Evaluating the Error Resilience of Parallel Programs

Bo Fang , Karthik Pattabiraman, Matei Ripeanu,

The University of British Columbia

Sudhanva Gurumurthi

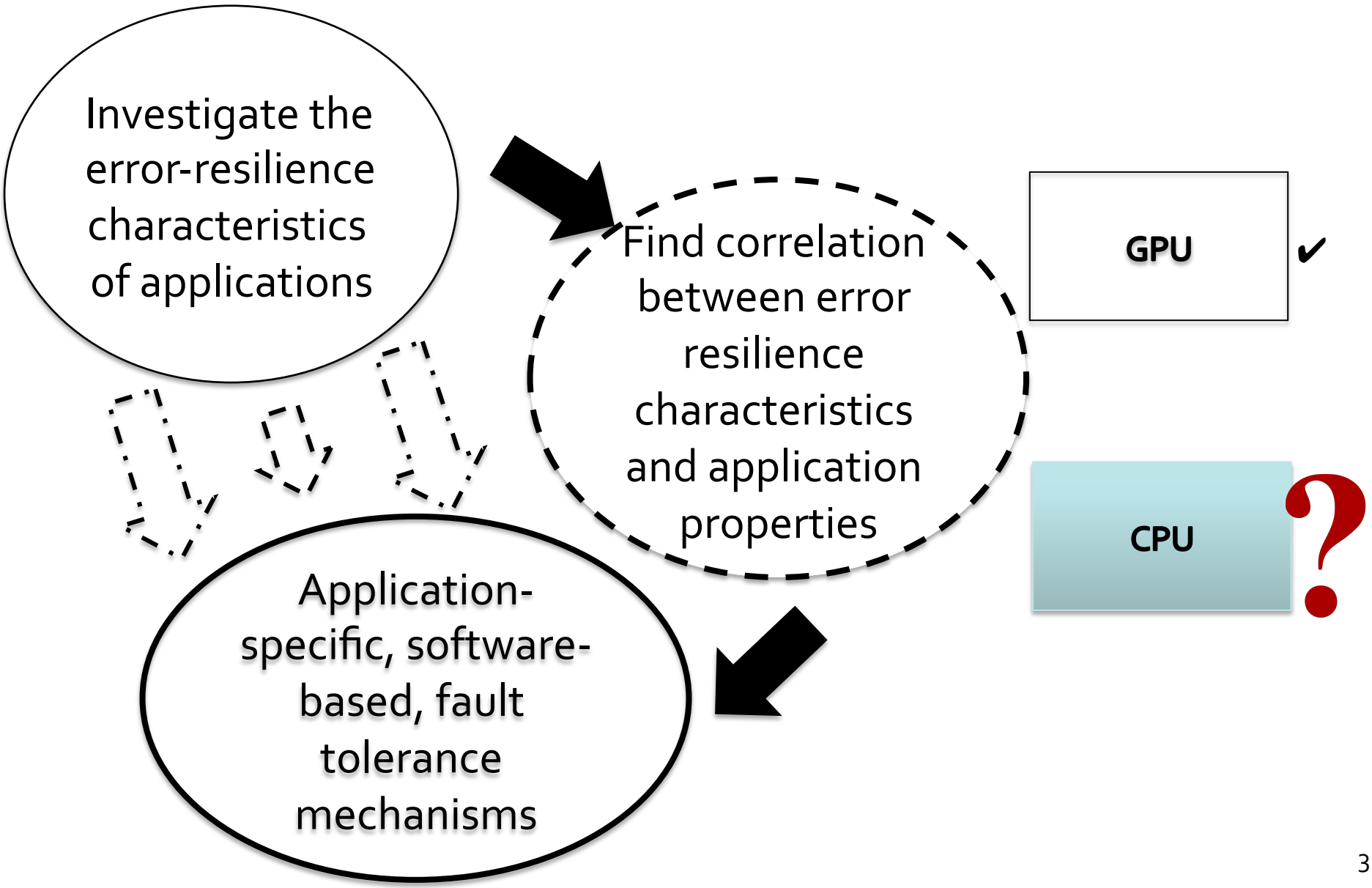
AMD Research

Reliability trends

- The soft error rate per chip will continue to **increase** [Constantinescu, Micro 03; Shivakumar, DSN 02]

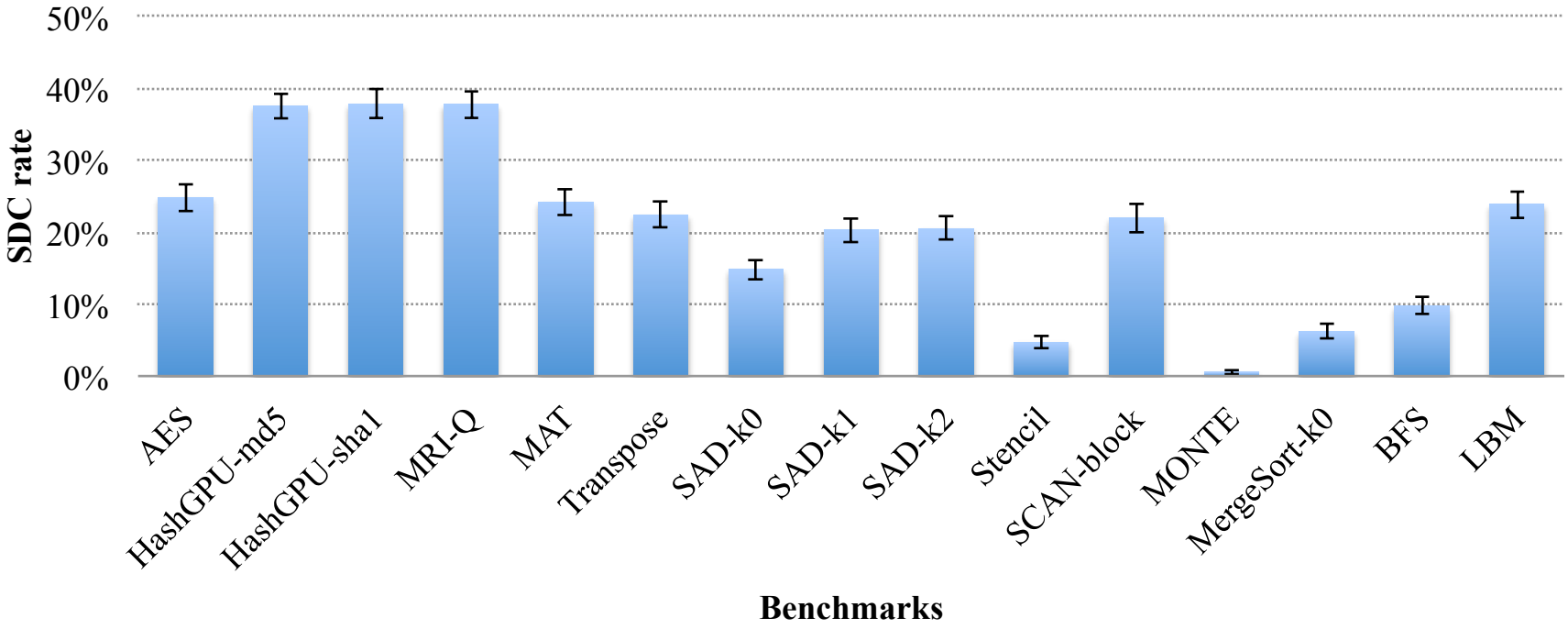
- ASC Q Supercomputer at LANL encountered 27.7 CPU failures per week [Michalak, IEEE transactions on device and materials reliability 05]

Goal


Investigate the error-resilience characteristics of applications
Find correlation between error resilience characteristics and application properties
Application-specific, software-based, fault tolerance mechanisms
GPU
✔
CPU
?

Previous Study on GPUs

- Understand the error resilience of GPU applications by building a methodology and tool called GPU-Qin [Fang, ISPASS 14]

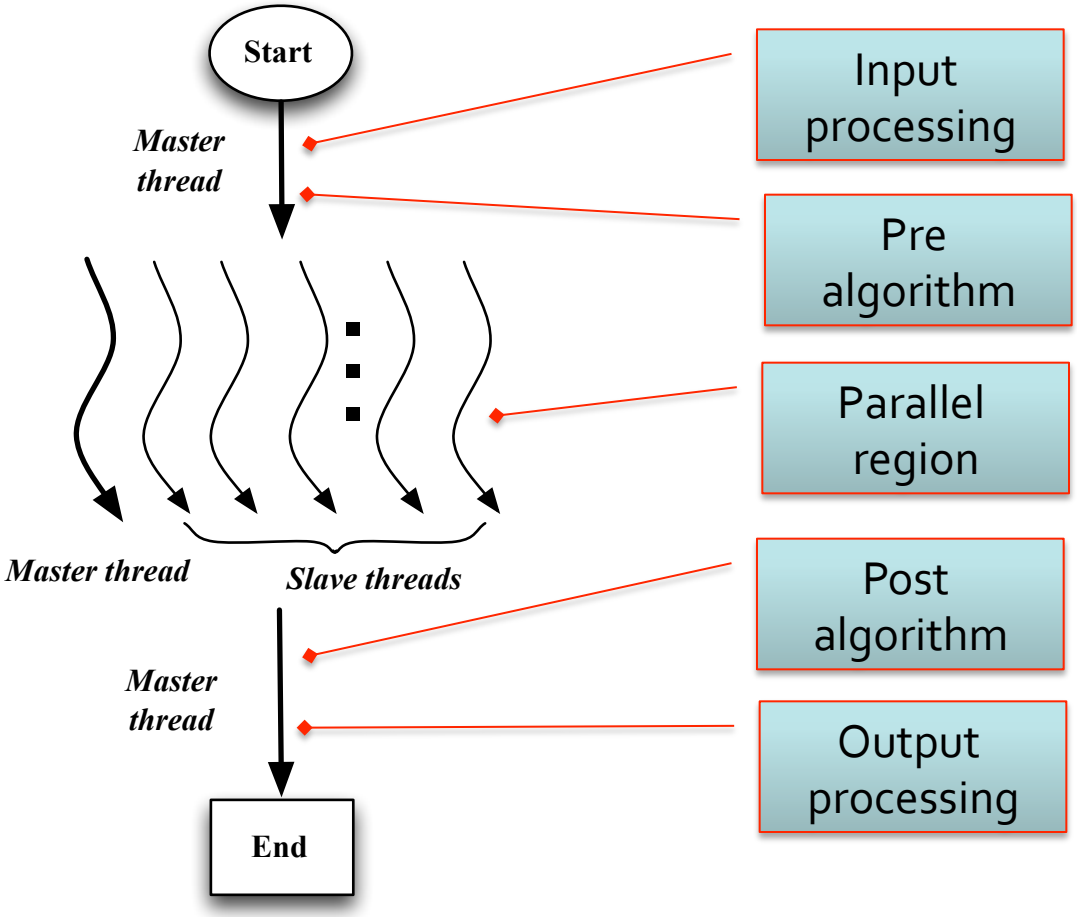


Operations and Resilience of GPU Applications

Operations	Benchmarks	Measured SDC
Comparison-based	MergeSort	6%
Bit-wise Operation	HashGPU, AES	25% - 37%
Average-out Effect	Stencil, MONTE	1% - 5%
Graph Processing	BFS	10%
Linear Algebra	Transpose, MAT, MRI-Q, SCAN-block, LBM, SAD	15% - 38%

This paper: OpenMP programs

- Thread model



- Program structure

```
Begin:
read_graphics(image_orig);
image =
image_setup(image_orig);
scale_down(image)

#pragma omp parallel for
shared(var_list1)
  …

scale_up(image)

write_graphics(image);

End
```

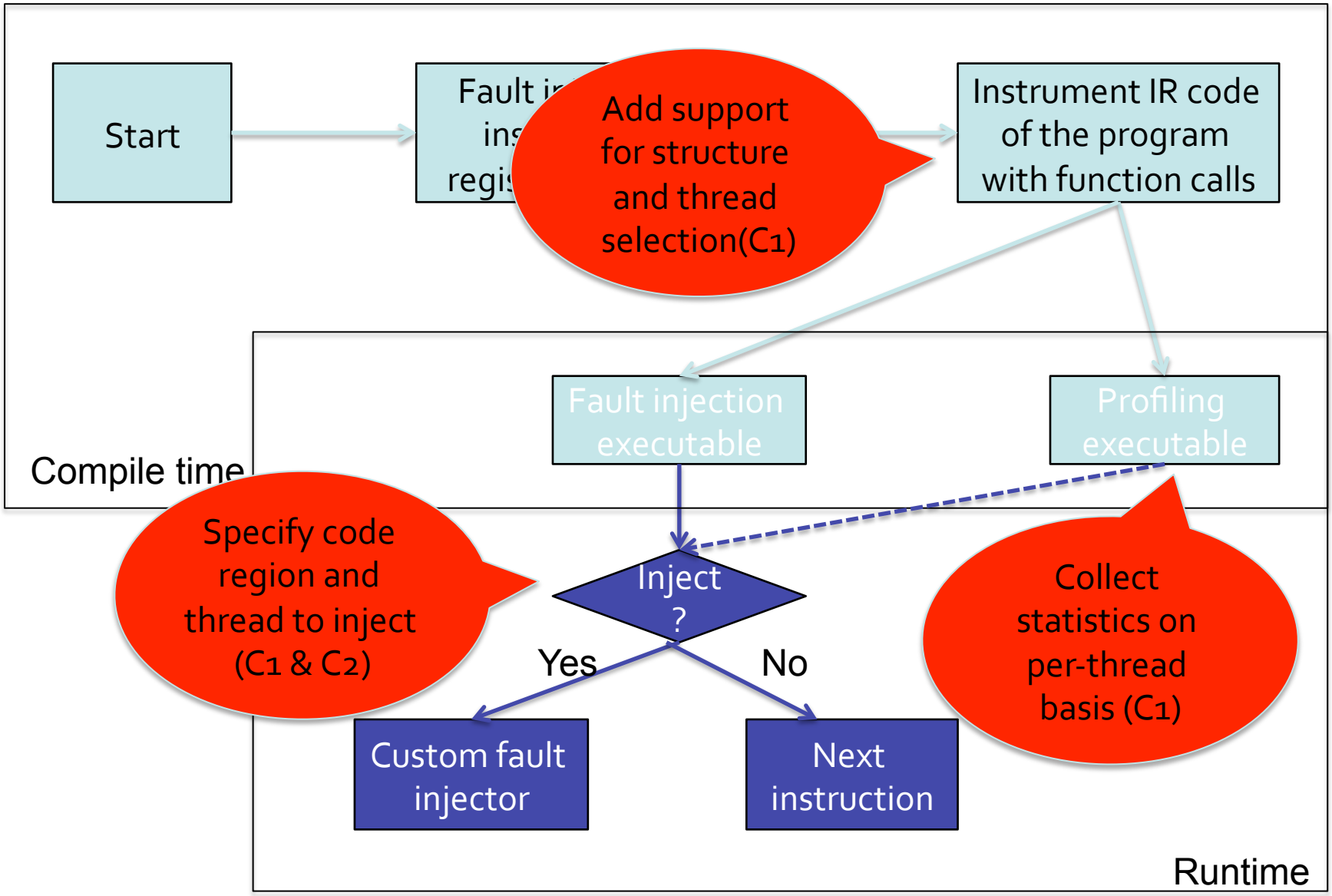
Evaluate the error resilience

- Naïve approach:
  - Randomly choose thread and instruction to inject
  - NOT working: biasing the FT design
- Challenges:
  - C1: exposing the thread model
  - C2: exposing the program structure

Methodology

- Controlled fault injection: fault injection in master and slave threads separately (C1)
  - Integrate with the thread ID
- Identify the boundaries of segments (c2)
  - Identify the range of the instructions for each segment by using source-code to instruction-level mapping
- LLVM IR level (assembly-like, yet captures the program structure)

LLFI and our extension



Fault Model

- Transient faults
- Single-bit flip
  - No cost to extend to multiple-bit flip
- Faults in arithmetic and logic unit(ALU), floating point Unit (FPU) and the load-store unit(LSU, memory-address computation only).
- Assume memory and cache are ECC protected; do not consider control logic of processor (e.g. instruction scheduling mechanism)

Characterization study

- Intel Xeon CPU X5650 @2.67GHz
  - 24 hardware cores
- Outcomes of experiment:
  - ***Benign***: correct output
  - ***Crash***: exceptions from the system or application
  - ***Silent Data Corruption (SDC)***: incorrect output
  - ***Hang***: finish in considerably long time
- 10,000 fault injection runs for each benchmark (to achieve < 1% error bar)

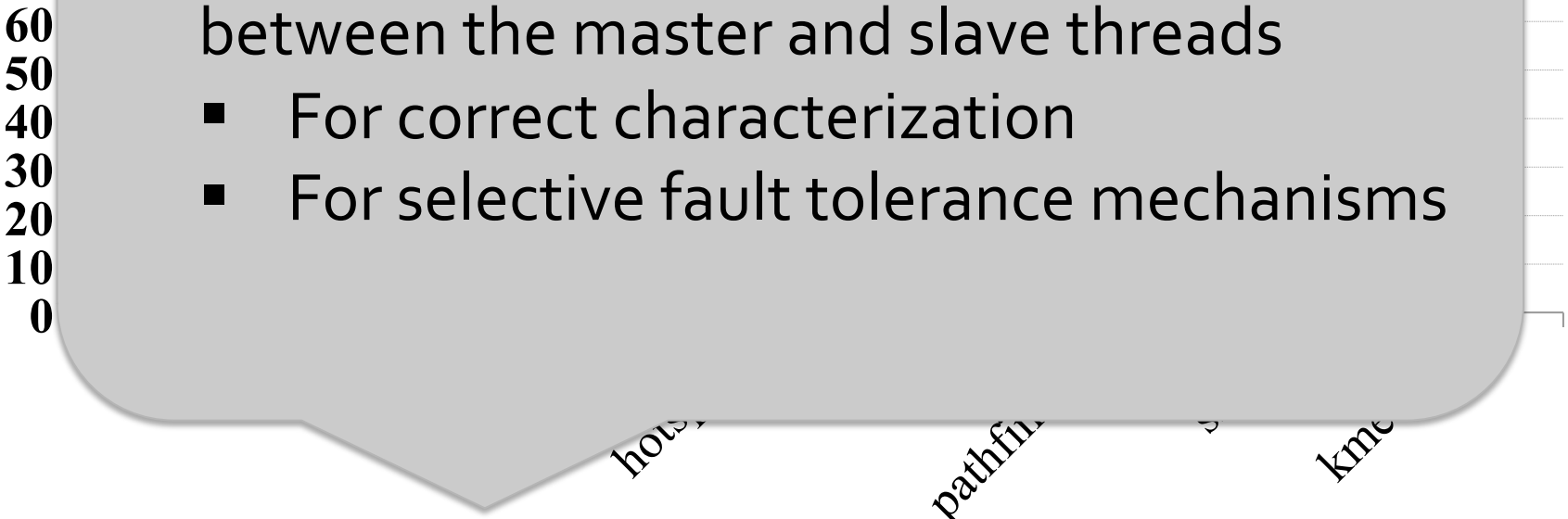
Details about benchmarks

- 8 Applications from Rodinia benchmark suite

Benchmark	Acronym
Bread-first-search	bfs
LU Decomposition	lud
K-nearest neighbor	nn
Hotspot	hotspot
Needleman-Wunsch	nw
Pathfinder	pathfinder
SRAD	srad
Kmeans	kmeans

Difference of SDC rates

- Co

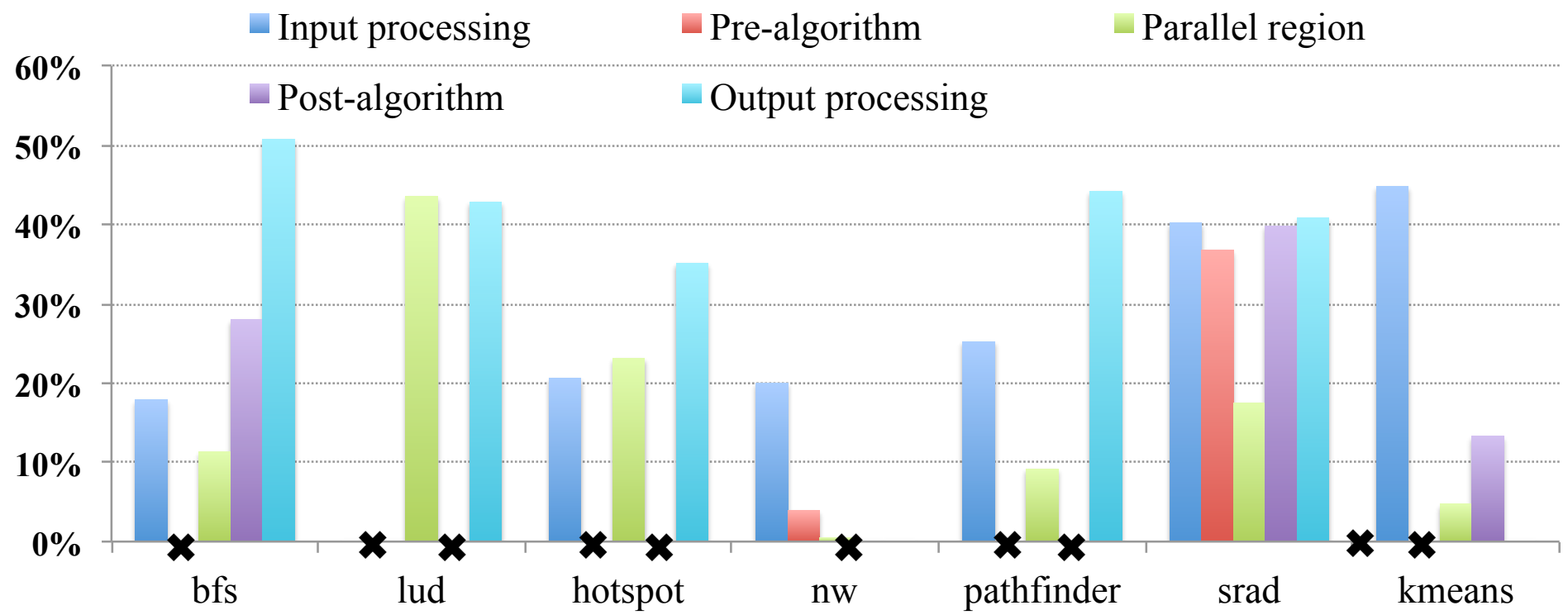


- It shows the importance of differentiating between the master and slave threads
  - For correct characterization
  - For selective fault tolerance mechanisms

- Master threads have higher SDC rates than slave threads (except for lud)
  - Biggest: 16% in pathfinder; Average: 7.8%

Differentiating between program segments

- Faults occur in each of the program segments



Mean and standard deviation of SDC rates for each segment

Segment	Input Processing	Pre-algorithm	Parallel Segment	Post-algorithm	Output processing
Number of applications	6	2	7	2	5
Mean	28.6%	20.3%	16.1%	27%	42.4%
Standard Deviation	11%	23%	14%	13%	5%

- Output processing has the highest mean SDC rate, and also the lowest standard deviation

Grouping by operations

Operations	Benchmarks	Measured SDC
Comparison-based	nn, nw	less than 1%
Grid computation		
Average-out Effect		
Graph Processing		
Linear Algebra		

Operations	Measured SDC
	%
	% - 37%
	% - 5%
	%
	% - 38%

- Comparison-based and average-out operations show the lowest SDC rates in both cases
- Linear algebra and grid computation, which are regular computation patterns, give the highest SDC rates

Future work

Operations	Benchmarks	Measured SDC	Operations	Measured SDC
Comparison-based	nn, nw	less than 1%	Comparison-based	6%
Grid computation	hotspot, srad	23%	Bit-wise Operation	25% - 37%
Average-out Effect	kmeans	4.2%	Average-out Effect	1% - 5%
Graph Processing	bfs, pathfinder	9% ~ 10%	Graph Processing	10%
Linear Algebra	lud	44%	Linear Algebra	15% - 38%

- Understand variance and investigate more applications
- Compare CPUs and GPUs
  - Same applications
  - Same level of abstraction

Conclusion

- Error resilience characterization of OpenMP programs needs to take into account the thread model and the program structure.
- Preliminary support for our hypothesis that error resilience properties do correlate with the algorithmic characteristics of parallel applications.

Project website:

http://netsyslab.ece.ubc.ca/wiki/index.php/FTHPC

Backup slide

SDC rates converge