On the Functional Test of the GPGPU Scheduler

E. Rodriguez Condia, B. Du,
M. Sonza Reorda, L. Sterpone

RESCUE
European Training Network

Goal

- To propose methods for testing General Purpose GPU (GPGPU) during the operational phase ~~for possible permanent faults~~.

Introduction

- Acceleration cards for 2D graphics
- First GPU (Nvidia GeForce 256) defined as

  a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second.
- General Purpose GPU as GPU expends in other applications for high performance parallel computing capability

  AI, Image/Video Processing (VR/AR), Autonomous Driving …

The Reliability Issue

- Design Verification (Requirement, Specification, Standards)
- Manufacturer Test (post-silicon verification, burn-in test)
- In-Field Test (Power-On Self Test, BIST)
- Fault Tolerant Design
  - Physical: shielding
  - Hardware: resource replication
  - Software: temporal redundancy, check-point recovery

Reliability Evaluation of Embedded GPGPUs for Safety Critical Applications*

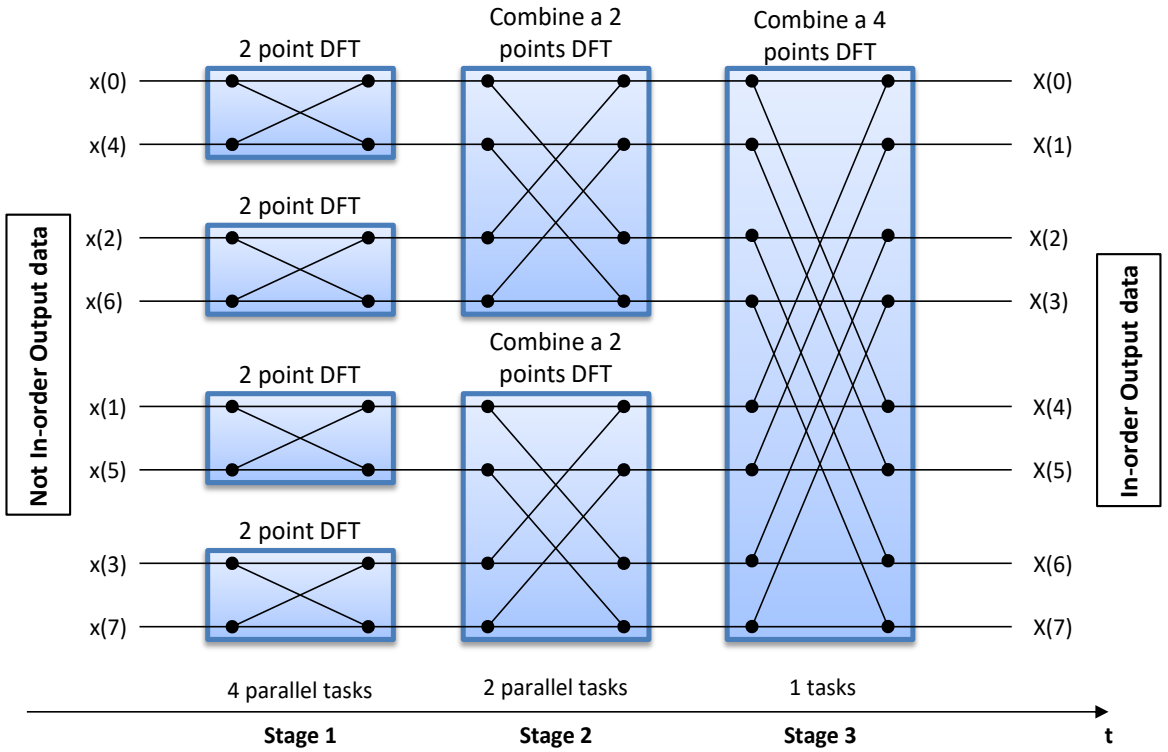
- **soft error sensitiveness** of a typical parallel algorithm, executed with different GPGPUs cache configurations.
- the same algorithm, executed with different threads distribution.
- Cooley-Tukey implementation for Fast Fourier Transform (FFT)

Test Program	Configuration
FFT_32	2 thread blocks, 32 threads per block, block scheduler not activated, thread scheduler not activated
FFT_64	2 thread blocks, 64 threads per block, block scheduler not activated, thread scheduler **activated**
FFT_64_NOL1	Same as FFT_64, L1 cache disabled

*D. Sabena, L. Sterpone, L. Carro and P. Rech, "Reliability Evaluation of Embedded GPGPUs for Safety Critical Applications," in *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3123-3129, Dec. 2014.

Cooley-Tukey FFT Implementation

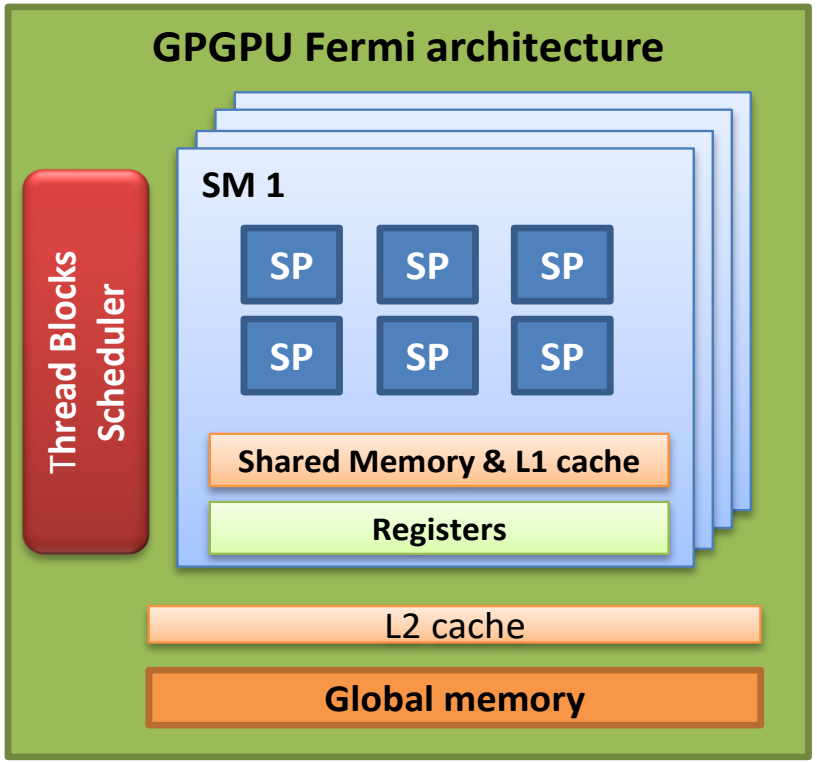
- common FFT implementation for embedded applications
- allows to reduce algorithm complexity from $O(N^2)$ to $O(N*\log_2 N)$

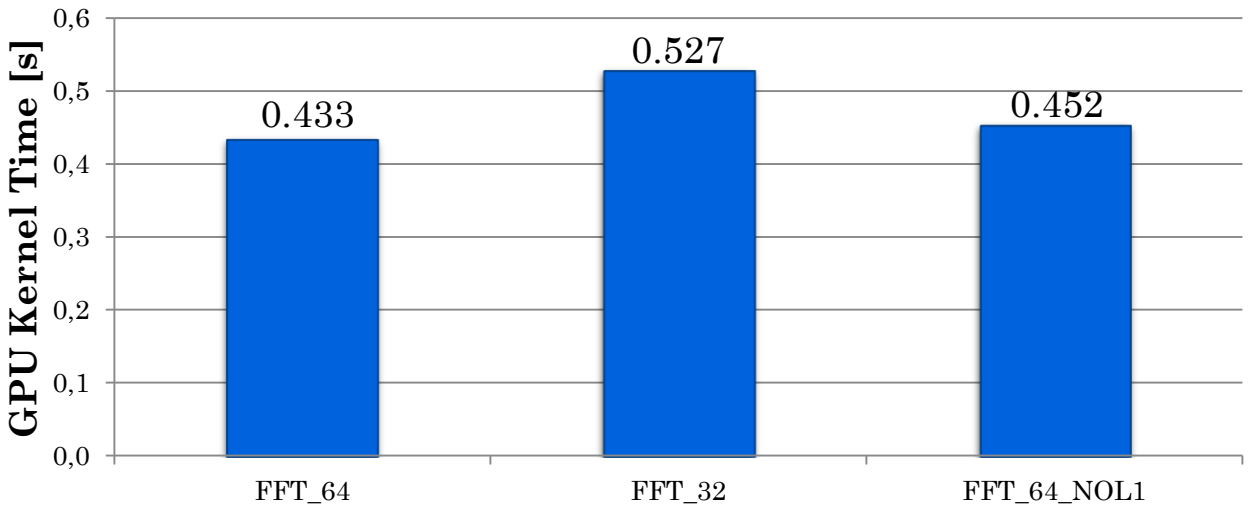


- At each stage a FFT *butterfly unit* combines the results of two smaller *Discrete Fourier Transforms* (DFTs) into a large DFT

Experimental Setup

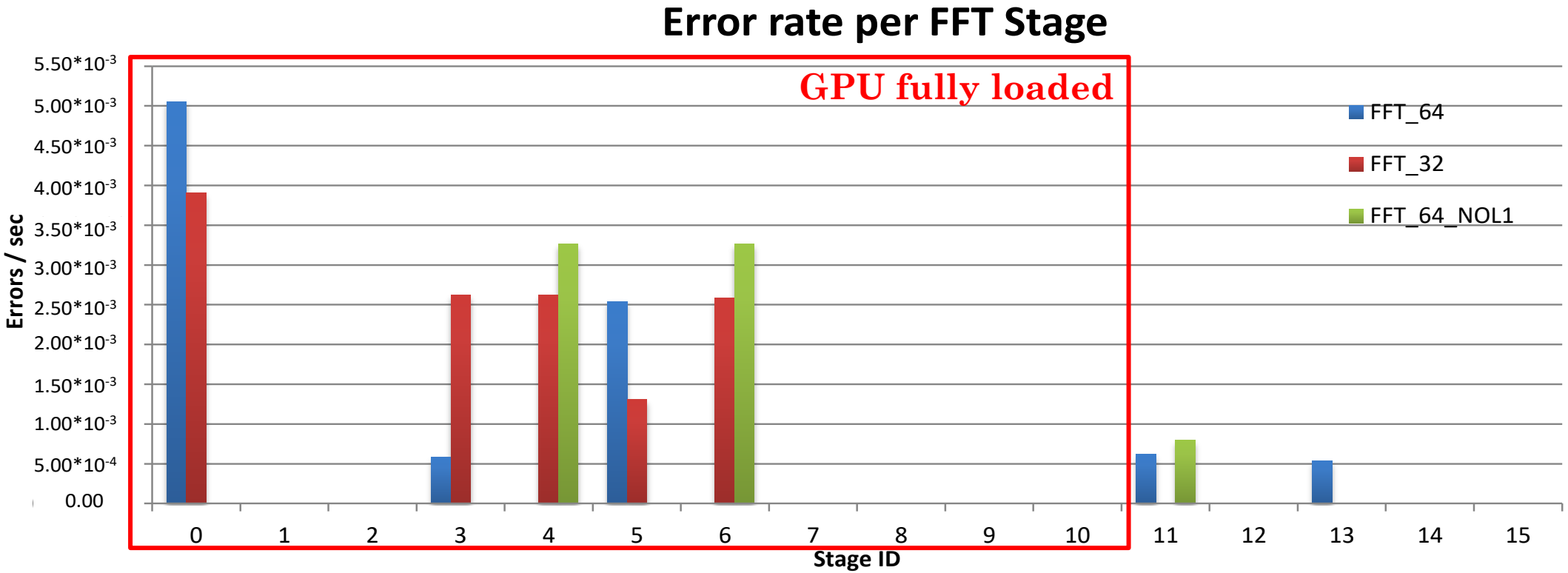
- DUT: CARMA DevKit features a Qseven NVIDIA Tegra 3 with a Quad-core ARM A9 CPU and the NVIDIA Quadro® 1000M GPGPU with 2 SM of 48 SP each for a total of 96 CUDA cores





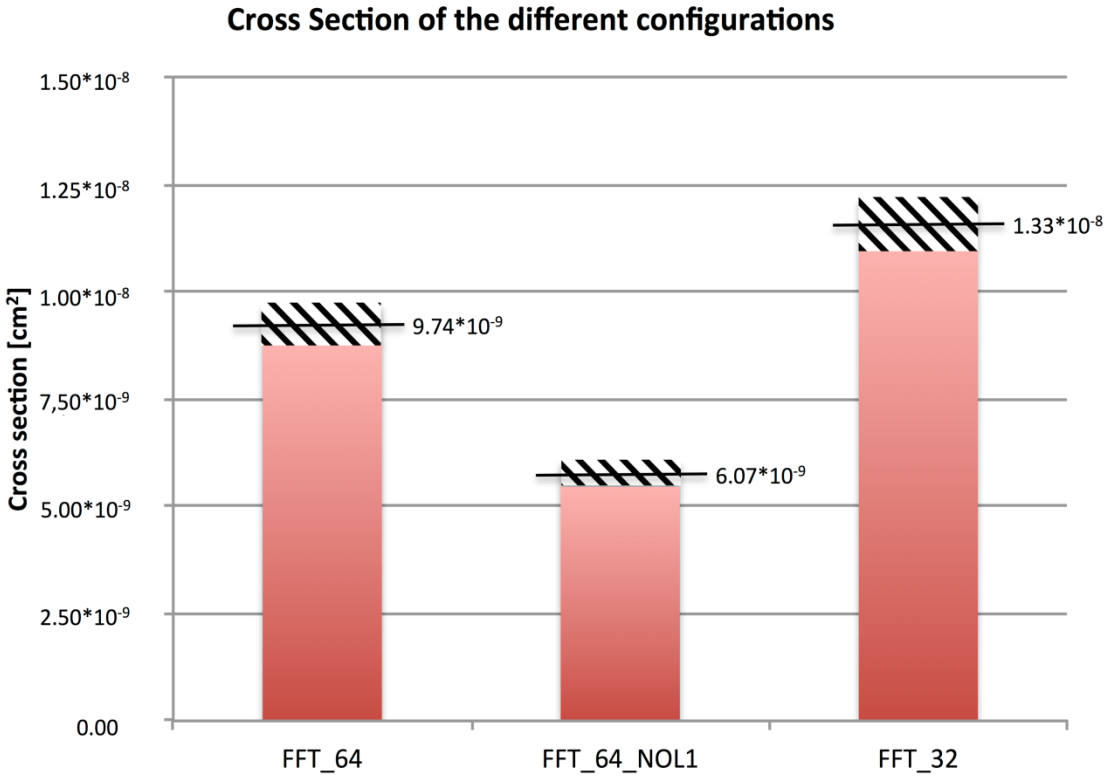
Experimental Results

- VESUVIO neutron facility at ISIS, Rutherford Appleton Laboratories (RAL), Didcot, UK
  - The available flux was of about $3.89 \cdot 10^4$ n/(cm^2·s)
  - at room temperature
  - beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra



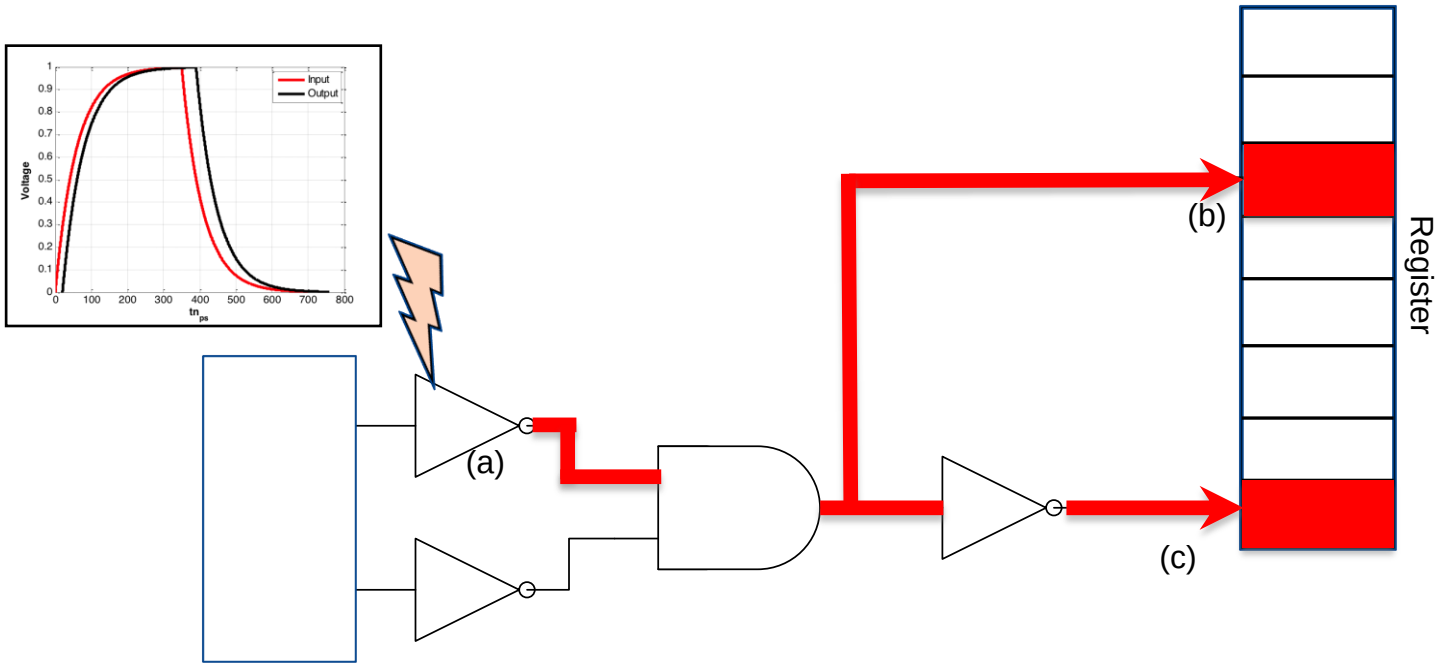
Experimental Results

- The cross section was measured dividing the number of observed error per second by the average neutron flux
- disabling the L1 caches reduces the cross section of the algorithm of 38%, while the performance overhead is not so relevant (about 4.5%)
- FFT_32 has a 25% higher cross section than FFT_64.
  In FF_64, the number of instantiated thread (i.e. 64) imposes the thread scheduler to continuously swap the active threads, causing data in the L1 cache refresh frequently.



A New Simulation-based Fault Injection Approach for the Evaluation of Transient Errors in GPGPUs *

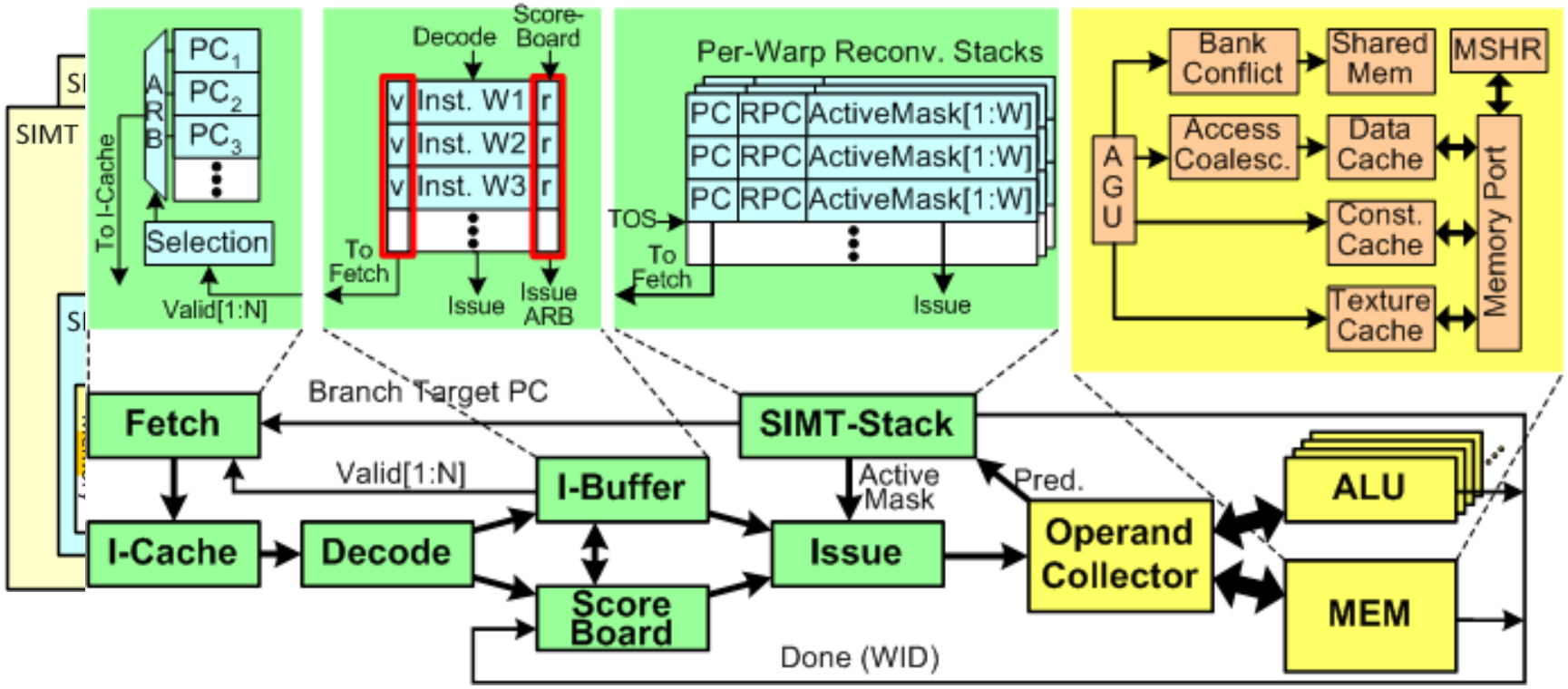
- Develop **a transient error simulation tool** for the early evaluation of transient errors in applications on GPGPU



*S. Azimi, B. Du, and L. Sterpone. "A New Simulation-Based Fault Injection Approach for the Evaluation of Transient Errors in GPGPUs." International Conference on Architecture of Computing Systems. Springer, Cham, 2016.

GPGPU-Sim*

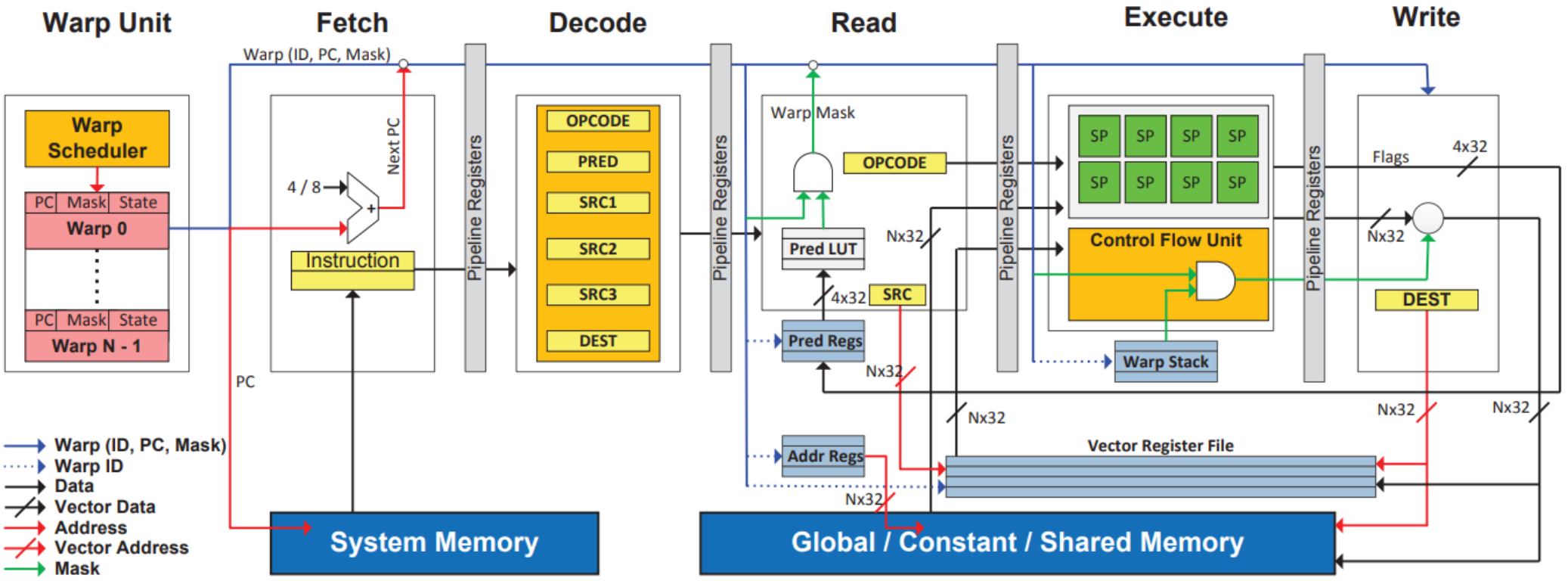
- a cycle-level GPU performance simulator that focuses on "GPU computing"



*A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, 2009.

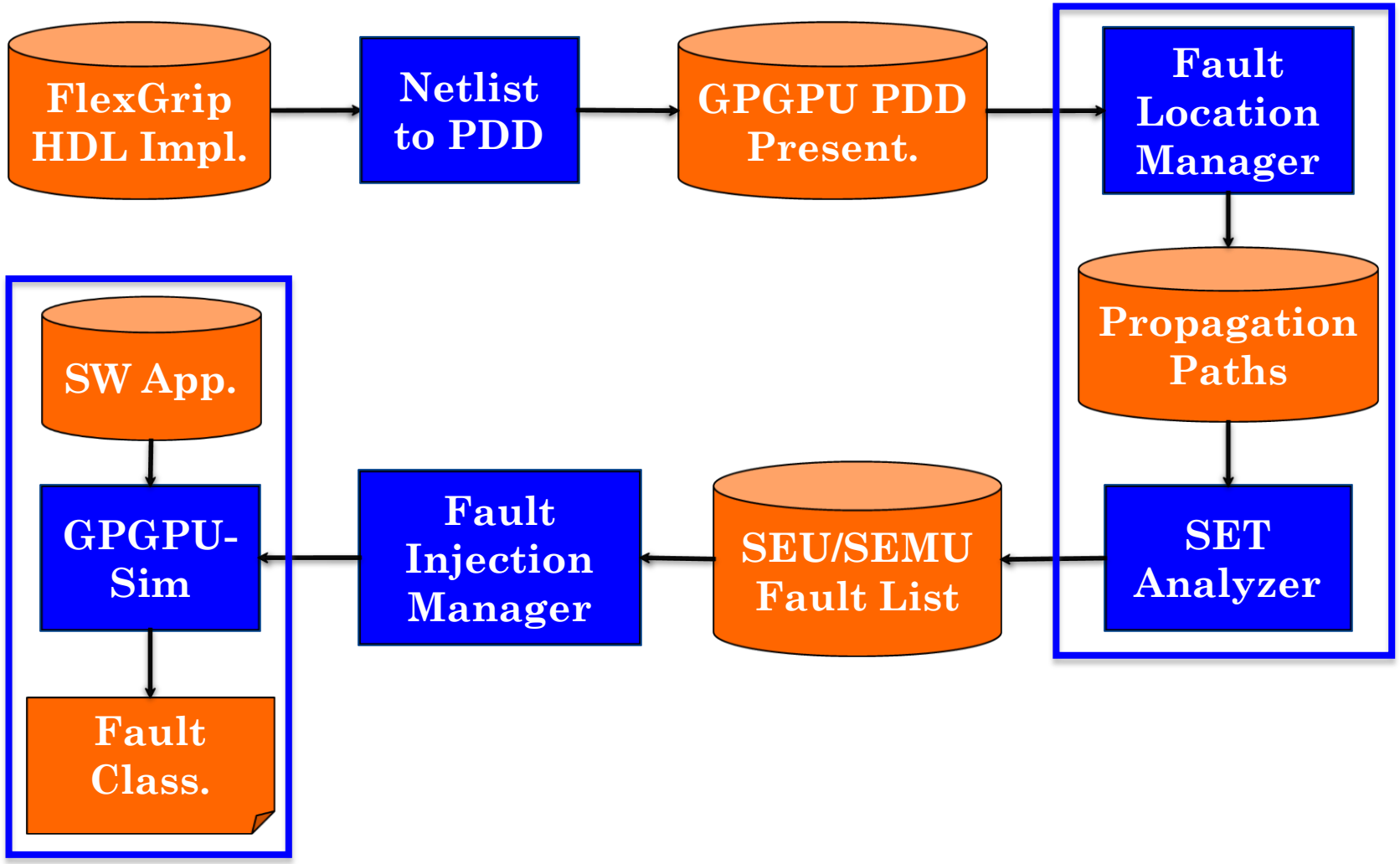
FlexGrip*: A soft GPGPU for FPGAs

- HDL implementation based on Nvidia G80 architecture



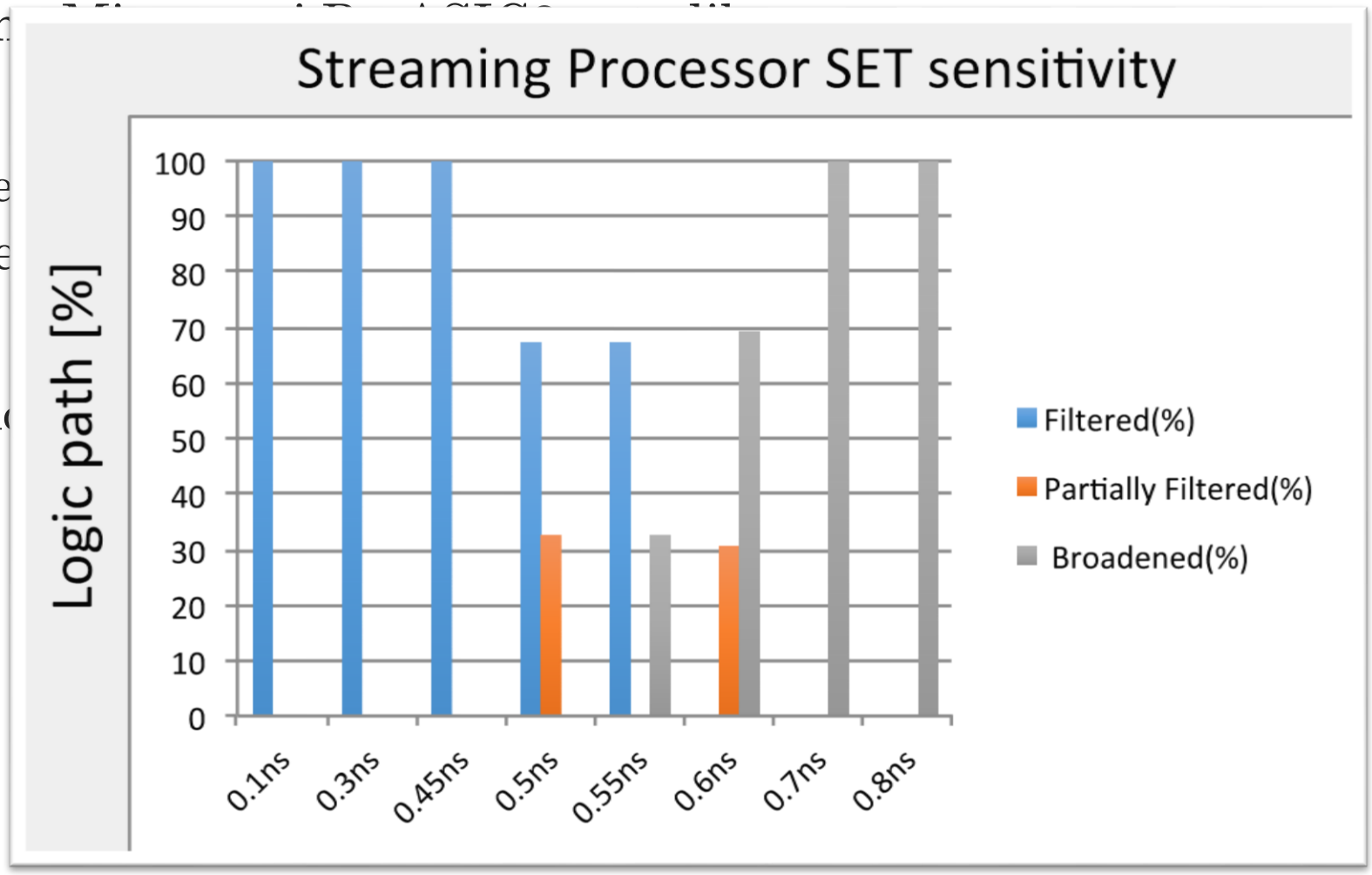
* K. Andryc, M. Merchant and R. Tessier, "FlexGrip: A soft GPGPU for FPGAs," 2013 International Conference on Field-Programmable Technology (FPT), Kyoto, 2013, pp. 230-237.

Proposed Flow

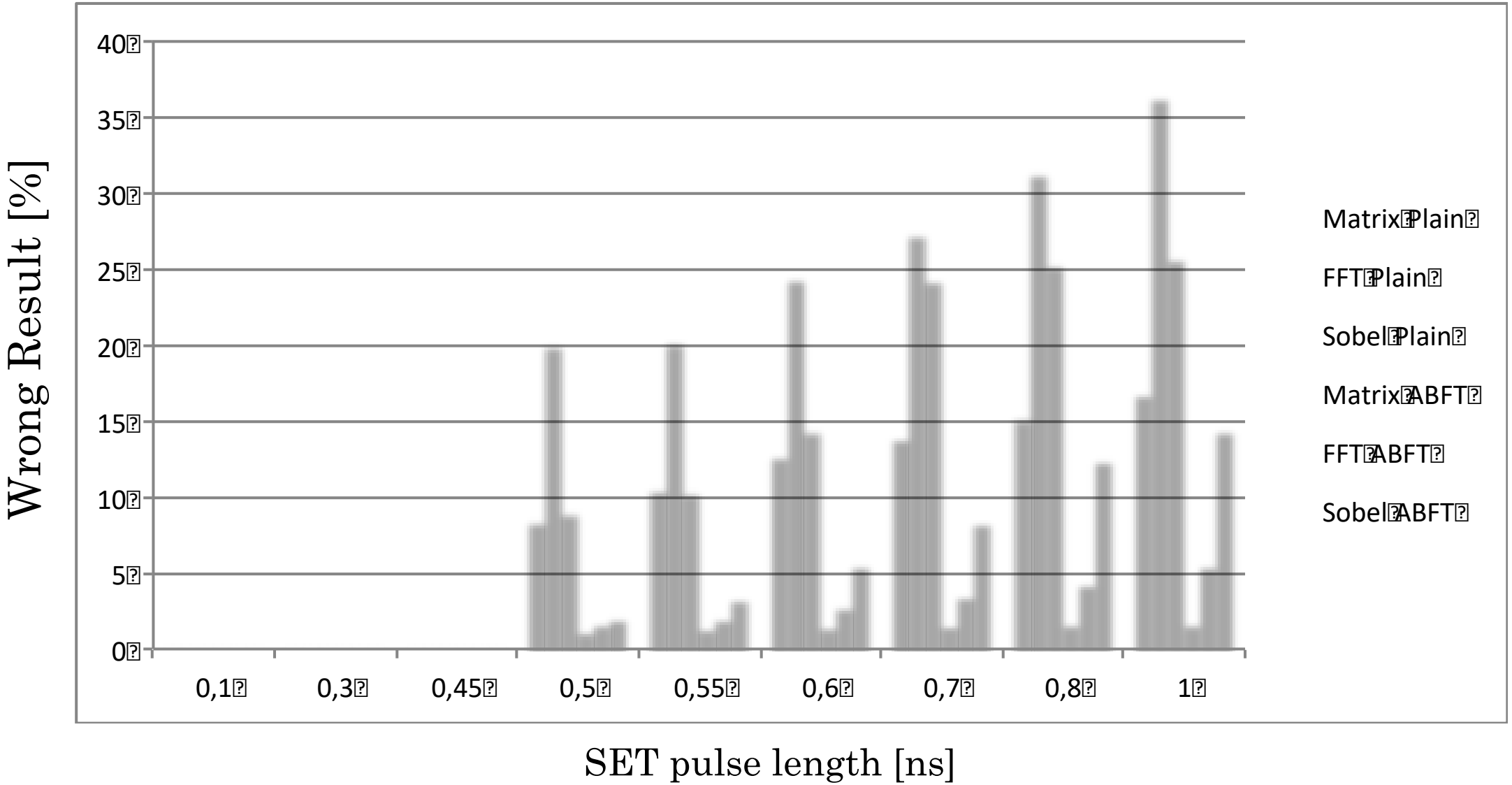


Experimental Setup

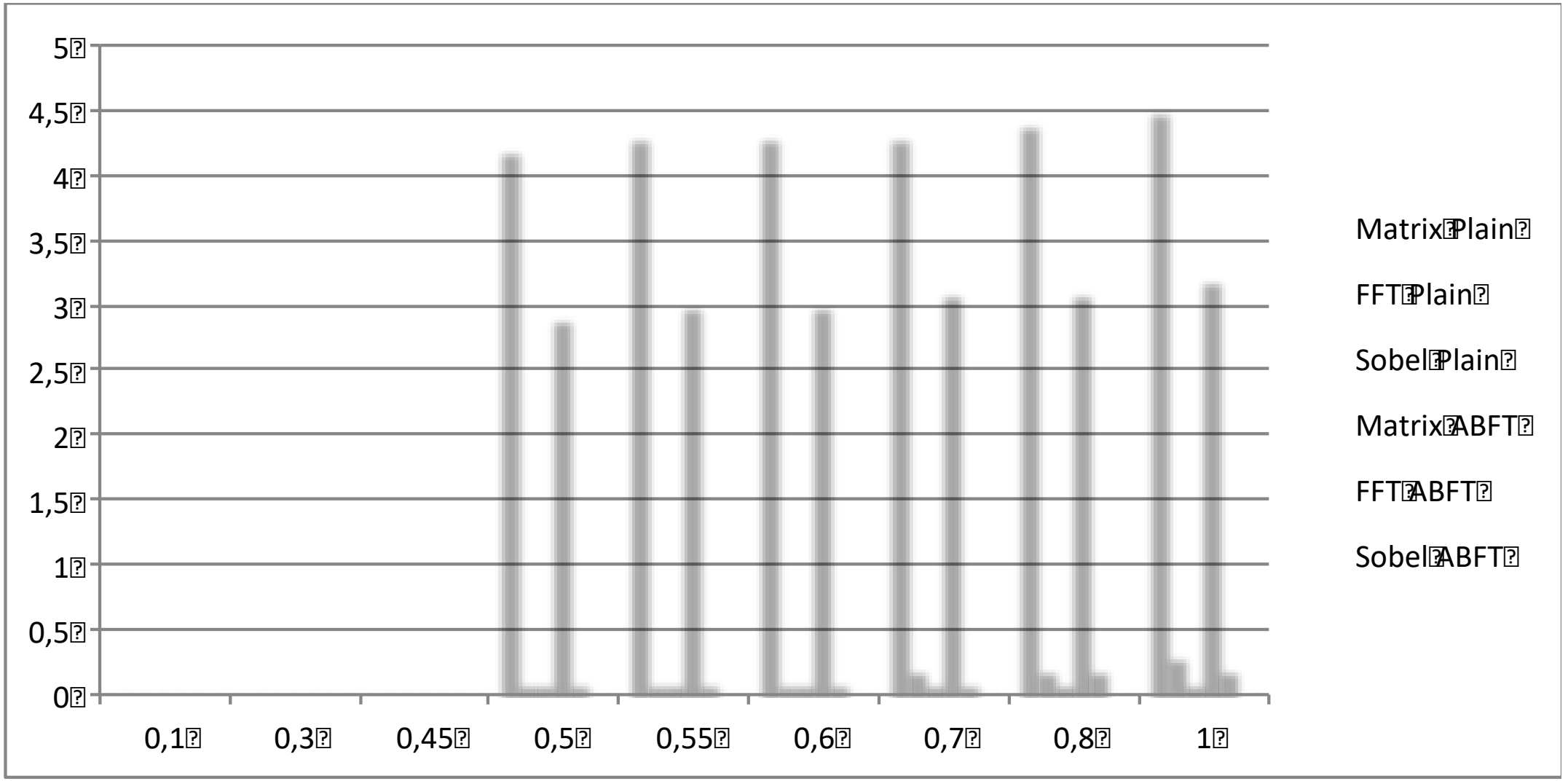
- Synthesized SM architecture
  - ~50K gates, ~1.5M logical paths
  - Implemented usin
- SET analysis
  - Eight different type
  - 1,000 random inje
- Applications (input
  - Matrix multiplicati
  - FFT
  - Sobel filter



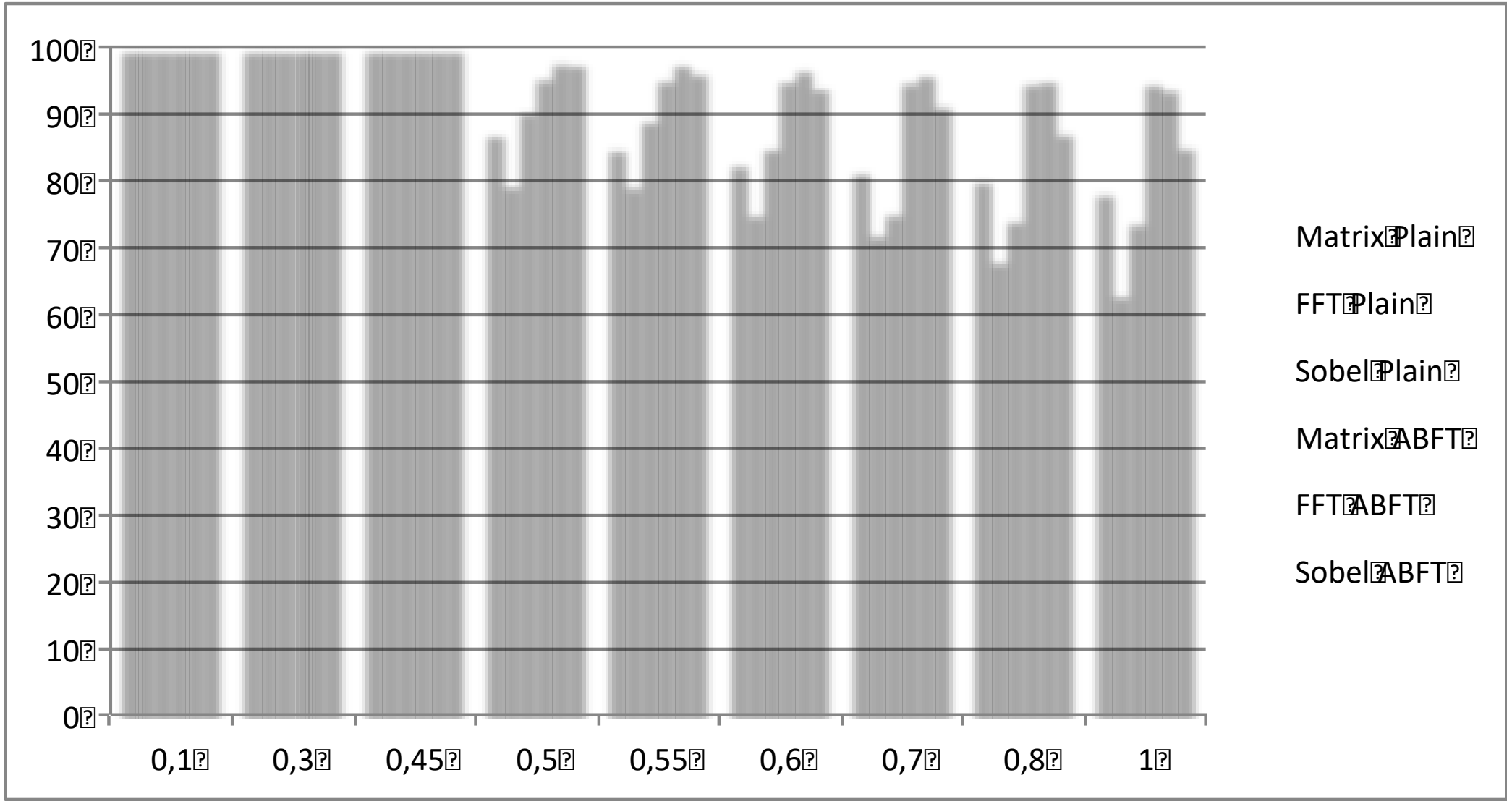
Experimental Result



Experimental Result



Experimental Result



In-Field GPGPU Test with Software-Based Self-Test Techniques*

- Analyze the effects of permanent faults in the GPGPU operation

* B. Du, Josie E. Rodriguez Condia, M. Sonza Reorda, L. Sterpone, accepted in IOLTS2018

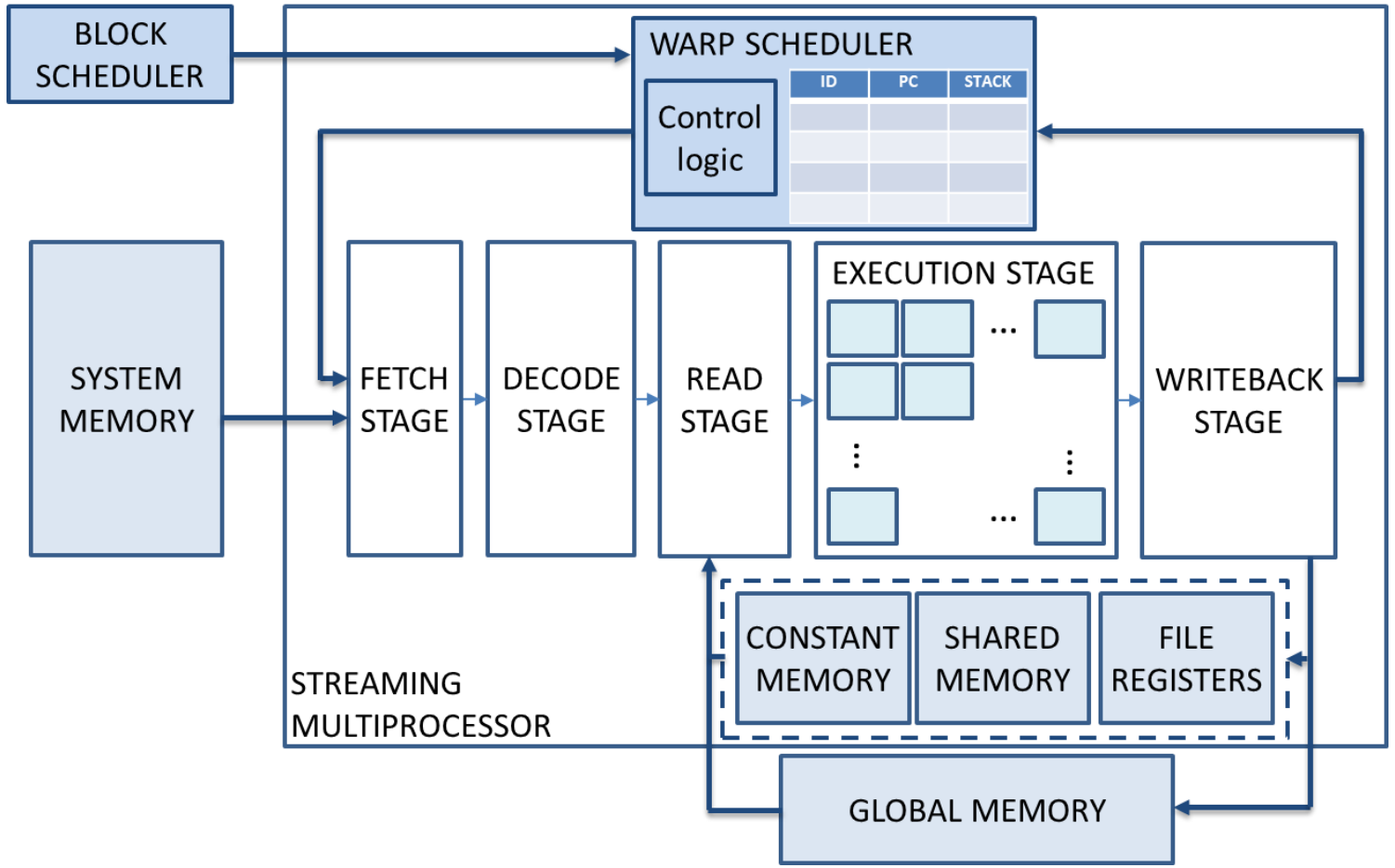
In-Field GPGPU Test with Software-Based Self-Test Techniques*

- Develop effective SBST techniques for testing permanent faults
  - based on architectural or structural description of modules (**Self-test program**)
  - at-speed and in-field
  - non-intrusive
  - flexibility
  - cost

* B. Du, Josie E. Rodriguez Condia, M. Sonza Reorda, L. Sterpone, accepted in IOLTS2018

FlexGrip*: A soft GPGPU for FPGAs

- HDL implementation based on Nvidia G80 architecture

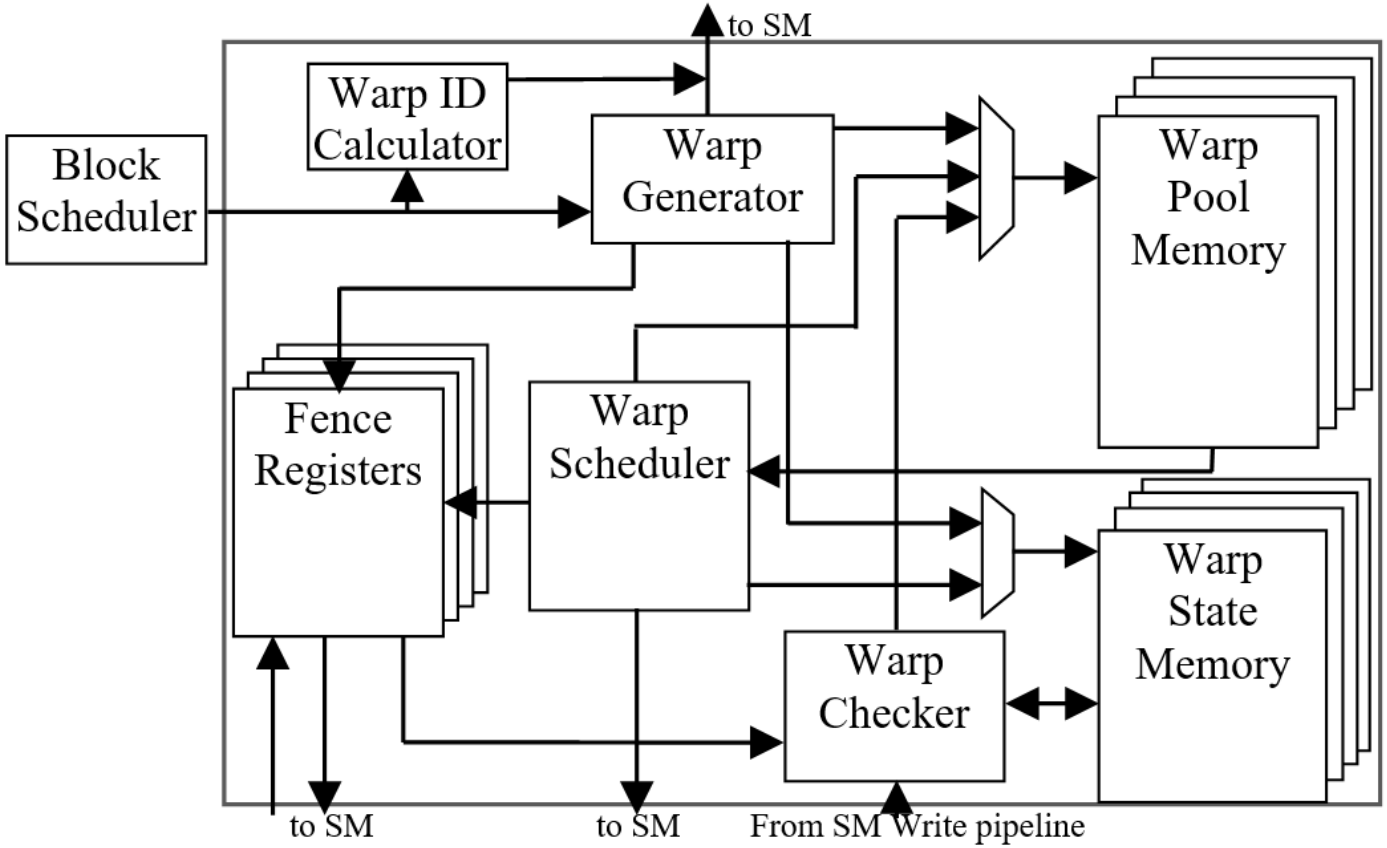


Opcode	Description
I2I	Copy integer value to integer with conversion
IMUL/ IMUL32/ IMUL32I	Integer multiply
SHL	Shift left
IADD	Integer addition between two registers
GLD	Load from global memory
R2A	Move register to address register
R2G	Store to shared memory
BAR	Barrier synchronization
SHR	Shift right
BRA	Conditional branch
ISET	Integer conditional set
MOV/ MOV32	Move register to register
RET	Conditional return form kernel
MOV R, S[]	Load from shared memory
IADD, S[], R	Integer addition between shared memory and register
GST	Store to global memory
AND C[], R	Logical AND
IMAD/ IMAD32	Integer multiply-add; all register operands
SSY	Set synchronization point; used before potentially divergent instructions
IADDI	Integer addition with an immediate operand
NOP	No operation
@P	Predicated execution
MVI	Move immediate to destination
XOR	Logical XOR
IMADI/ MAD32I	Integer multiply-add with an immediate operand
LLD	Load from local memory
LST	Store to local memory
A2R	Move address register to data register

* K. Andryc, M. Merchant and R. Tessier, "FlexGrip: A soft GPGPU for FPGAs," 2013 International Conference on Field-Programmable Technology (FPT), Kyoto, 2013, pp. 230-237.

Warp Unit (Thread Scheduler) in FlexGrip

- Execution units in the pipeline are similar to normal processor and well addressed in literatures for testing such as Adder, Multiplier etc.
- In charge of wrap and thread scheduling, synchronization



Proposed Methods

- M1 based on creating divergence paths among threads and detect by checking timing information
  - targets the permanent faults affecting the ID field of the warp pool line
  - faults affecting the bits in warp PC by extra comparisons
- M2 inserts extra global memory access instructions in divergence paths and detect by checking timing information and/or global memory checking
  - slower global memory access could enlarge timing difference for easing fault detection effort
  - faults causing “stuck” thread could be easily detected by memory checking
- M3 writes thread signature to global memory and detect by memory checking only
  - does not depend on availability of performance monitor resources

Proposed Methods

`j ← 0`	▶ **Initialize divergence control var.**
…	▶ Normal app. Execution
`Sig_per_thread[] ← 0`	▶ Initialize signature (M3)
`for i ϵ {set of ThreadId in SM} do`	▶ Evaluate for every ThreadID
`if i == j then`	▶ If ThreadID Matches
`Divergence_path_GroupA();`	▶ Divergence path Group A
`NOP`	▶ Not operation instruction
`Thread_Store_in_memory();`	▶ Memory results store (M2)
`Sig_per_thread[i] ← Sig_per_thread[i]+1`	▶ Set signature (M3)
`Sig_store_in_memory();`	▶ Store signature (M3)
`else`	
`Divergence_path_GroupB ();`	▶ Divergence path Group B
`j←j+1`	▶ Move to next ThreadID

Experimental Setup

- FlexGrip
  - Removed FPGA (Xilinx Virtex6) library dependency
  - Replaced with generic VHDL implementations
  - Synthesized with OpenCell library using Synopsys Design Compiler
  - RegisterFile/Memory components are kept back as behavioral model
  - Post-Synthesis netlist is extracted
- Fault injection campaign
  - targets on warp pool memory lines (2048 sites) and interface connection with other modules (478 sites)
  - stuck-at faults
  - test program based on VectorAdd sample program
  - FlexGrip configured to with one grid, 256 blocks and 24 threads per wrap

Experimental Results

	VectorAdd	M1	M2	M3
Total Faults	2,048	2,048	2,048	2,048
Testable Faults	984	984	984	984
Detected Faults	624	728	984	984
Hang	440	613	616	616
Memory Mismatch	184	115	112	368
Performance Degradation	0	0	256	0
Testable Fault Coverage (%)	63.41	73.98	100	100
Fault Coverage (%)	30.46	35.54	48.04	48.04

Conclusions

- The key idea is to generate divergence paths of thread execution and use performance variation among the threads and/or memory write of signature in global memory for testing and detecting permanent faults in thread scheduler.
- M3 is effective even with only final memory checking.
- SBST method is applicable for testing GPGPU and could achieve high fault coverage.
- Other modules? Multiple Streaming Multiprocessors? Block Scheduler?
- Other faults? Transient faults? Online testing?
- Mitigation Techniques?

Ongoing Activities

- FlexGrip Modifications
  - Extension of currently supported instructions
  - Extension of functionalities (e.g. Multiple SMP)
- SBST targeting on other modules
- Embedded GPGPU
  - Timing, performance, reliability
  - Nvidia Jetson
  - HLS on Xilinx FPGA, OpenCL support
- GPGPU acceleration of testing (analysis) automation

Thank You For Your Attention!

- Questions?

- email: boyang.du@polito.it