Enabling Software Resilience in GPGPU Applications via Partial Thread Protection

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

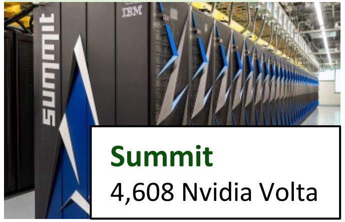
William & Mary

WILLIAM & MARY

CHARTERED 1693

GPUs & Soft Errors

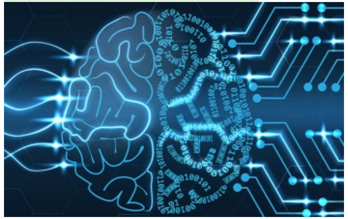
Supercomputing



Summit
4,608 Nvidia Volta

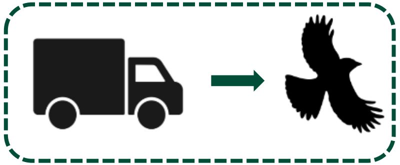
Selene
2,240 Nvidia Ampere

Deep learning



Self-driving cars

SDC → Misclassification



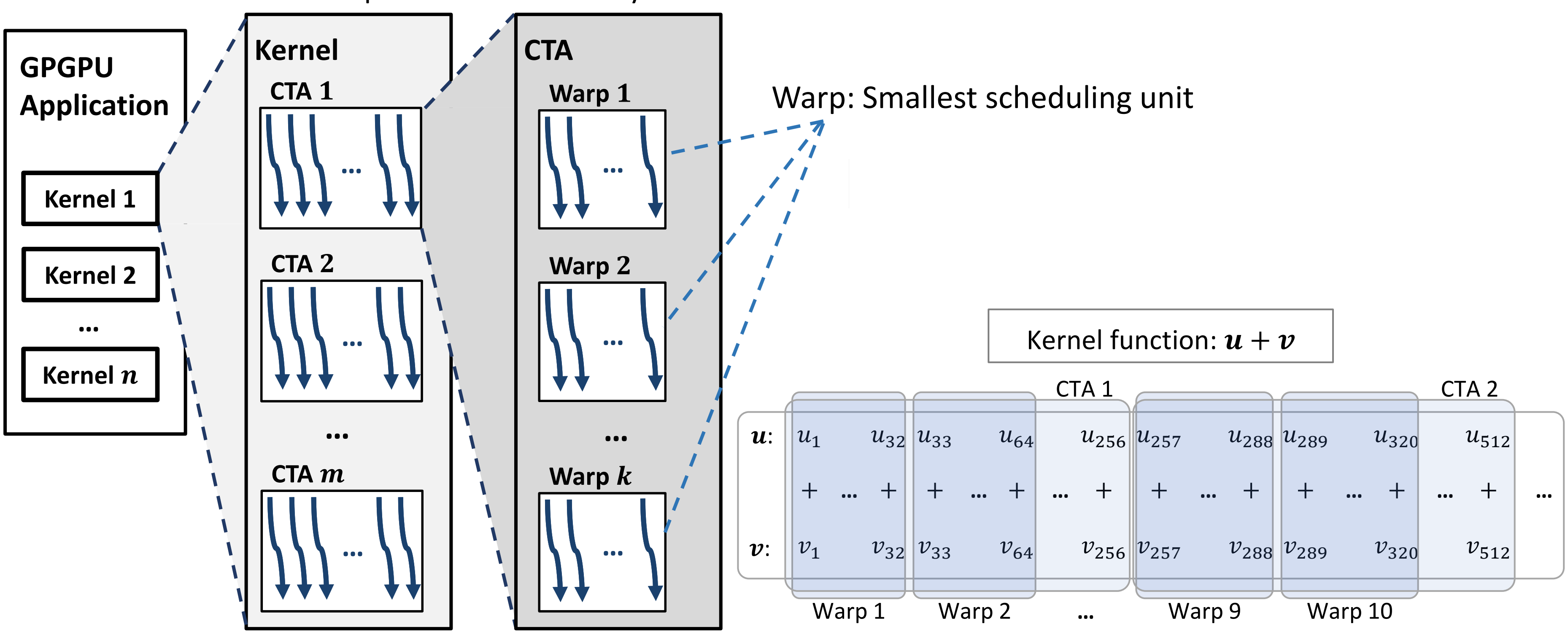
- GPUs are commonly deployed
- GPUs are prone to **soft errors**
  - High-energy radioactive particles (i.e., cosmic rays) cause **bit flips**
  - Commonly observed
  - Impact on long-running applications can be tremendous
    - ***Masked*** output: Correct
    - ***Crash, hang***, …
    - ***Silent Data Corruption (SDC)*** output: Incorrect
  - SDCs in critical applications can be dangerous
- Protection:
  - Error correction code (ECC) 5%~40% overhead
  - Software solution: re-computation
    - Detection: Duplication x**2** Computation
    - Correction: Triplication x**3** Computation

RQ: How to protect GPGPU applications **selectively**?

*Li, Guanpeng, et al. "Understanding error propagation in deep learning neural network (DNN) accelerators and applications." SC 2017.

GPGPU Application Parallelization

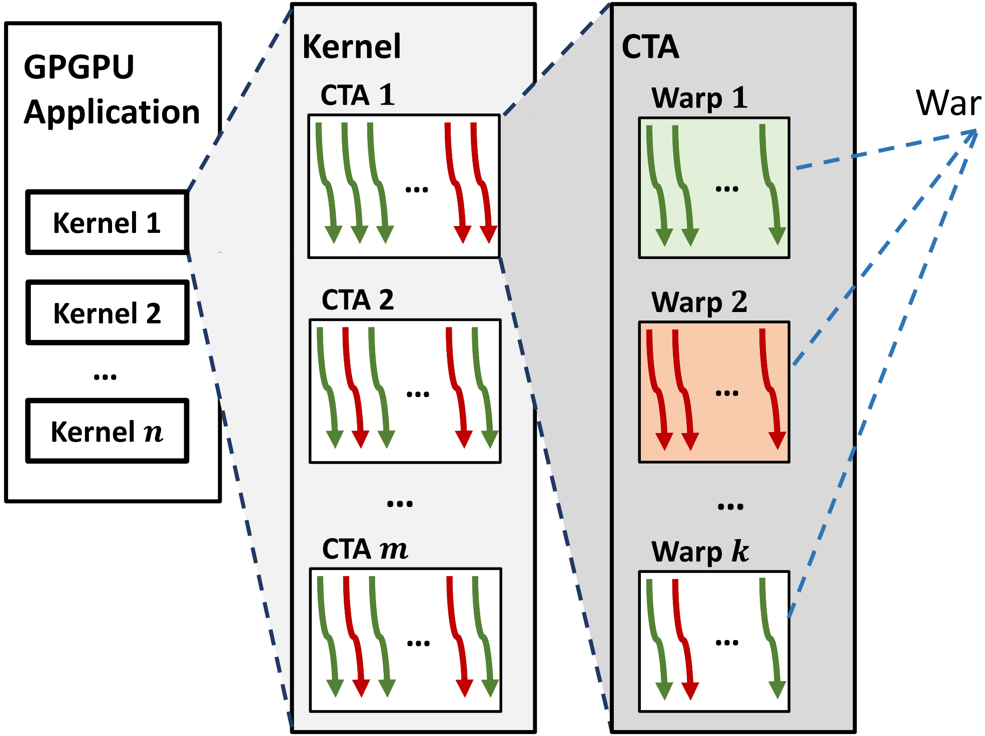
*CTA: Cooperative Thread Array


GPGPU Application
Kernel 1
Kernel 2
...
Kernel n
Kernel
CTA 1
CTA 2
...
CTA m
CTA
Warp 1
Warp 2
...
Warp k
Kernel function: $\boldsymbol{u}+\boldsymbol{v}$
CTA 1
CTA 2
$\boldsymbol{u}$: u_1 u_{32} u_{33} u_{64} u_{256} u_{257} u_{288} u_{289} u_{320} u_{512}
$+$ ... $+$ $+$ ... $+$ ... $+$ $+$ ... $+$ $+$ ... $+$ ... $+$ ...
$\boldsymbol{v}$: v_1 v_{32} v_{33} v_{64} v_{256} v_{257} v_{288} v_{289} v_{320} v_{512}
Warp 1
Warp 2
...
Warp 9
Warp 10

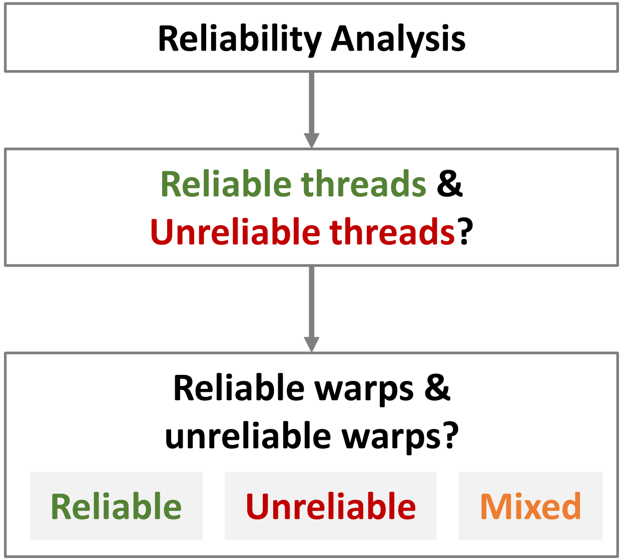
Warp: Smallest scheduling unit

GPGPU Application Parallelization

*CTA: Cooperative Thread Array

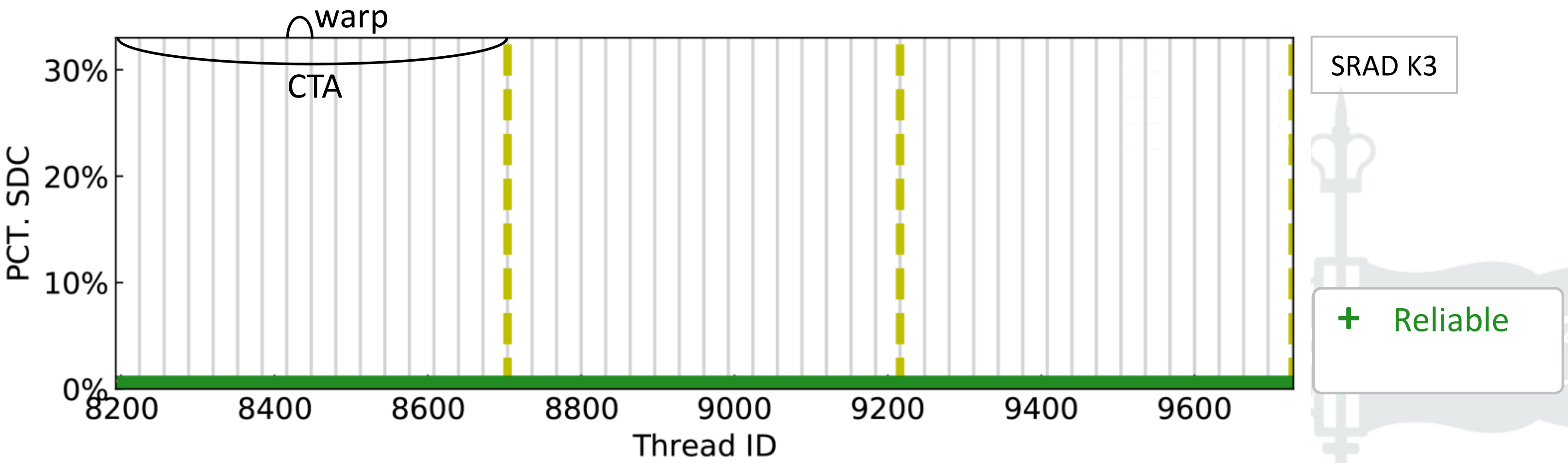


Warp: Smallest scheduling unit



Characterization

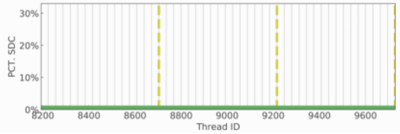
- All threads are **reliable**



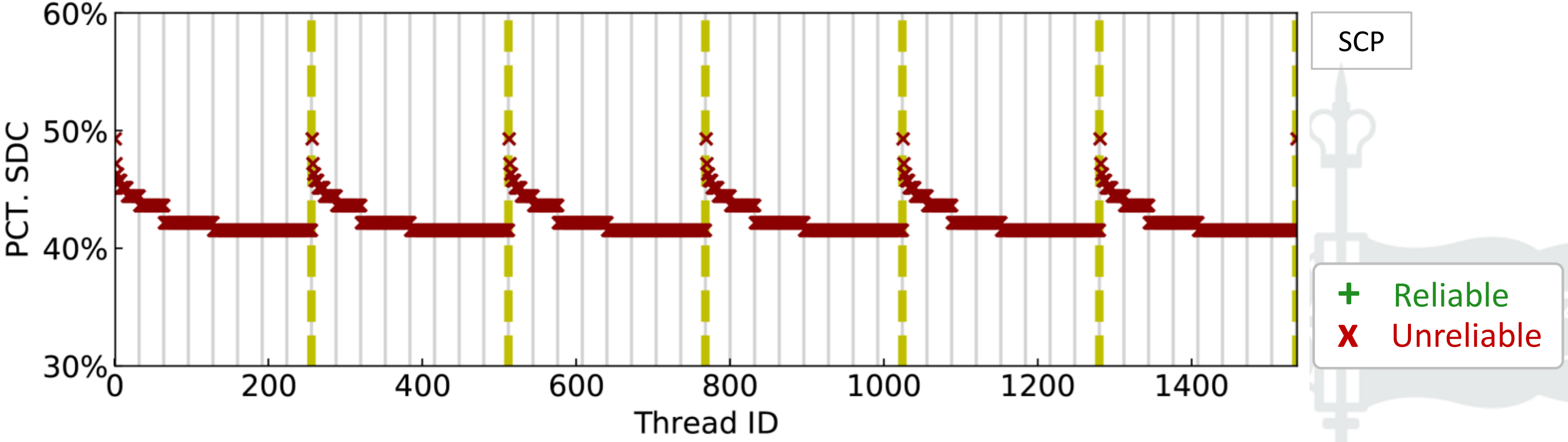
Representative benchmarks: *SRAD K3~K4, NeuralNetwork K1~K4*

Characterization

- ❖ All threads are **reliable** *(SRAD K3~K4, NeuralNetwork K1~K4)*



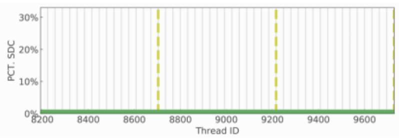
- ➢ All threads are **unreliable**



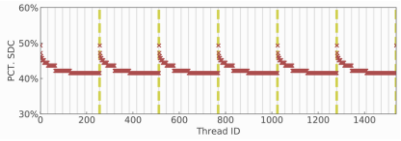
Representative benchmarks: *SCP, MVT*

Characterization

- ❖ All threads are **reliable**
  (*SRAD K3~K4, NeuralNetwork K1~K4)*

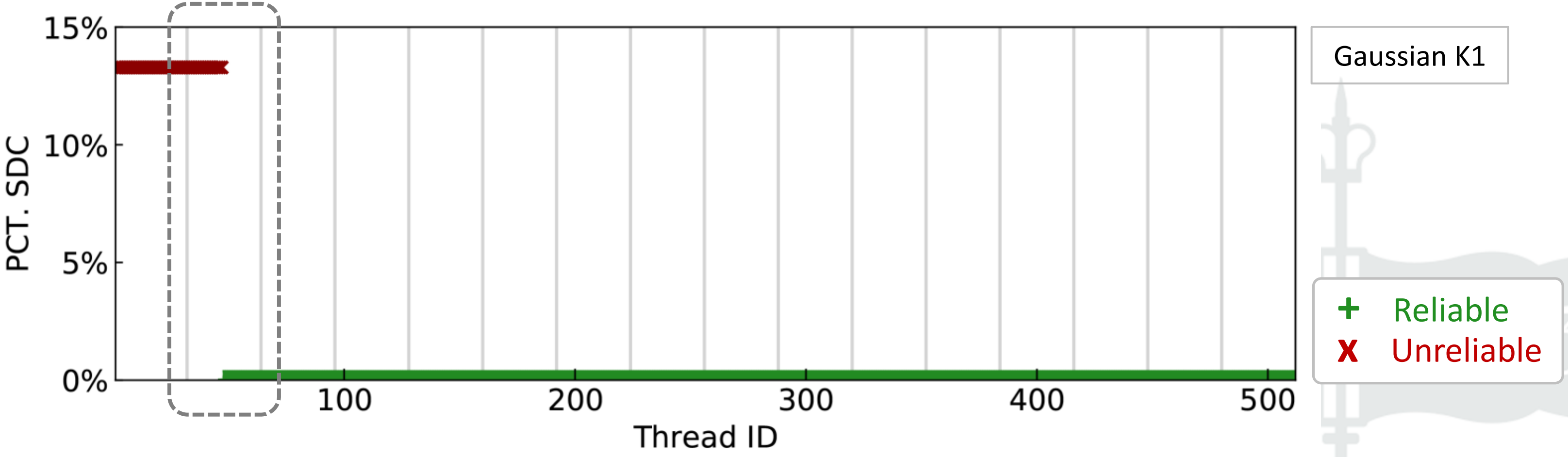


- ❖ All threads are **unreliable**
  (*SCP, MVT)*



➢ **Mixed** warps: reliable threads + unreliable threads

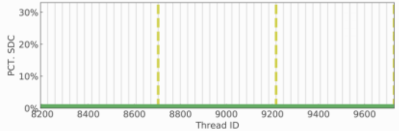
- Well-organized



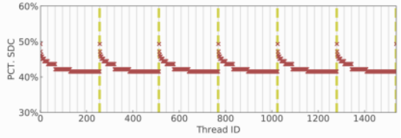
Representative benchmarks: *Gaussian K1, NearestNeighbor*

Characterization

❖ All threads are **reliable**
(SRAD K3~K4, NeuralNetwork K1~K4)

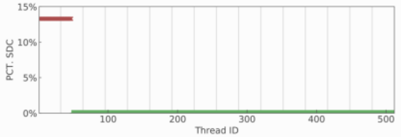


❖ All threads are **unreliable**
(SCP, MVT)

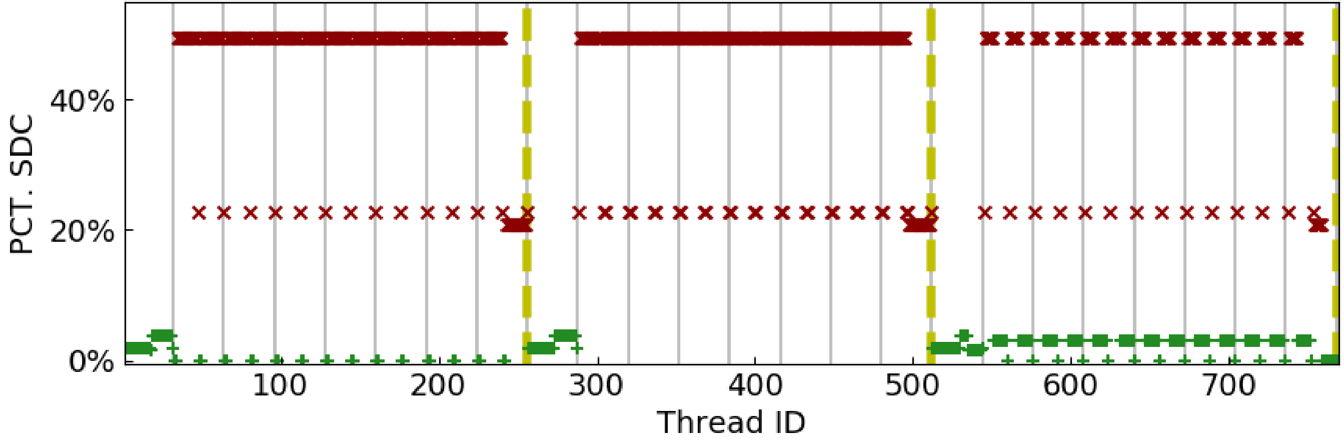


➢ **Mixed** warps: reliable threads + unreliable threads

❖ Well-organized
(Gaussian K1, NearestNeighbor)



- Badly-organized (Need remapping)



Hotspot
(First 768 threads / 9216 in total)

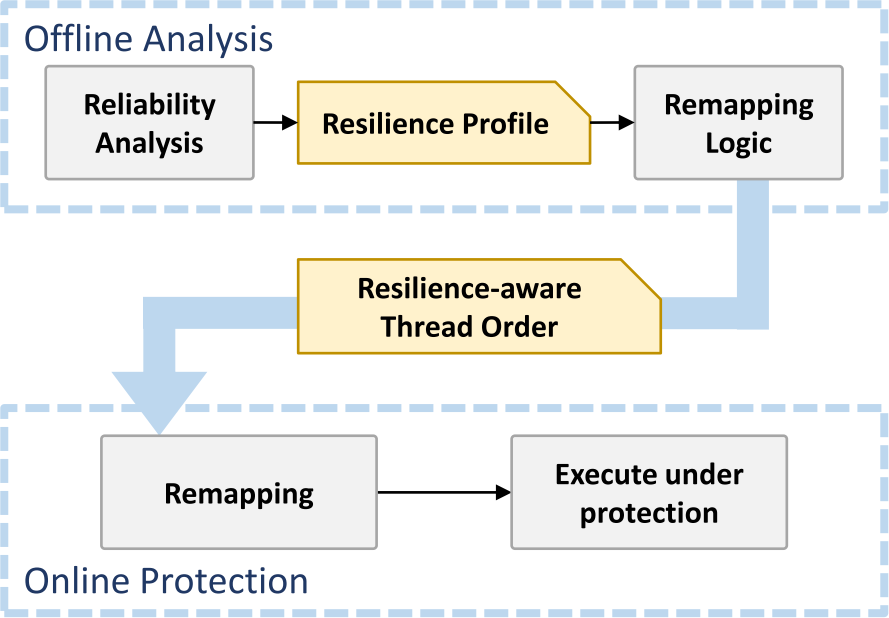
Representative benchmarks: *Gaussian K2, PathFinder, MeanFilter, Laplacian, 2DCONV, HotSpot, Jmeint*

Resilient Software Protection via Remapping

🟩 Reliable

🟥 Unreliable

🟧 Replication



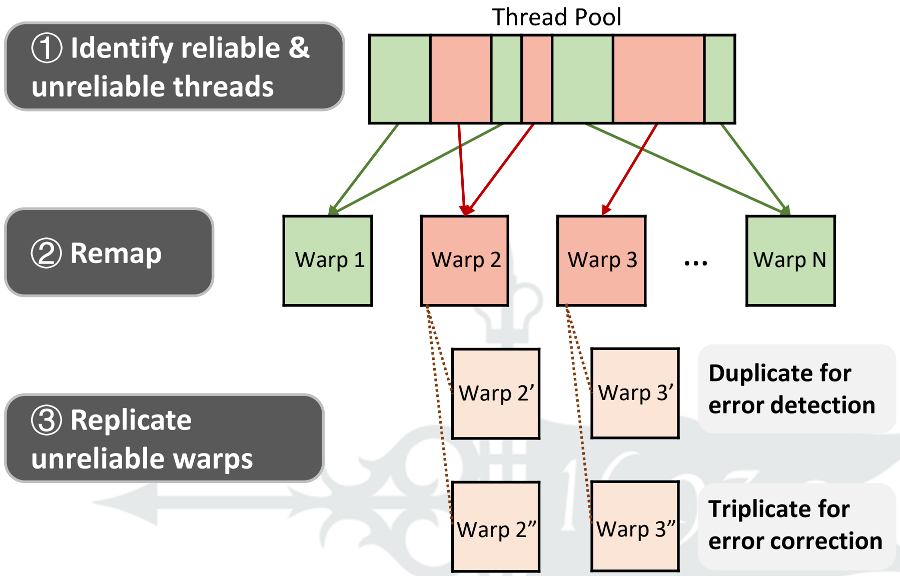
Offline Analysis

Reliability Analysis → Resilience Profile → Remapping Logic

Resilience-aware Thread Order

Remapping → Execute under protection

Online Protection



Thread Pool

① Identify reliable & unreliable threads

② Remap

Warp 1 Warp 2 Warp 3 … Warp N

③ Replicate unreliable warps

Warp 2’ Warp 3’

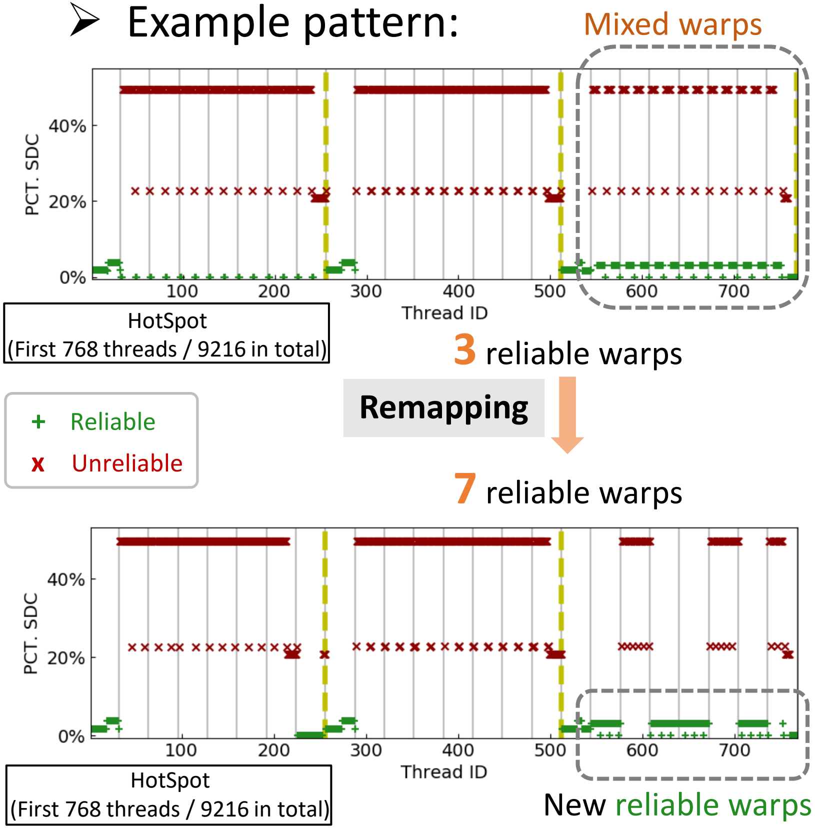
Duplicate for error detection

Warp 2” Warp 3”

Triplicate for error correction

Evaluation: Effectiveness

➢ Example pattern:



Mixed warps

HotSpot
(First 768 threads / 9216 in total)

3 reliable warps

Remapping

7 reliable warps

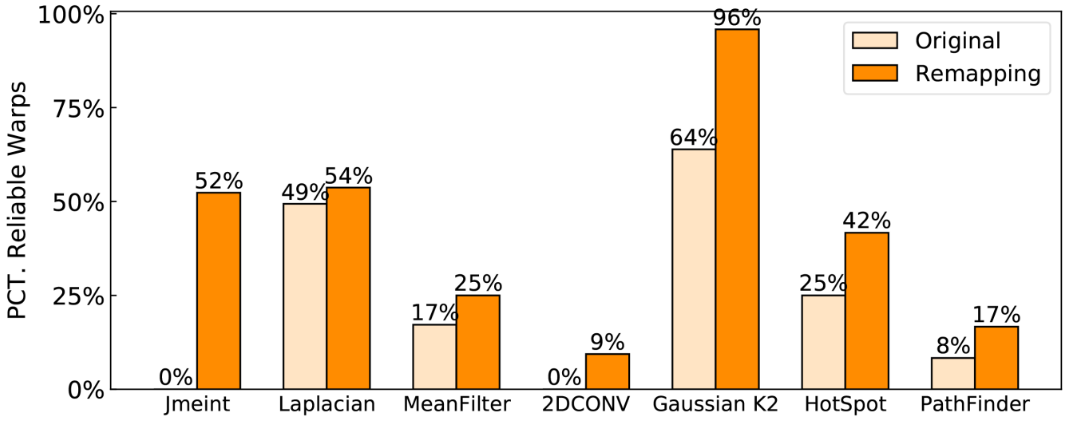
- + Reliable
- x Unreliable

HotSpot
(First 768 threads / 9216 in total)

New reliable warps

➢ Percentage of reliable warps:

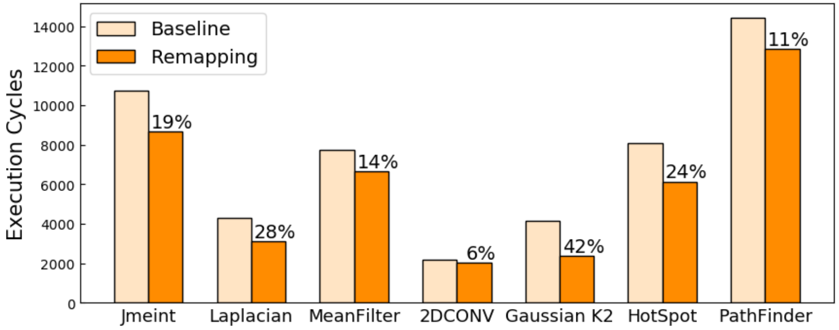
23.40% ⟶ 42.08%



Evaluation: Execution Savings

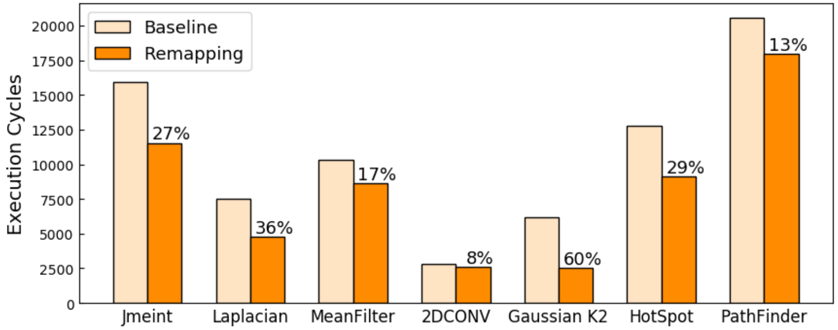
- Error detection: *Remapping* vs. *Duplication* (***R**edundant **M**ulti-**T**hreading*)

  Average Saving: **20.61%**



- Error correction: *Remapping* *vs.* *Triplication* (***T**riple **M**odular **R**edundancy*)

  Average Saving: **27.15%**

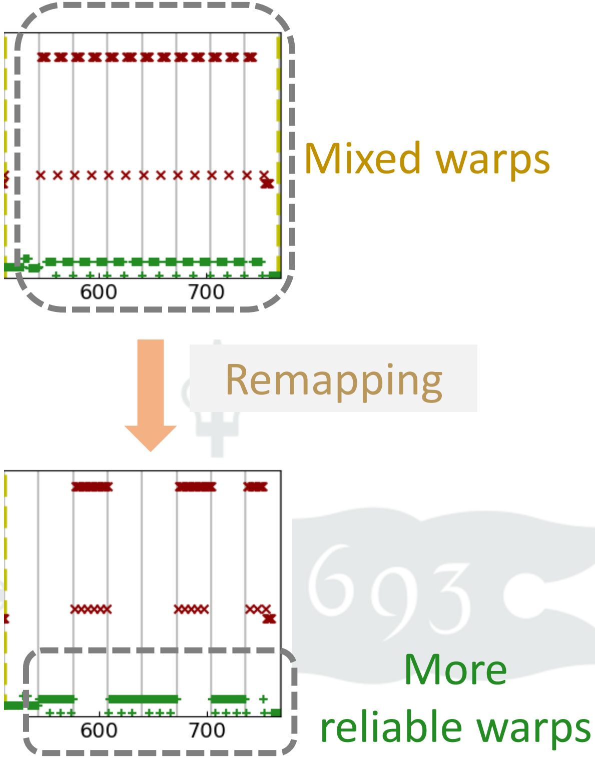


RQ: How to protect GPGPU applications **selectively**?

Answer:

- Reliability characterization at warp level
  - Identify reliable/unreliable/mixed warps
- Remap threads to CTAs: more reliable warps!
- Partial protection

- Low overhead: 1.63%
- Significant execution savings:
  - Error detection: 20.61%
  - Error correction: 27.15%



Thank you :)

Enabling Software Resilience in GPGPU Applications via Partial Thread Protection

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

This work is supported by NSF grant CCF-1717532