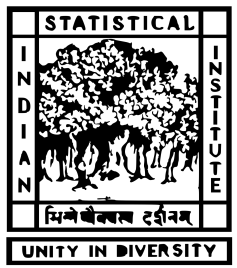
Approximate Computing for Multithreaded Programs in Shared Memory Architectures

*Bernard Nongpoh, †**Rajarshi Ray**, ‡Ansuman Banerjee



*National Institute of Technology Meghalaya, Shillong, India
†Indian Association for the Cultivation of Science, Kolkata, India
‡Indian Statistical Institute, Kolkata, India

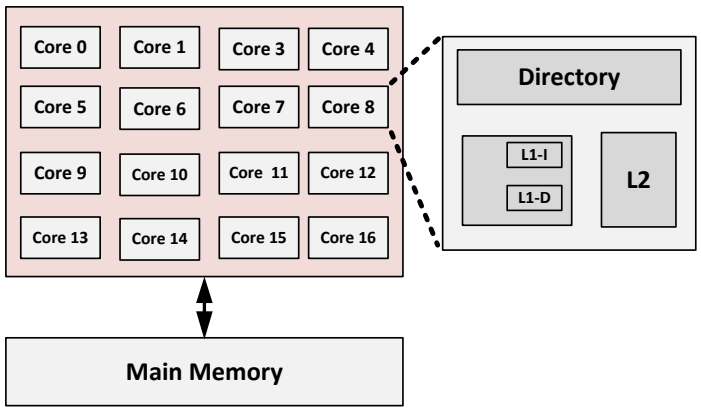
9 October 2019

Outline

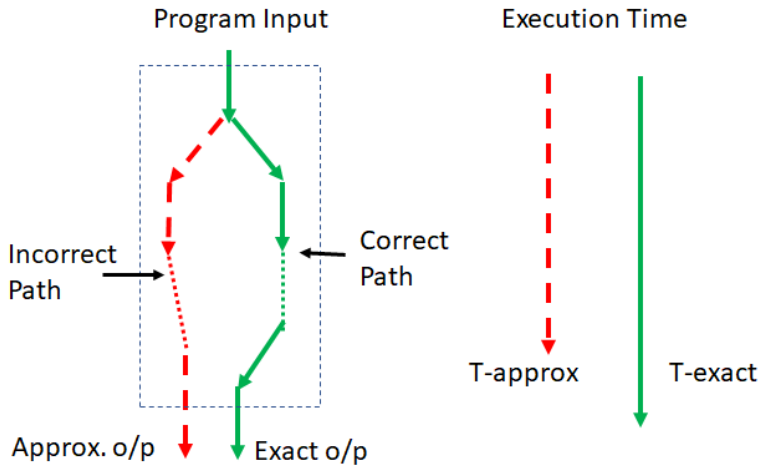
Multicore Architecture and Approximate Computing

Alternatives to slowing Moore's law for performance scaling in microprocessors:

1. Multicore Architecture - many computing cores on-chip.
2. Approximate Computing - trade-off accuracy for performance.



(a) Multicore Architecture



(b) An illustration of Approximate Computing

Challenges in Multicore Processors: The Cache-Coherence Problem

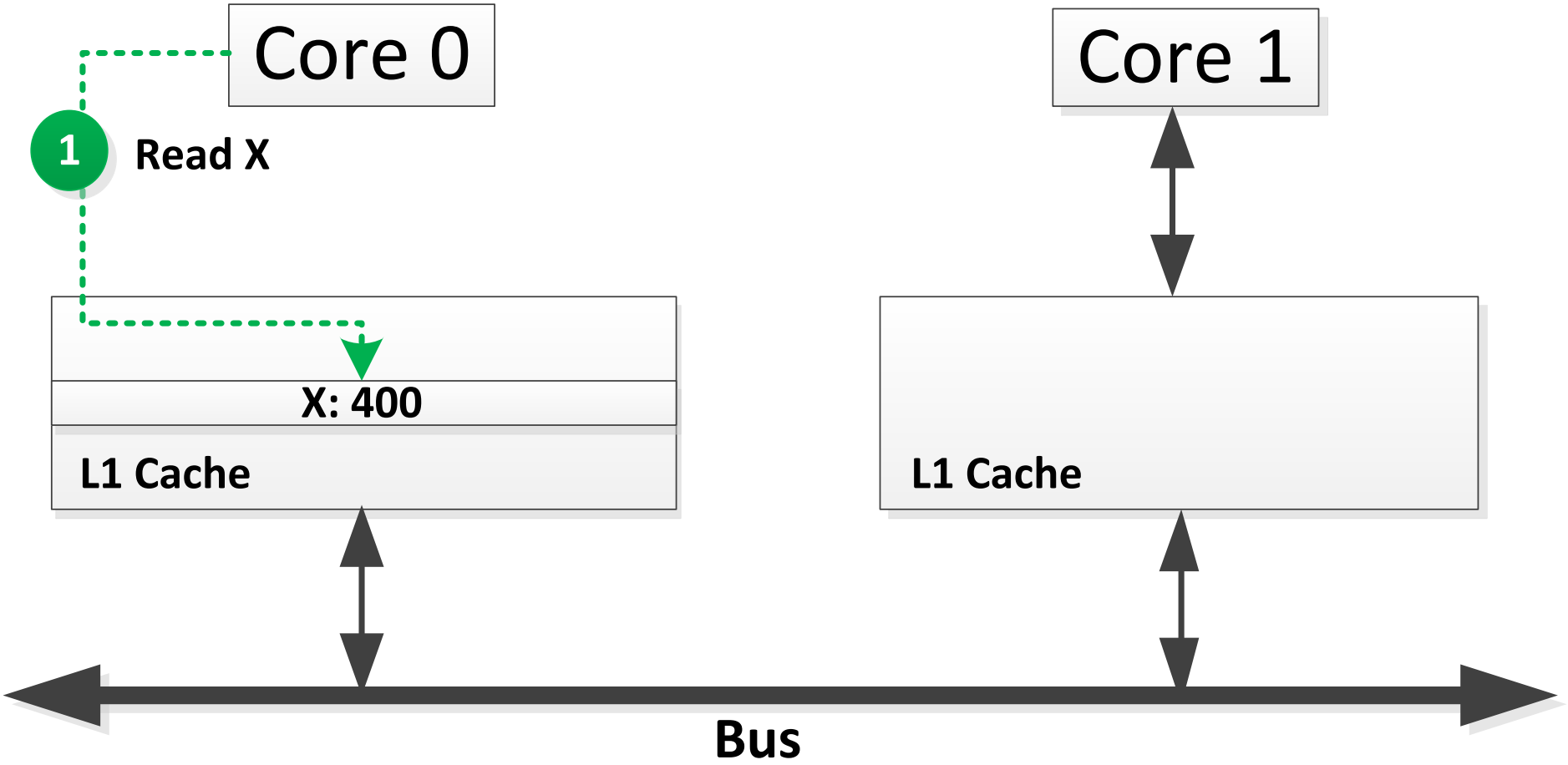


Figure: An Illustration of the Cache-Coherence Problem

The Cache-Coherence Problem

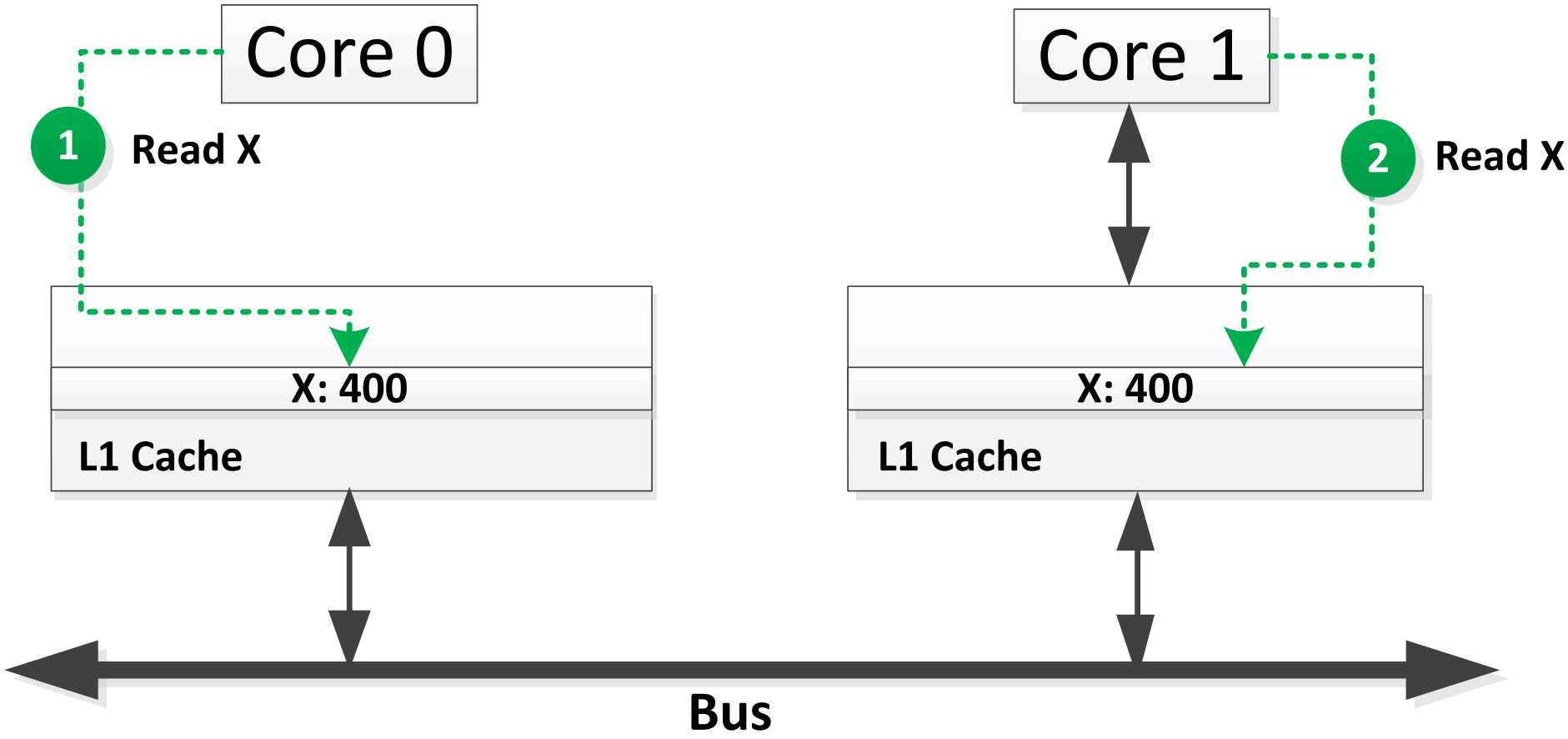


Figure: An Illustration of the Cache-Coherence Problem

The Cache-Coherence Problem

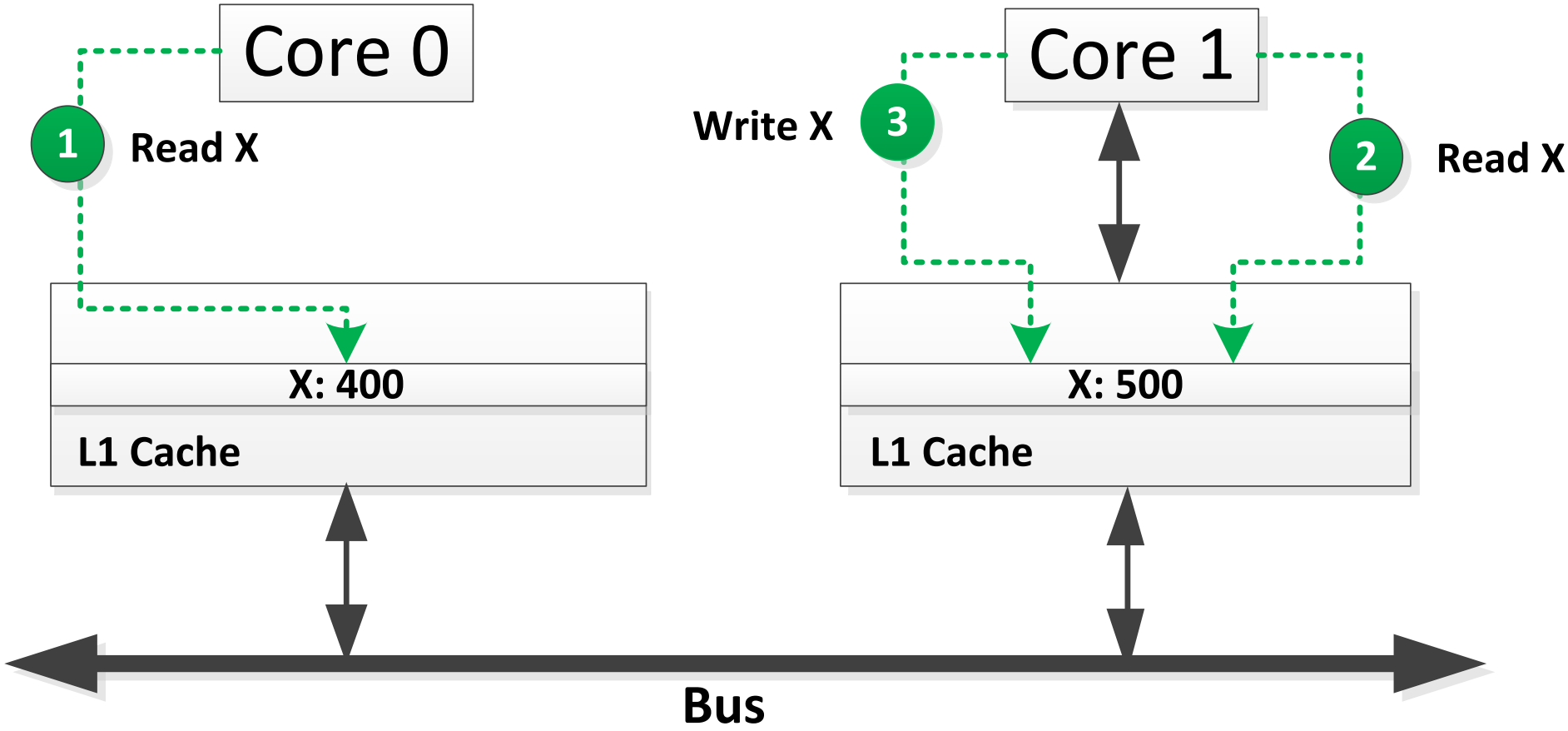


Figure: An Illustration of the Cache-Coherence Problem

The Cache-Coherence Problem

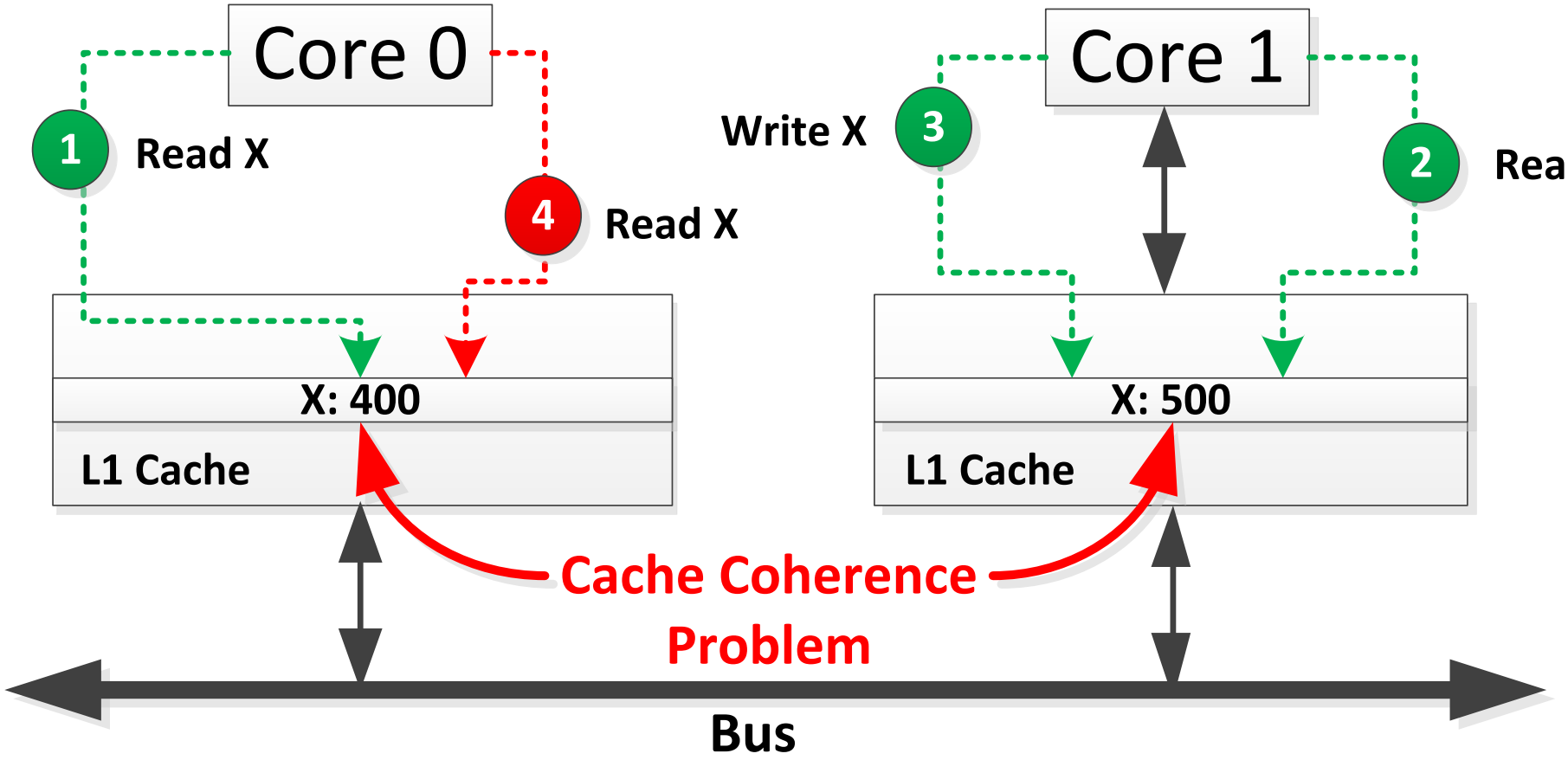
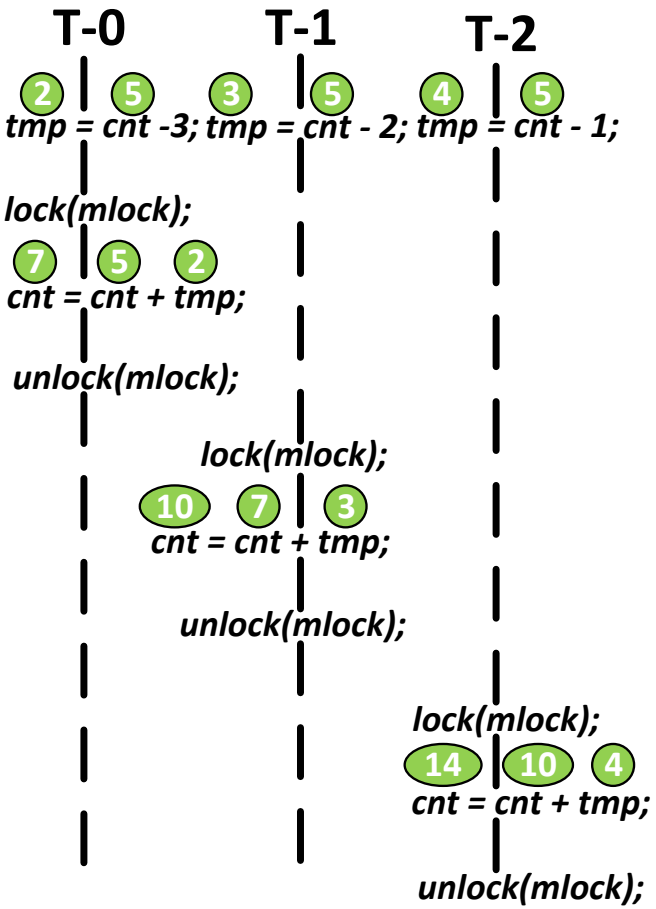


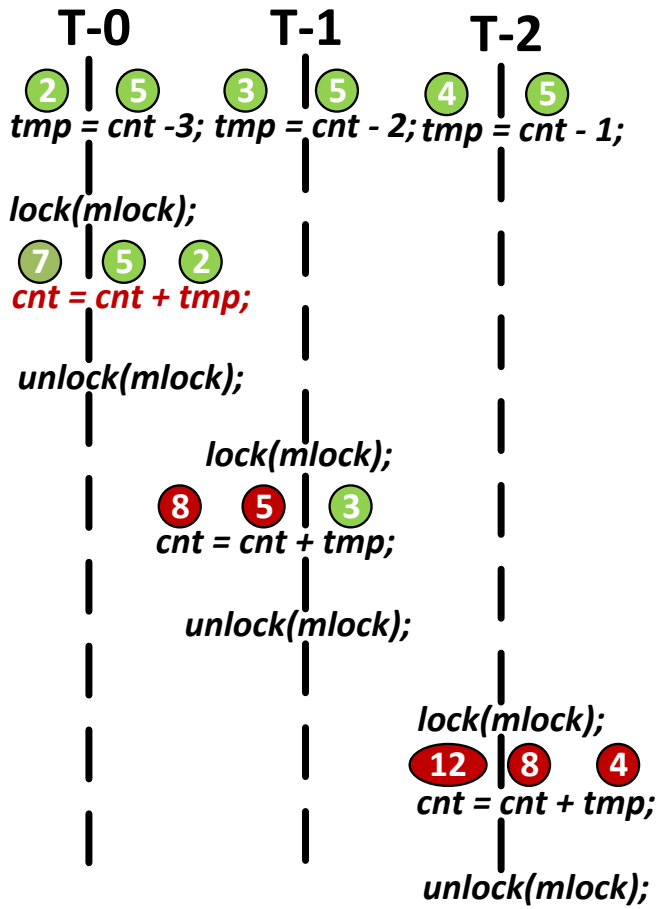
Figure: An Illustration of the Cache-Coherence Problem

Resiliency to Coherence Failure: An Example

A multithreaded program with each thread running on a different core. **cnt is shared** and *tmp* is local to each thread.



(a) Exact execution



(b) Approximate execution

Our Contribution

- **Enhance performance of multi-threaded programs** on shared memory multicore processors by **embracing approximate computing**.
  - We identify write operations on shared data which are tolerant to **coherence faults**.
  - For the fault tolerant memory writes, we propose a modified cache-coherence protocol with a reduced overhead of communication.
  - An auxiliary L1-cache structure to reduce coherency-misses.

Sensitivity Analysis of SWAPs

Definition

Coherence Fault: Let P be a multithreaded program and s be a SWAP in P writing to a shared data x. Let e be the execution of P on an input I. A coherence fault in s refers to the execution e' on the same input I when the write of data v into x is not communicated to the sharers of x. □

Definition

Approximable SWAP: Given a confidence of inference θ and an acceptable QoS distortion measure α, a SWAP s in a multithreaded program P is said to be approximable iff for all executions e of P, the probability that the program output remains within the acceptable QoS distortion α in the presence of a coherence fault in s, is at least θ. □

Detection of Approximable SWAPs by Hypothesis Testing

Given a SWAP s, we consider the Hypotheses:

$$H : Pr(X = 1) \geq \theta; H' : Pr(X = 1) < \theta \quad (1)$$

Where,

- $Pr(X = 1)$ denotes the probability that a coherence fault on the shared data written by the instruction s keeps the program output within the acceptable QoS threshold α, and
- θ is the user-given confidence of inference.
- H and H' respectively represent the null and contrary hypothesis. Note that the verification of the null hypothesis H implies that the SWAP s under analysis is approximable.

Directory-Based Cache-Coherence Protocol

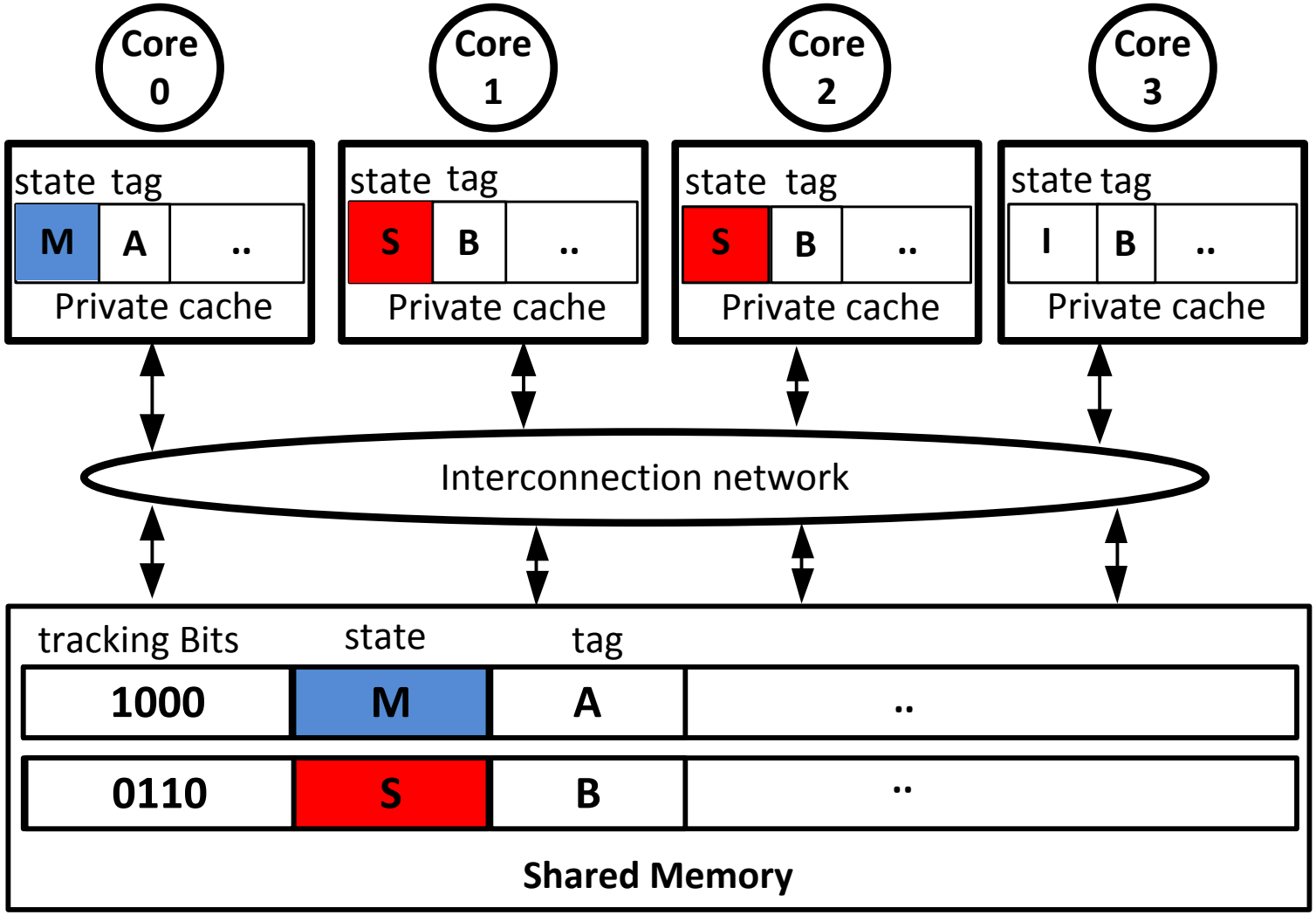
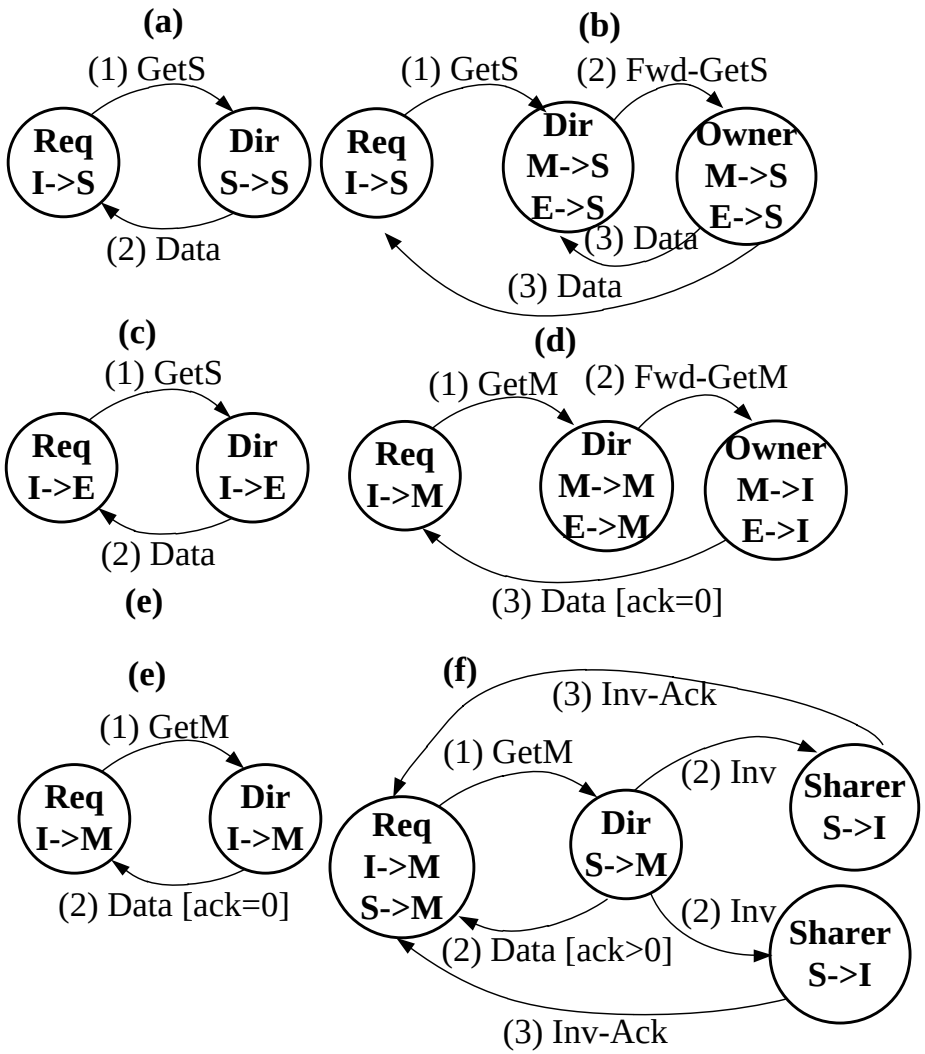


Figure: An Illustration of Directory-based Cache-Coherence Protocol

The MESI Cache-Coherence Protocol



M **Modified**

E **Exclusive**

S **Shared**

I **Invalid**

GetS: Obtain block in read only mode

GetM: Obtain block in read write mode

Fwd-GetS: Forward **GetS** request to owner

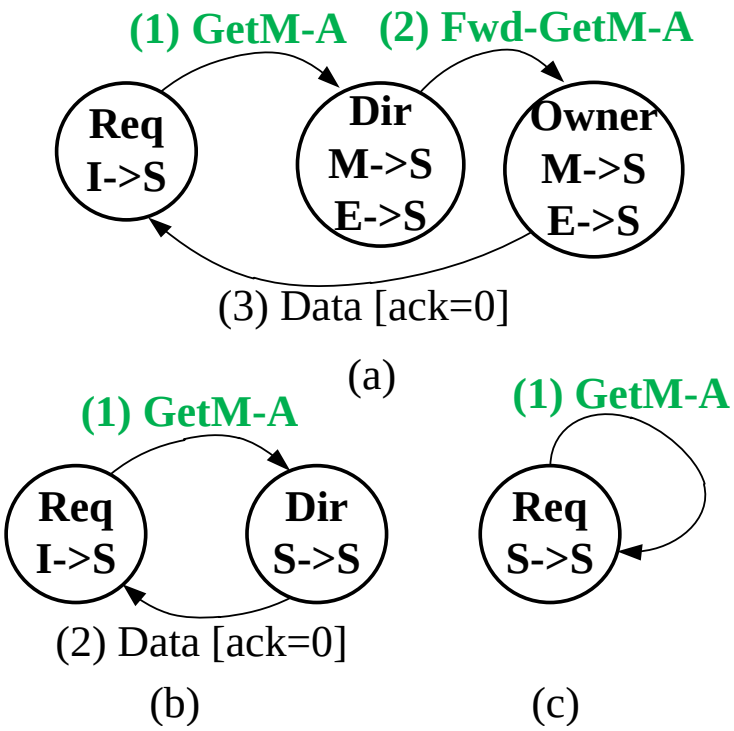
Fwd-GetM: Forward **GetM** request to owner

Inv: Change block to invalid state

Inv-Ack: An acknowledgement of block invalidation

Figure: State Machine of MESI Cache-Coherence Protocol

Modified MESI Protocol



M Modified

E Exclusive

S Shared

I Invalid

GetM-A: Approximable write-requests by requested cores.

Fwd-GetM-A: Forward GetM-A approximable write requests by requested cores to owner

Figure: Modified MESI protocol

Methodology

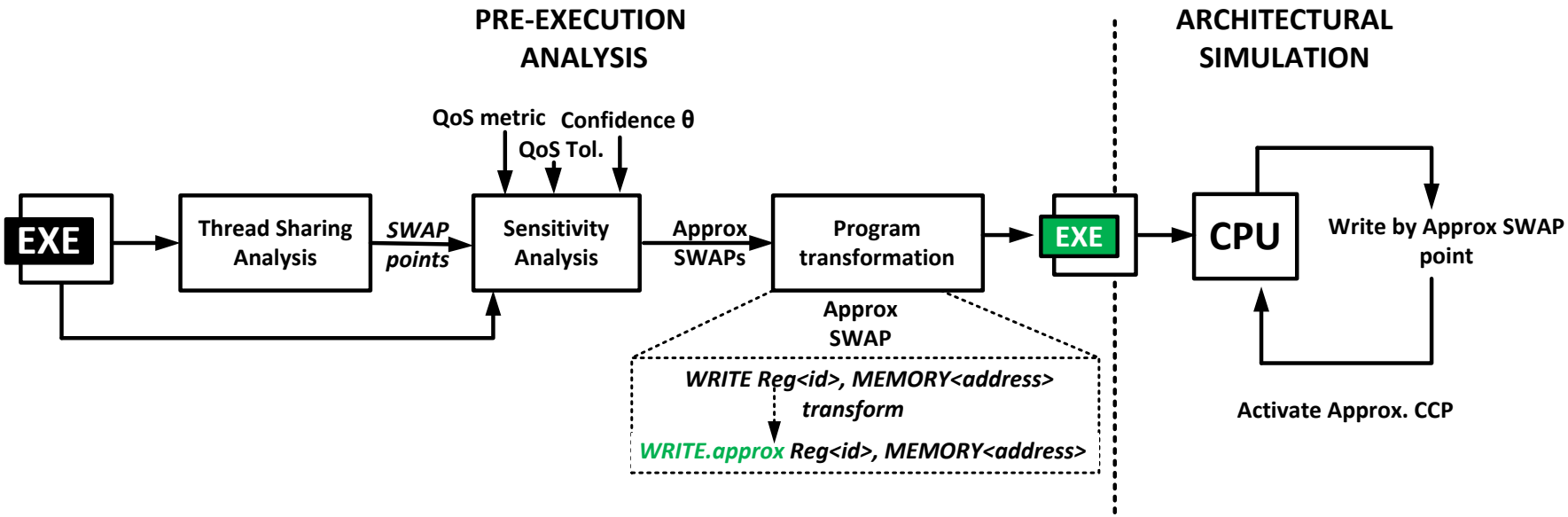


Figure: Workflow

Experimental Setup: Architectural Simulation Configuration

Parameters	Values
Cores	32 cores, Nehalem 2.24GHz
L1	32KB I-Cache, 32KB D-Cache, 32B cache block size, private per core, 2 way associative, 2 cycles hit latency
L2	64KB per core, 32B cache block size, 2-way associative, 20 cycles hit latency.
L3	1024KB, 32B cache block size, 8-way associative, S-NUCA, 100 cycles hit latency, MESI cache coherence protocol
NoC	Mesh 4 $\times$ 8, 4 Mem-Ctlr

Table: Baseline configuration

Parameters	Values
Cores	32 cores, Nehalem 2.24GHz
L1	32KB I-Cache, 32KB D-Cache, 32B cache block size, private per core, 2 way associative, 2 cycles hit latency
Auxiliary Cache	256KB, 32B cache block size,8-way associative, 2 cycles hit latency
L2	64KB per core, 32B cache block size, 2-way associative, 20 cycles hit latency.
L3	1024KB, 32B cache block size, 8-way associative, S-NUCA, 100 cycles hit latency, **Approximate MESI cache coherence protocol**
NoC	Mesh 4 $\times$ 8, 4 Mem-Ctlr

Table: Proposed configuration

Results - Sensitivity Analysis of SWAPs

Table: Sensitivity analysis of SWAPs by hypothesis testing.

Application	LoC	QoS Metric	QoS Tol.	SWAPs Tested	SWAPs Appr.	% Appr.	Time (hrs)
FMM	2945	ARE	≤ 0.1	144	9	6.25	50
OCEAN NC	2731	ARE	≤ 0.1	92	63	68.3	15.6
OCEAN C	4283	ARE	≤ 0.1	88	36	40.9	16.6
RAYTRACER	5783	PSNR	≥ 20	156	98	62.8	17
CHOLESKY	3847	PE	≤ 0.1	190	85	44.7	8.10
FFT	668	NME	≤ 0.1	24	19	79.1	0.82
LU NC	494	NME	≤ 0.1	28	20	71.4	0.51
LU C	916	NME	≤ 0.1	27	20	74	0.52
RADIX	654	matching	0	31	21	67.7	0.8

ARE : Average Relative Error; Appr.:Approximable; %Appr. : Percentage of the tested SWAPs inferred approximable; LoC: Lines of Code; NME : Normalized Mean Error; PE : Percent Error; QoS: Quality of Service; Tol. : Tolerance

- An average of 57% of the tested SWAPs are approximable
- On average 12 hrs analysis time.

Results - Reduced CPU-Cycles and Energy Footprint with Approximate MESI Protocol

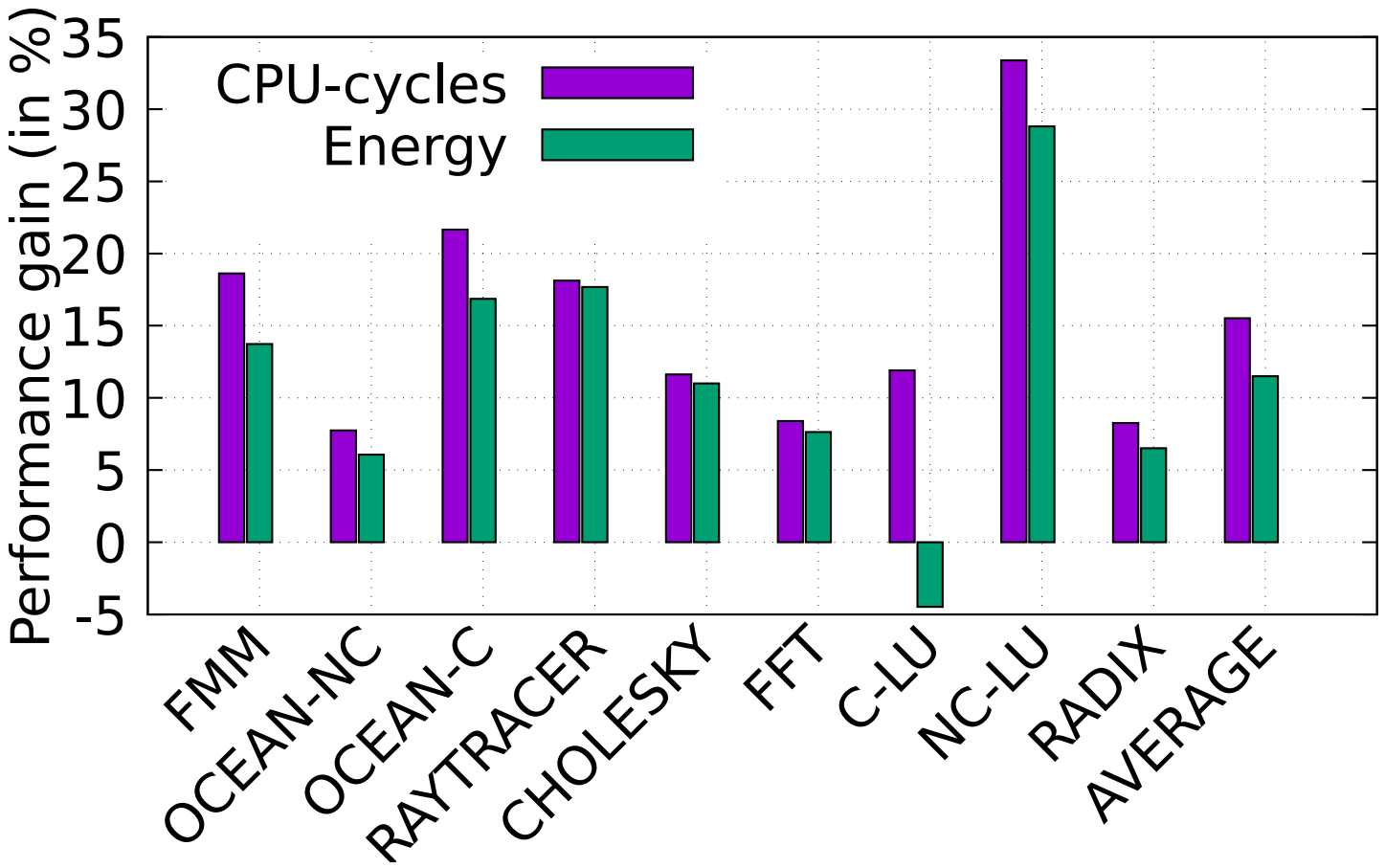


Figure: CPU-cycles, energy reduction (%) in comparison with baseline execution (exact)

Results - Comparison with Related Work

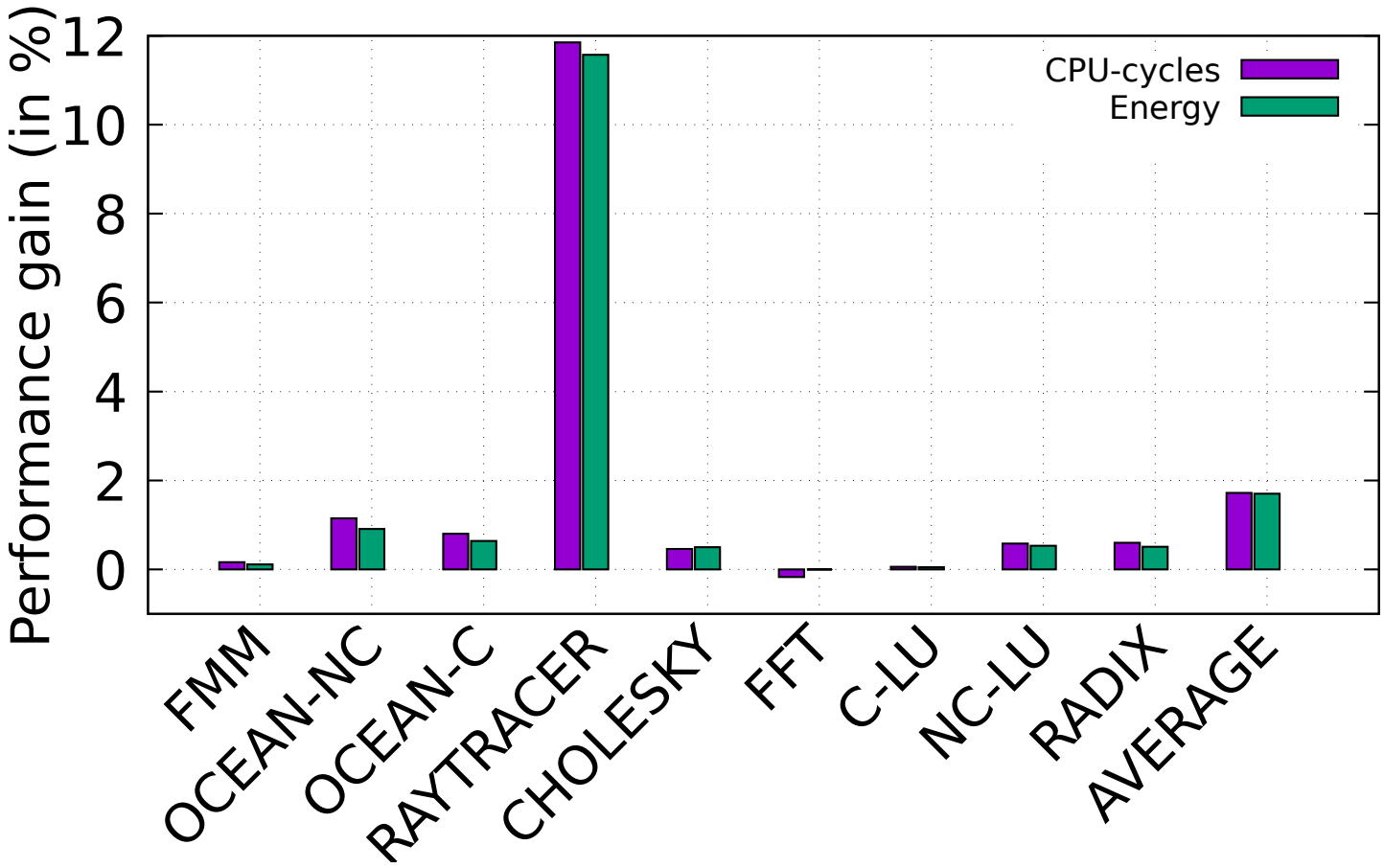


Figure: CPU-cycles, energy reduction (%) in comparison with the work of P. V. Rengasamy et.al., Exploiting staleness for approximating loads on CMPs, PACT'15. [1]

Results - Reduced Coherence-invalidation and Remote cache-loads

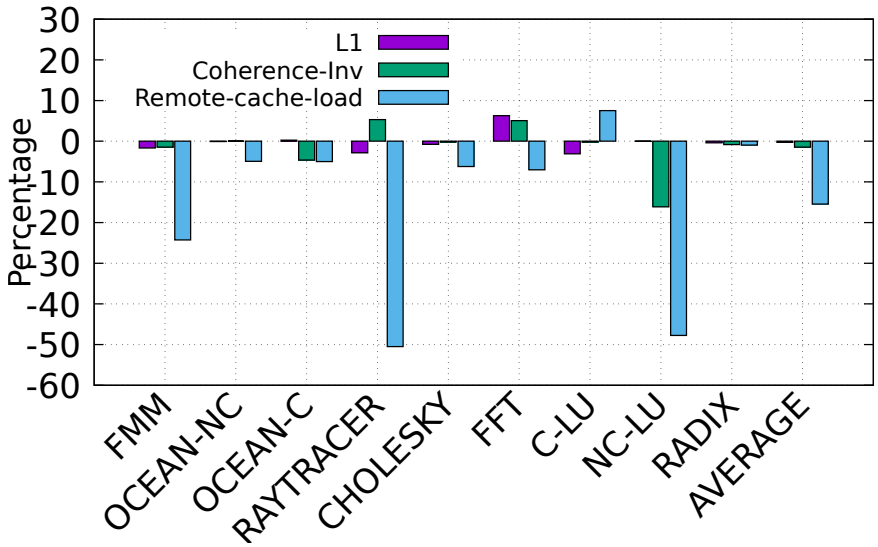


Figure: L1-D cache misses, Coherence invalidation and Remote-cache-load (%) in comparison with baseline.

On average,

- 0.26% reduction in L1-D misses
- 1.45% reduction in invalidation messages due to coherence and
- 15.46% reduction in request to load data messages from remote caches.

Conclusion

- We present a sensitivity analysis technique for instructions in a multi-threaded program, in order to determine the ones which are approximable.
- An average of **57%** of the tested SWAPs are approximable in the considered applications from the SPLASH 3.0 applications.
- In order to reduce the cost of implementing cache-coherency in a processor, we propose an approximate cache-coherence protocol that is invoked on selective write operations.
- On an average, our approach shows **15.5%** gain in terms of CPU cycles and **11.5%** reduction in energy used.

Thank You!

Questions?

References

Prasanna Venkatesh Rengasamy, Anand Sivasubramaniam, Mahmut T. Kandemir, and Chita R. Das.
Exploiting staleness for approximating loads on cmps.
In *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 343–354, Washington DC, USA, 2015. IEEE Computer Society.