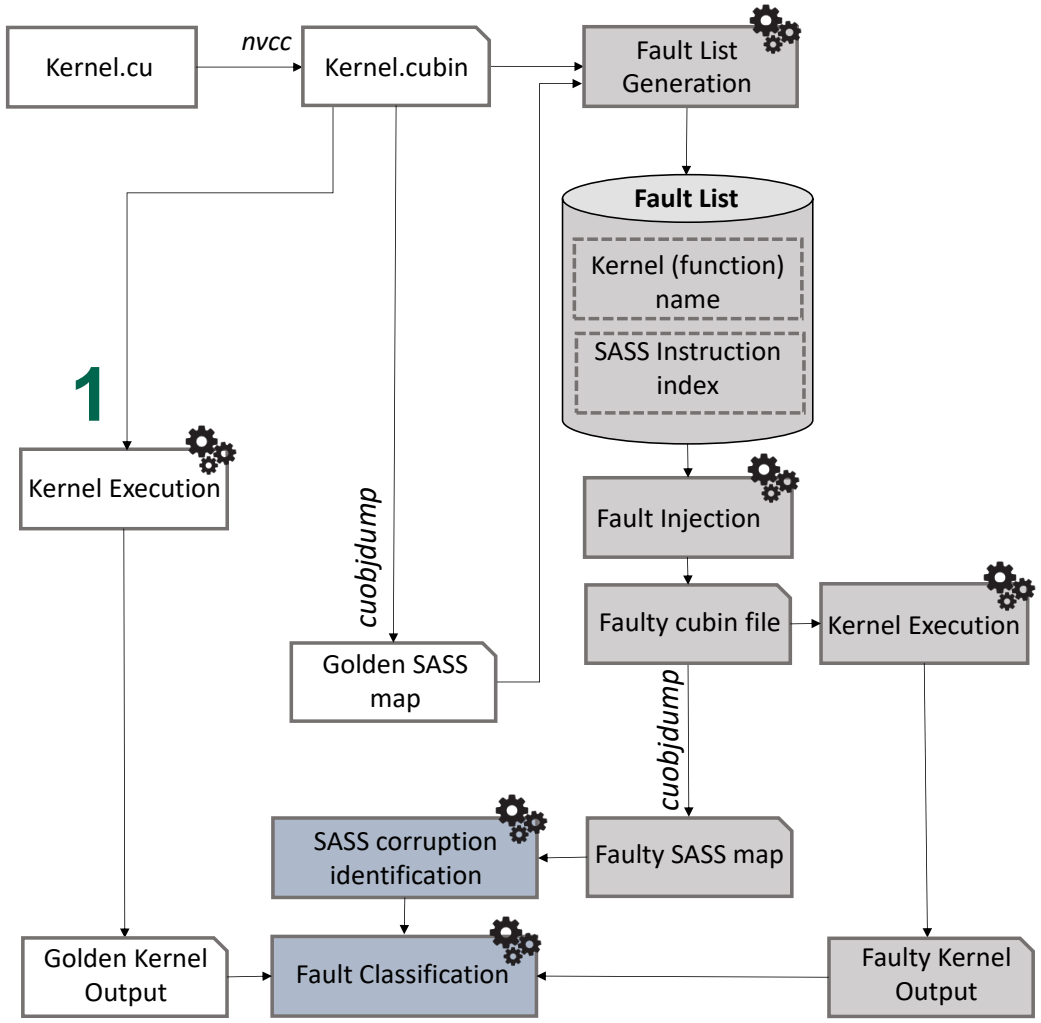
Embedded GPUs computing scenario

- **The developed fault injection tool CUBINJ allows two kinds of bitflips affecting any type of instruction**
  - In all the threads
  - Only in certain specific threads
- **The fault injection tool consists on two flows**
  - Host Application
  - Fault Injection

Host Application Execution Flow

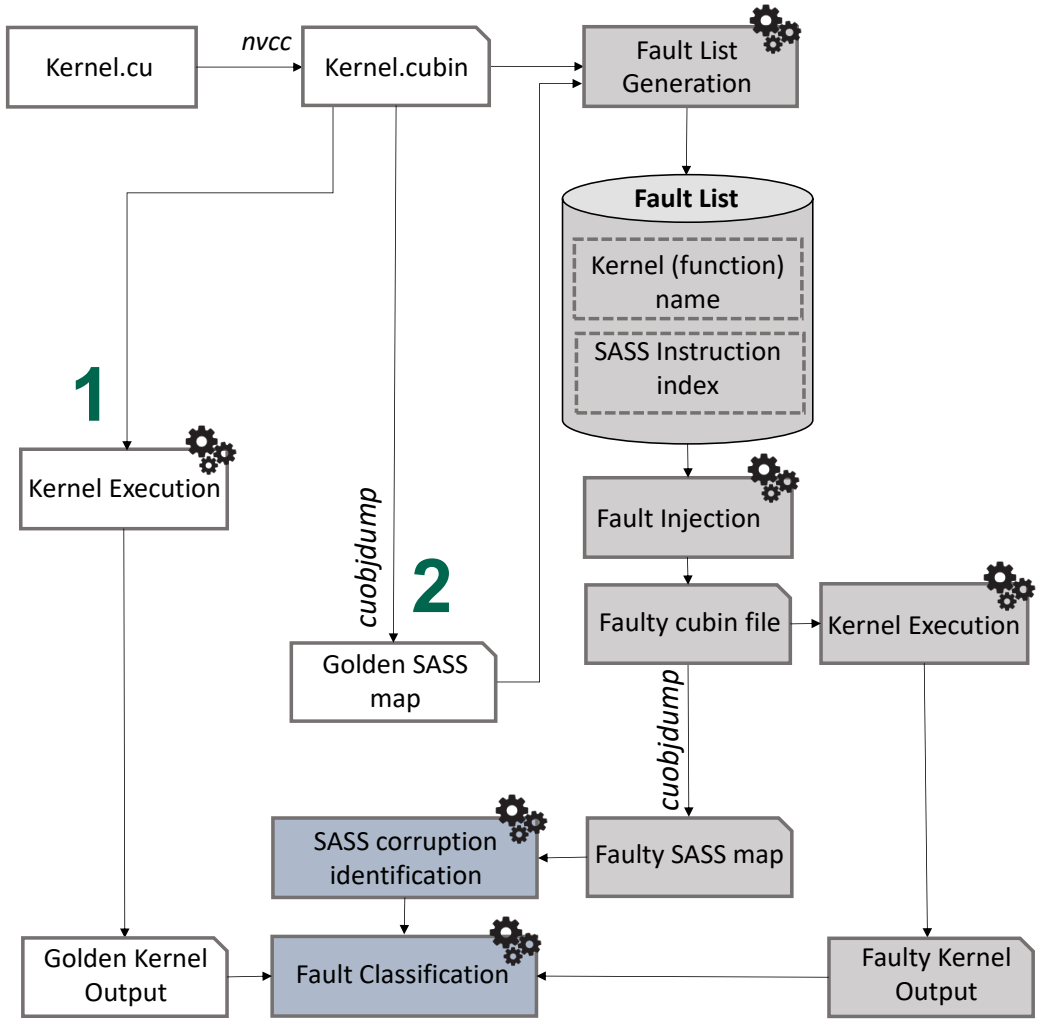


Fault Injection Flow



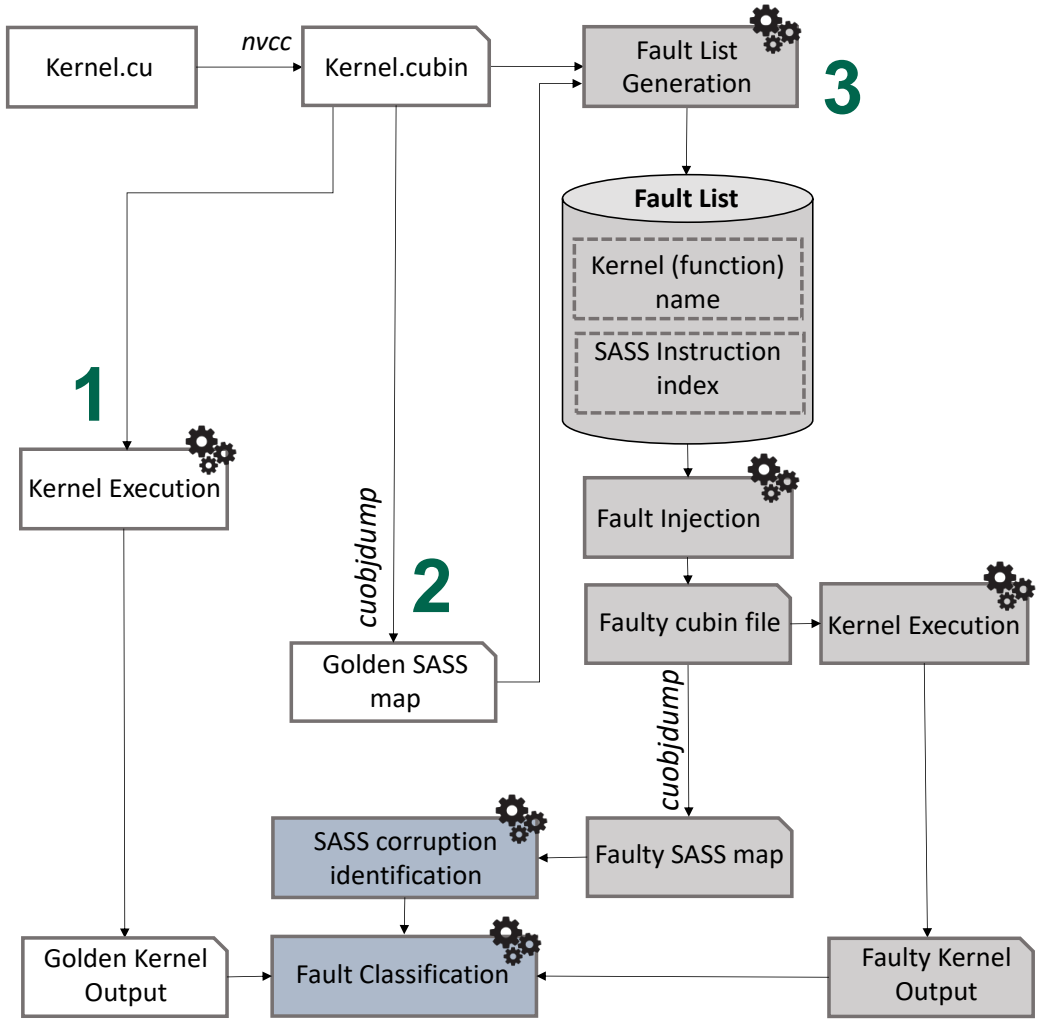
1. **Golden Run execution**

Fault Injection Flow



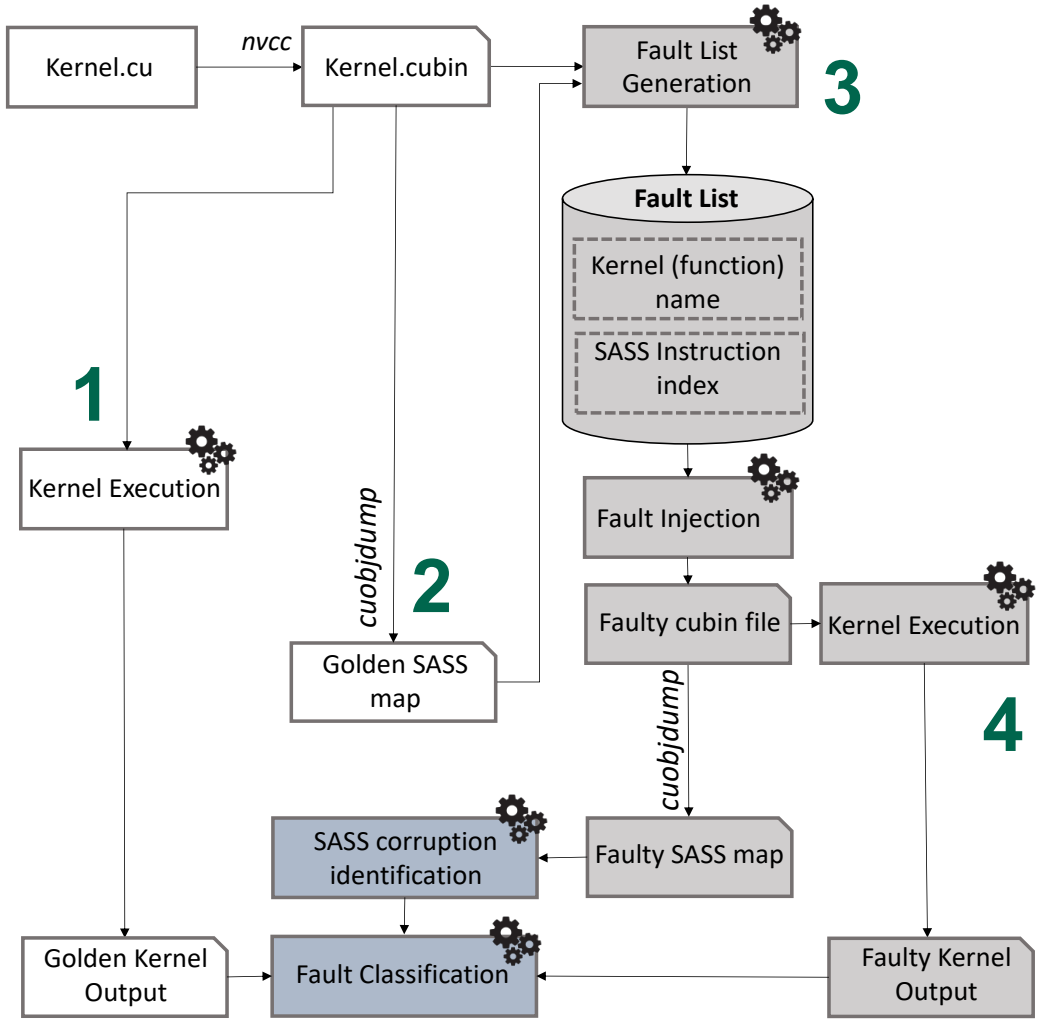
1. Golden Run execution
2. Cuda object dump for fault list generation

Fault Injection Flow



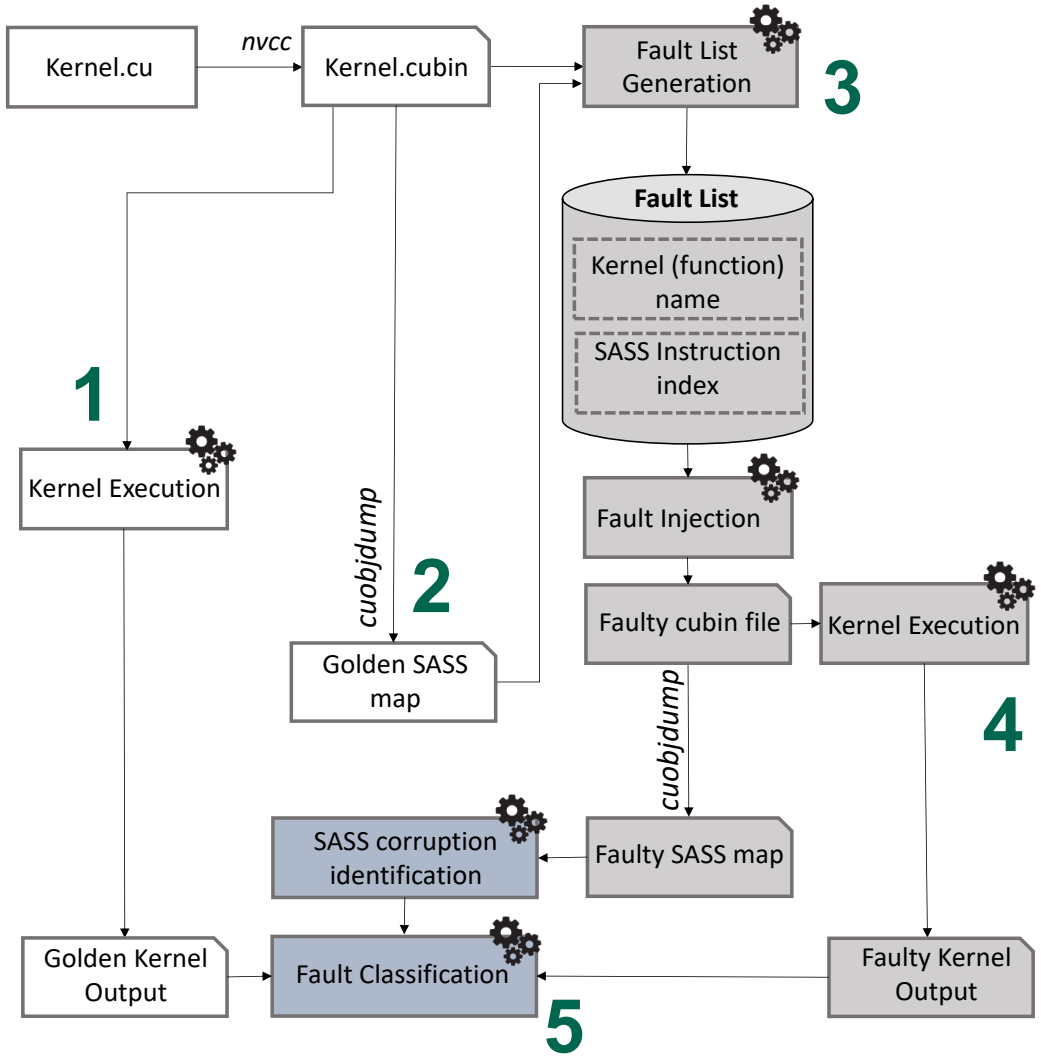
1. **Golden Run execution**
2. **Cuda object dump for fault list generation**
3. **Fault List Generation**
    1. Kernel identification
    2. SASS index

Fault Injection Flow



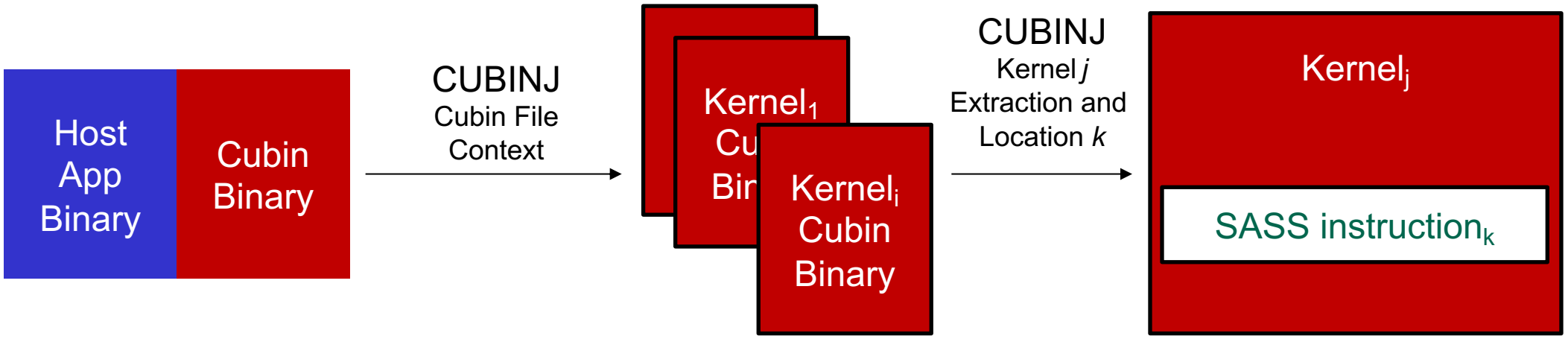
1. **Golden Run execution**
2. **Cuda object dump for fault list generation**
3. **Fault List Generation**
   1. Kernel identification
   2. SASS index
4. **Fault Injection execution**
   1. Cubin file context
   2. Kernel extraction and location
   3. Injection within resource

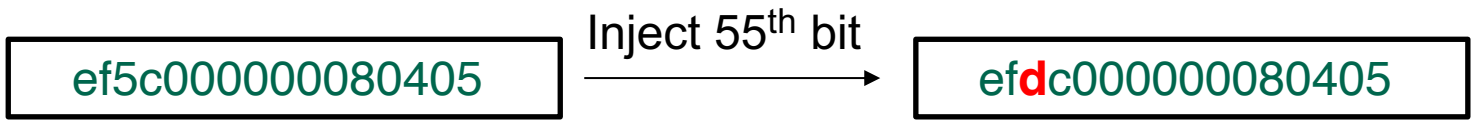
Fault Injection Flow



1. **Golden Run execution**
2. **Cuda object dump for fault list generation**
3. **Fault List Generation**
   1. Kernel identification
   2. SASS index
4. **Fault Injection execution**
   1. Cubin file context
   2. Kernel extraction and location
   3. Injection within resource
5. **Fault Classification**

Back end on Fault Injection Execution





SASS instruction
"@!P0IADD R5, R5, R6 ;"

SASS injected instruction
"@!P0ALD.PHYS R5, a[R4], R0 ;"

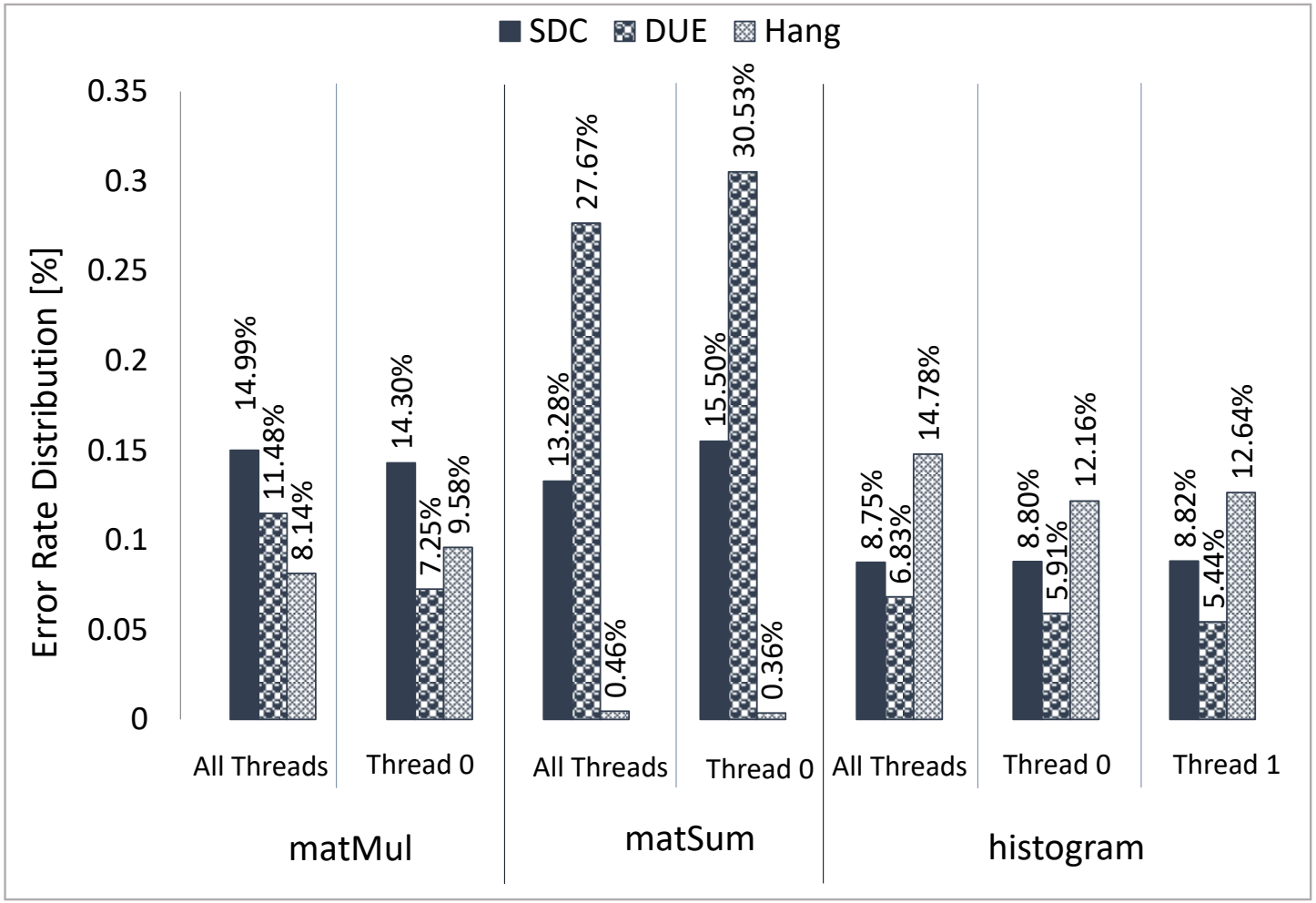
Error Classification

- **Silent Data Corruption (SDC)**
  - There is a mismatch(es) with the faulty-free run
- **Detected Unrecoverable Error (DUE)**
  - Kernel did not finish normally as DAPI function call returns an error
  - Host application is not able to copy the results from GPGPU device memory to host
- **Hang**
  - Execution plus the memory copy operations are not able to finish within a detection latency time
- **Masked**
  - Kernel finished normally and no error is observed in the output.

Experimental Results

Benchmark	Selected Thread	SASS instructions [#]	Kernel time [ns]	Faults [#]
matSum	All Threads	18	4,483	1,536
	Thread 0	24	4,489	832
matMul	All Threads	234	2,361,958	19,908
	Thread 0	456	2,529,229	14,144
histogram	All Threads	354	839,492	30,208
	Thread 0	762	849,401	22,976
	Thread 1	762	848,201	22,976

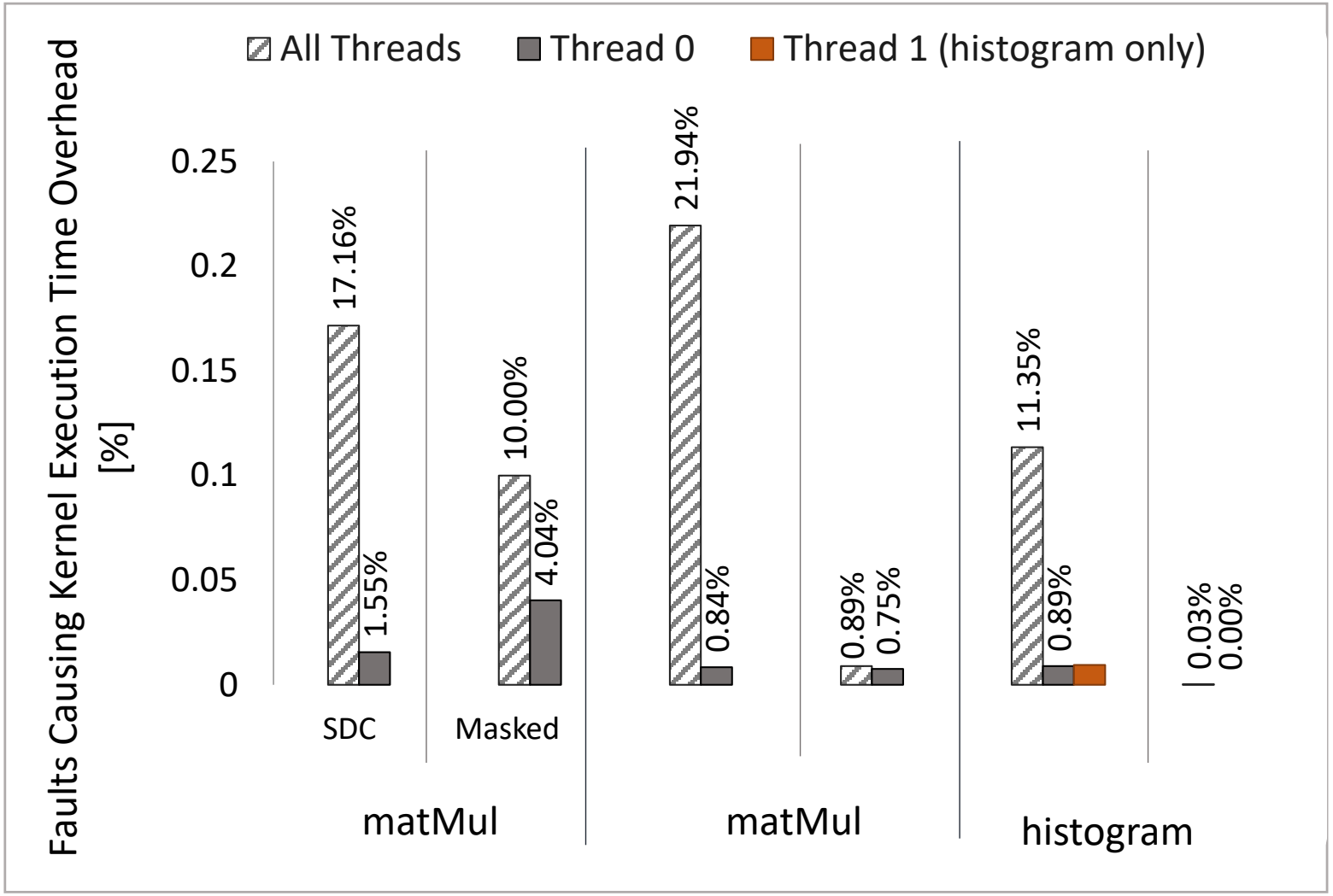
Error Rate Distribution



Silent Data Corruption

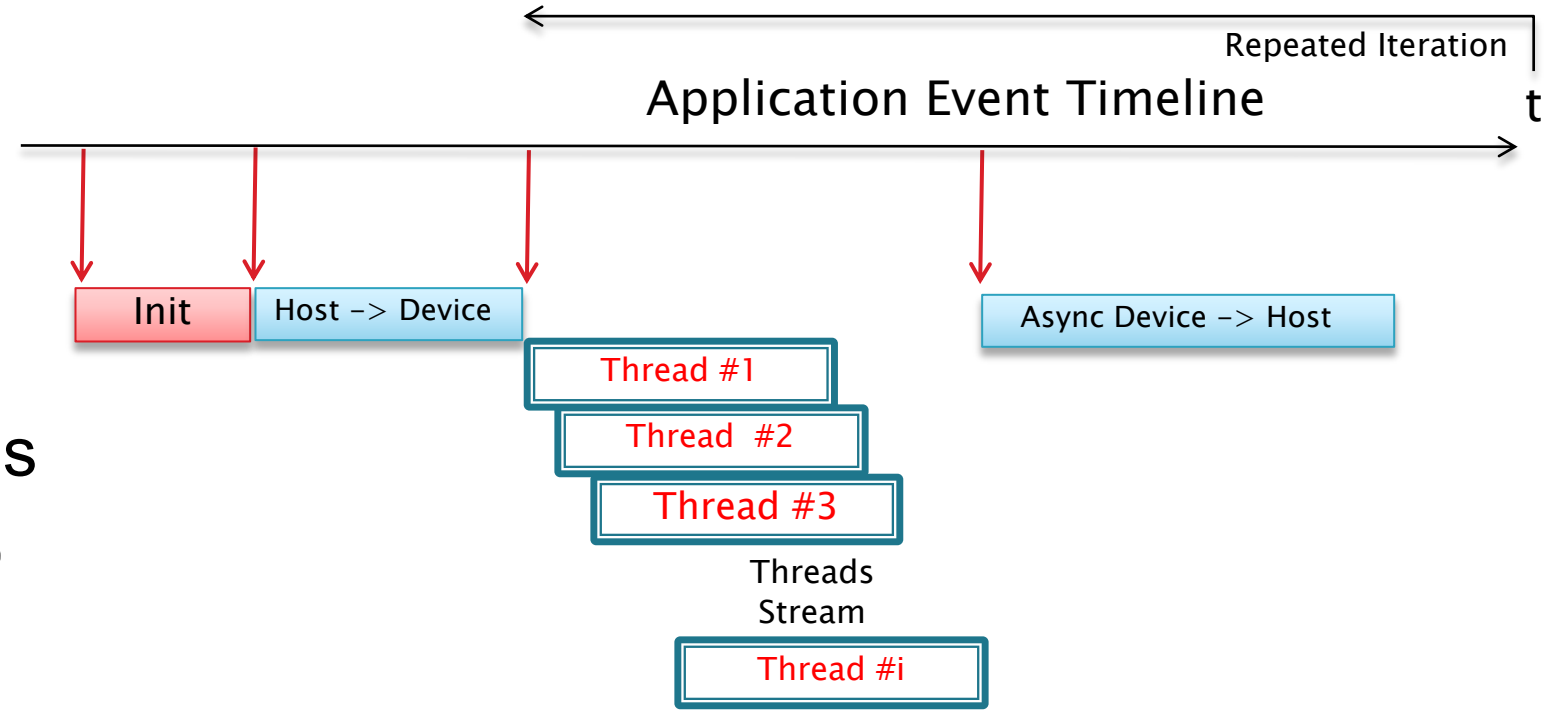
Benchmark	Selected Thread	One Error [%]	All Wrong [%]	Average Errors [#]
matSum	All Threads	0.00	61.76	126.53
	Thread 0	96.90	0.00	1.03
matMul	All Threads	0.00	73.35	254,414.31
	Thread 0	87.24	0.00	6.51
histogram	All Threads	0.53	53.33	181.99
	Thread 0	74.02	0.89	39.70
	Thread 1	72.77	0.89	39.50

Performance Comparison



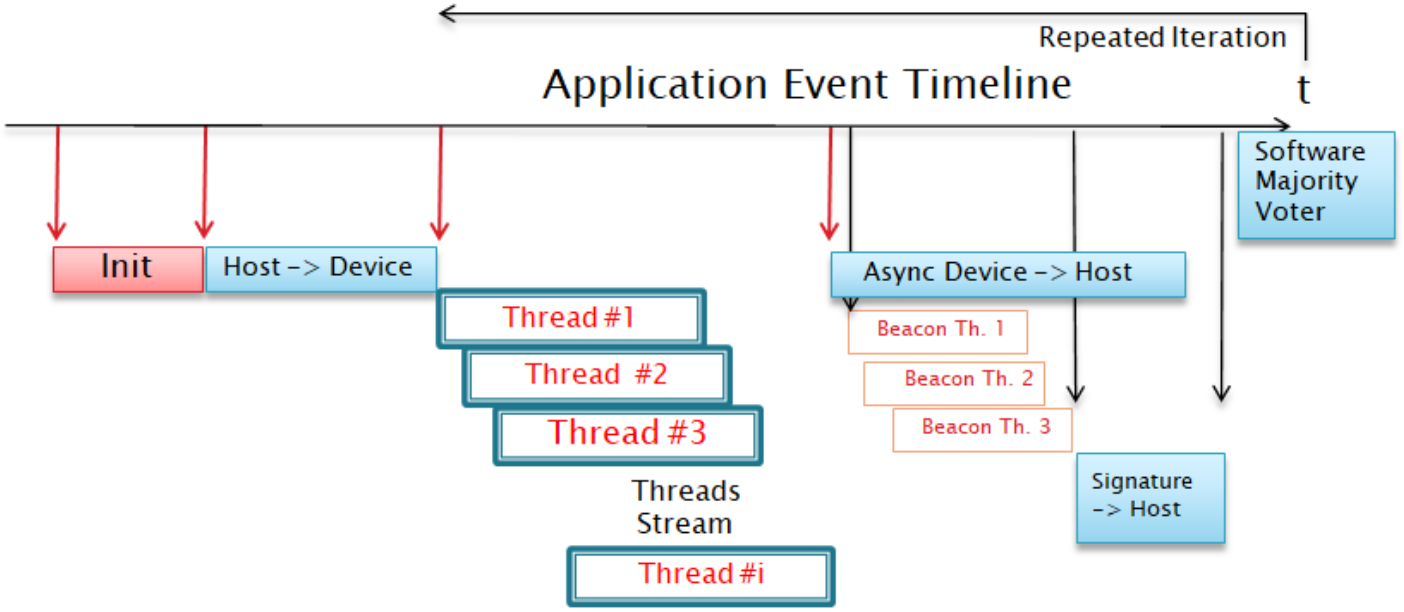
Soft Error Mitigation

- **Tunable according to the computational characteristics of a typical application running on embedded GPGPUs**
  - Initialization
  - synchronization with the external data sample
  - data stream host to GPU memory
  - running the kernel threads
  - data stream from GPU to host memory



Soft Error Mitigation

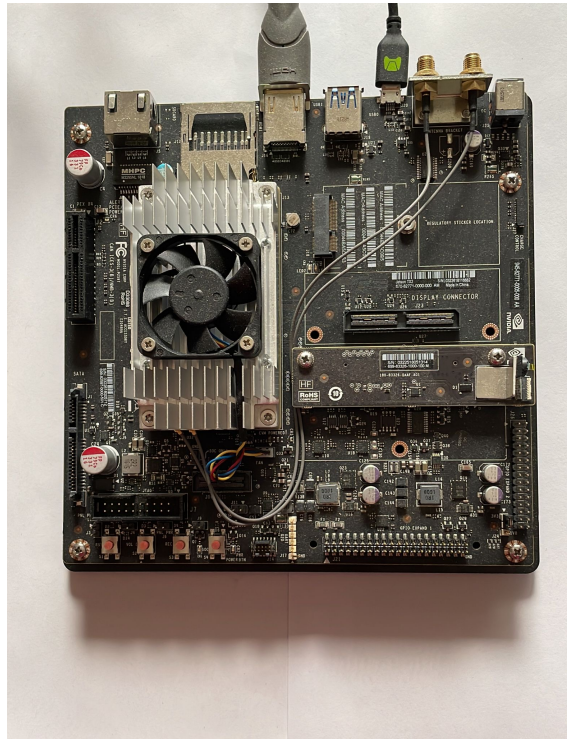
- **Adapted asynchronous memory copy functions from device to host**
- **Execution of error detection and comparison threads**
  - Simultaneous execution during data transfer
  - Periodical activation (Beacon Threads)



Experimental Results

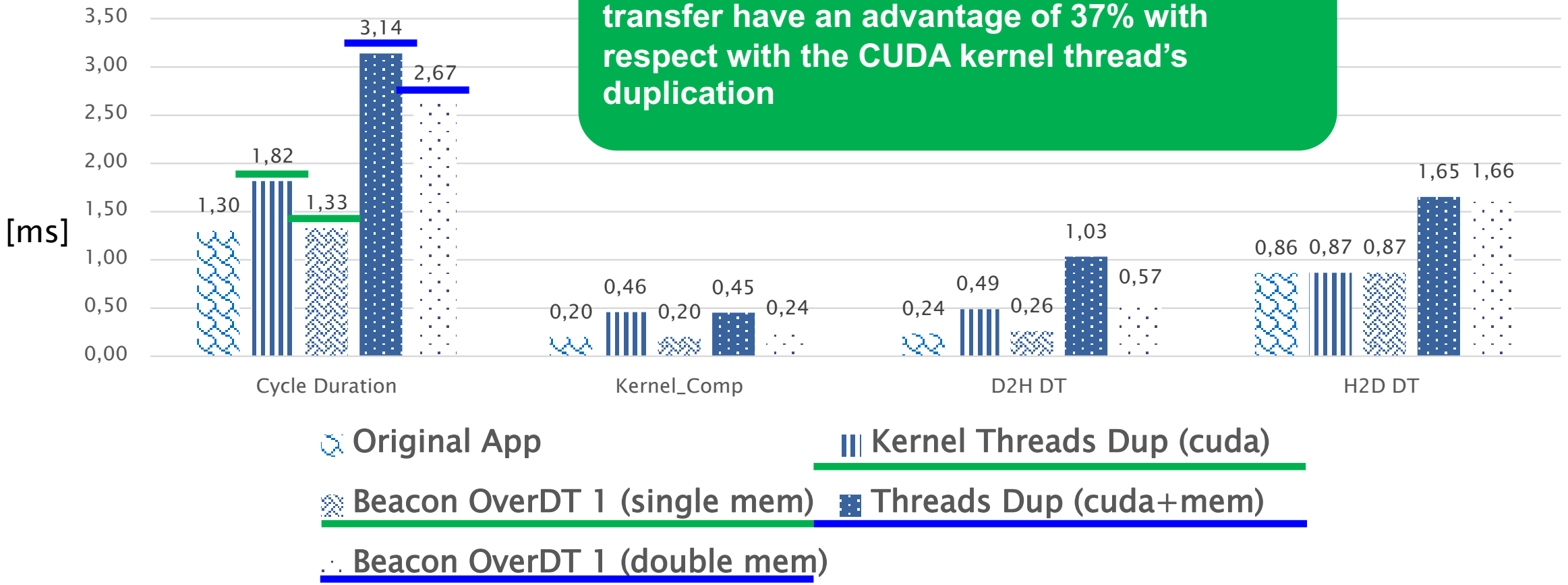
- We performed the test on the Jetson TX2 NVIDIA development board
- We used a data package of around 12MB
- Injection of 120K single bit flip

Benchmark	SDC [%]	Detected SDC [%]	Detection Latency [ms]
Original	96.7	-	-
Kernel Dup Single Mem	68.2	84.7	2.14
Beacon OverDT Single Mem	12.5	98.0	1.20
Duplicated Kernel and Mem	23.8	87.4	3.08
Beacon OverDT Duplicated Mem	4.7	98.8	1.08



Performance Comparison

The developed beacon threads over data transfer have an advantage of 37% with respect with the CUDA kernel thread's duplication



Conclusions

- **A new reliability analysis and mitigation approach for GPGPU is presented**
- **Analysis is based on a fault injection solution targeting the SASS instructions executed on real devices**
  - CUBINJ is able to target all threads, or some specific thread(s)
- **The developed beacon thread mitigation solution shows an improved resiliency of more than 37%**
  - Negligible performance degradation

Future works

- We are building SASS encoding maps to further exploit the possibility for instrumentation to support the fault injection environment
- Extend the fault injection analysis and mitigation on GPGPU clusters for HPC

Thank you

luca.sterpone@polito.it

Reliability analysis

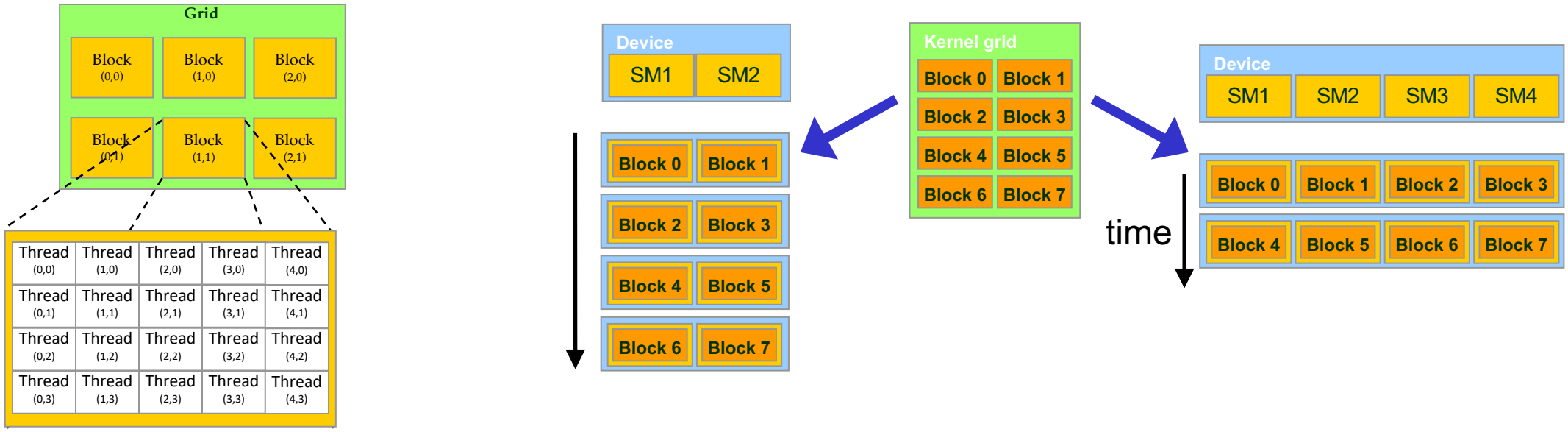
- **Radiation experiments have been performed using accelerated beam**
  - Focus on various aspects of the analysis and mitigations of soft errors in GPGPUs
- **Radiation beam hits the device indistinctively**
  - Besides, internal device implementation is not available

Reliability analysis – Architectural models

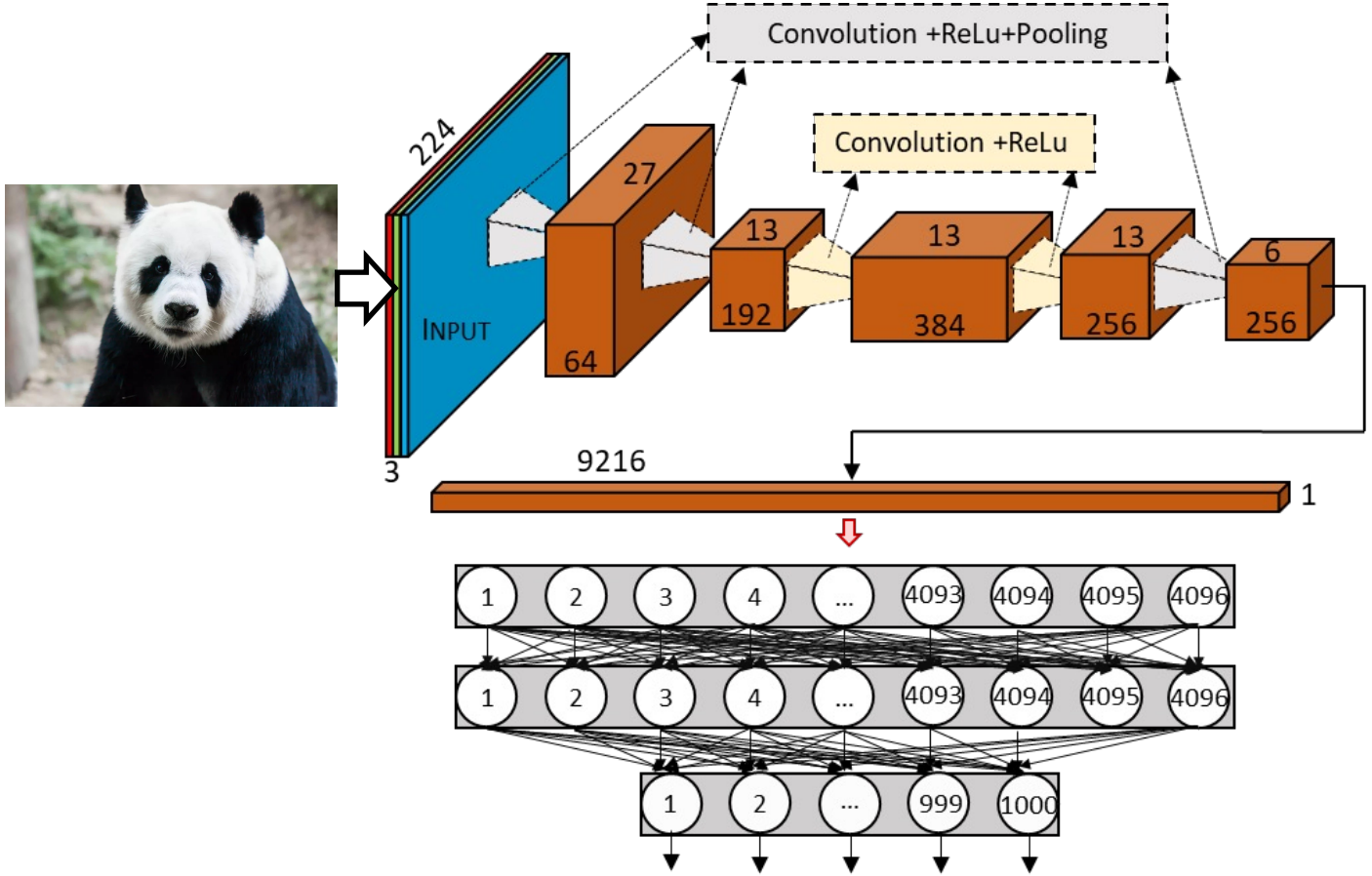
- **Development of architectural models in order to perform a more detailed analysis**
  - CUDA binary of streaming multiprocessor computation based on NVIDIA G80 - FlexGrip

Transparent Scalability

- The GPU runtime system can execute thread blocks in any order relative to each other
- This flexibility enables to execute the same application code on hardware with different numbers of SM



Convolutional Neural Networks



Panda	**79 %**
Cat	**11 %**
Clown	**6 %**
Penguin	**4 %**

NVBitFI

Benchmark	SASS instructions [#]	Kernel time [ns]
matSum	18	56,801
matMul	234	1,671,000
histogram	354	1,425,500