COMET: Communication-Optimised Multi-threaded Error-detection Technique

Konstantina Mitropoulou, Vasileios Porpodas and Timothy M. Jones

Computer Laboratory

CASES 2016, Pittsburgh

Outline

- Redundant multi-threaded error detection
- Breakdown of overheads
- COMET optimisations
- Results

Redundant Multi-threaded Error Detection

original code	software redundant multi–threading error detection code	
r1 = r1 + 16	r1 = r1 + 16	r1' = r1' + 16
load r2, (r1)	call enqueue(r1)	r1 = call dequeue()
r2 = r2 + 100	load r2, (r1)	cmp r1, r1'
r3 = r1 + 16	call enqueue(r2)	jmp
store (r3), r2	r2 = r2 + 100	r2 = call dequeue()
	r3 = r1 + 16	r2' = r2 + 100
	call enqueue(r2)	r3' = r1' + 16
	call enqueue(r3)	r2 = call dequeue()
	store (r3), r2	cmp r2, r2'
		jmp
		r3 = call dequeue()
		cmp r3, r3'
		jmp
	original thread	checker thread

Redundant Multi-threaded Error Detection

software redundant multi–threading error detection code

communication for checking

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
call enqueue(r1)	r1 = call dequeue()
load r2, (r1)	cmp r1, r1'
call enqueue(r2)	jmp
r2 = r2 + 100	r2 = call dequeue()
r3 = r1 + 16	r2' = r2 + 100
call enqueue(r2)	r3' = r1' + 16
call enqueue(r3)	r2 = call dequeue()
store (r3), r2	cmp r2, r2'
	jmp
	r3 = call dequeue()
	cmp r3, r3'
	jmp

Redundant Multi-threaded Error Detection

software redundant multi–threading error detection code

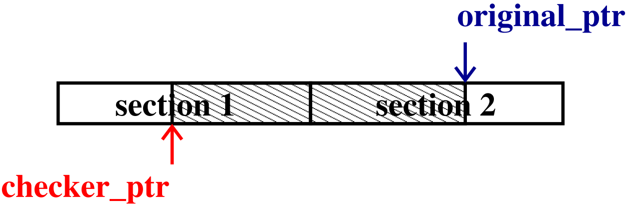
communication for checking

communication for update

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
call enqueue(r1)	r1 = call dequeue()
load r2, (r1)	cmp r1, r1'
call enqueue(r2)	jmp
r2 = r2 + 100	r2 = call dequeue()
r3 = r1 + 16	r2' = r2 + 100
call enqueue(r2)	r3' = r1' + 16
call enqueue(r3)	r2 = call dequeue()
store (r3), r2	cmp r2, r2'
	jmp
	r3 = call dequeue()
	cmp r3, r3'
	jmp

Redundant Multi-threaded Error Detection

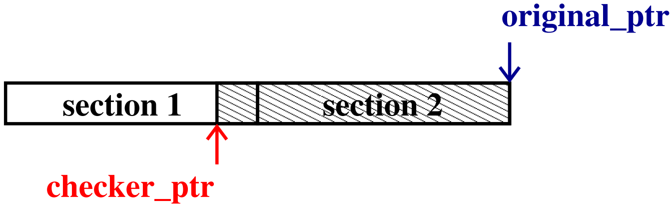
The two threads communicate through a multi-section queue.



- Each section is exclusively used by one thread.

Redundant Multi-threaded Error Detection

The two threads communicate through a multi-section queue.



- Each section is exclusively used by one thread.
- Each thread cannot access the following section if the other thread still uses it.

UNIVERSITY OF CAMBRIDGE

Breakdown of Communication Overheads

Breakdown of Communication Overheads

- Frequent communication

software redundant multi–threading error detection code

communication for checking

communication for update

```
r1 = r1 + 16
call enqueue(r1)
load r2, (r1)
call enqueue(r2)
r2 = r2 + 100
r3 = r1 + 16
call enqueue(r2)
call enqueue(r3)
store (r3), r2
```

original thread

```
r1’ = r1’ + 16
r1 = call dequeue()
cmp r1, r1’
jmp
r2 = call dequeue()
r2’ = r2 + 100
r3’ = r1’ + 16
r2 = call dequeue()
cmp r2, r2’
jmp
r3 = call dequeue()
cmp r3, r3’
jmp
```

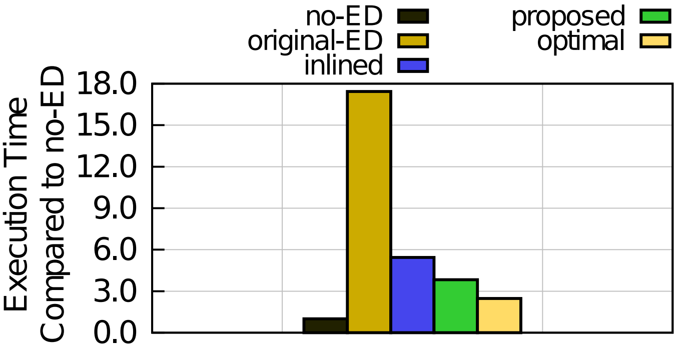
checker thread

Breakdown of Communication Overheads

- Frequent communication
- Function call overhead

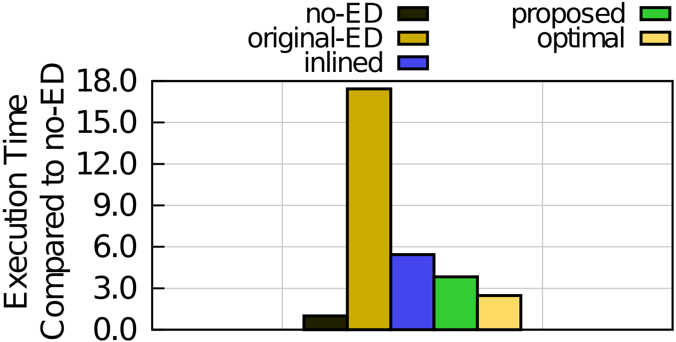
Breakdown of Communication Overheads

- Frequent communication
- Function call overhead



Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- inline enqueue/dequeue functions



Breakdown of Communication Overheads

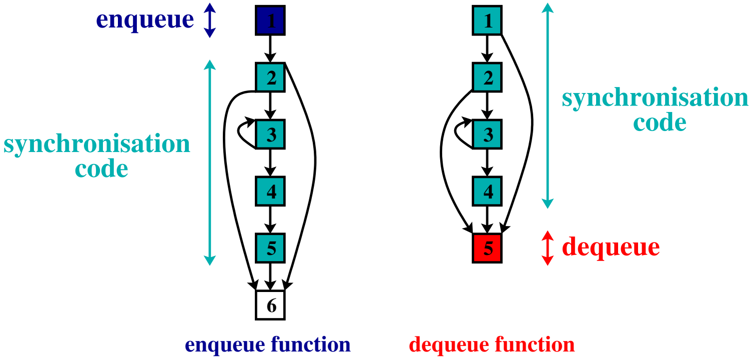
- Frequent communication
- Function call overhead
- Overheads of inlining

Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- Overheads of inlining
    - control-flow overhead

Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- Overheads of inlining
    - control-flow overhead

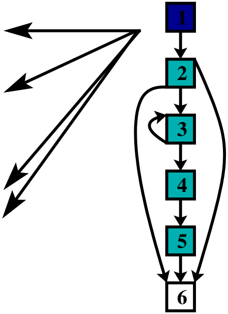


Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- Overheads of inlining
    - control-flow overhead

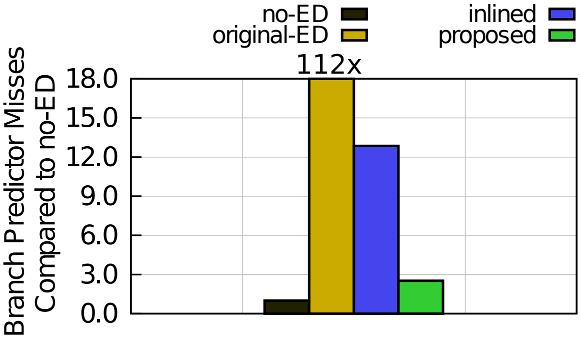
```
r1 = r1 + 16
call enqueue(r1)
load r2, (r1)
call enqueue(r2)
r2 = r2 + 100
r3 = r1 + 16
call enqueue(r2)
call enqueue(r3)
store (r3), r2
```

original thread



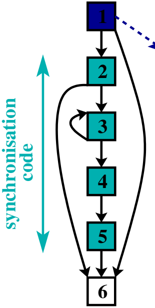
Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- Overheads of inlining
  - control-flow overhead



Breakdown of Communication Overheads

- Frequent communication
- Function call overhead
- Overheads of inlining
  - control-flow overhead
  - **6** instructions in the critical path



enqueue

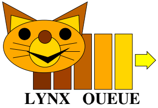
lea rax, [rdx+8]	incr pointer
mov QWORD_PTR [rdx], rcx	store
mov rdx, rax	compiler's copy
and rdx, ROTATE_MASK	rotate pointer
test eax, SECTION_MASK	end of section
jne .L2	skip sync code

Reduce Communication Overhead

Optimality: enqueue and dequeue operations should have **2** instructions overheads:

- a store instruction for writing the data to the queue
- an addition to increment the enqueue or dequeue pointer

Reduce Communication Overhead

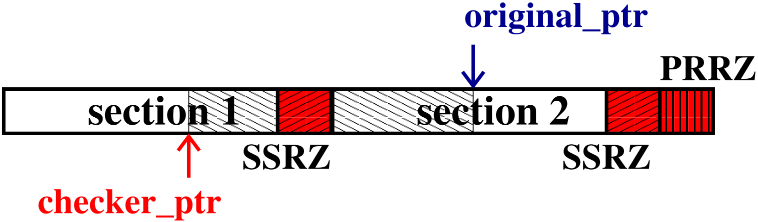


Lynx is a SP/SC queue which relies on memory protection system. In this way, each enqueue/dequeue operation has **2** instructions overhead.

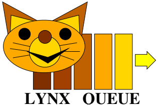
Reduce Communication Overhead

LYNX QUEUE

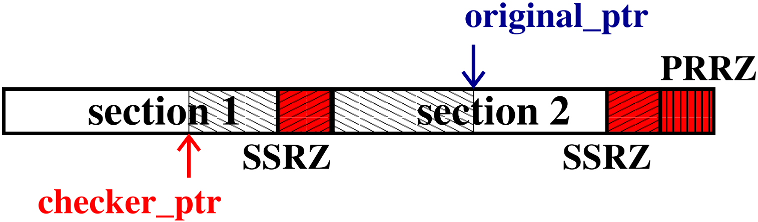
Lynx is a SP/SC queue which relies on memory protection system. In this way, each enqueue/dequeue operation has **2** instructions overhead.



Reduce Communication Overhead

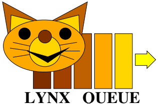


Lynx is a SP/SC queue which relies on memory protection system. In this way, each enqueue/dequeue operation has **2** instructions overhead.

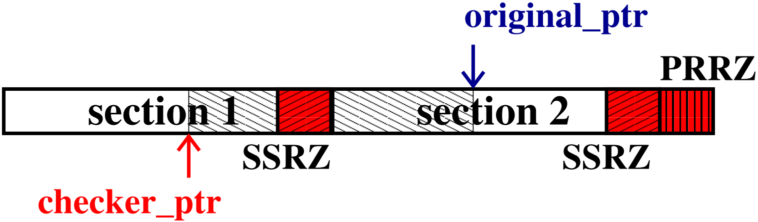


- SSRZ: Section Synchronisation Red Zone

Reduce Communication Overhead

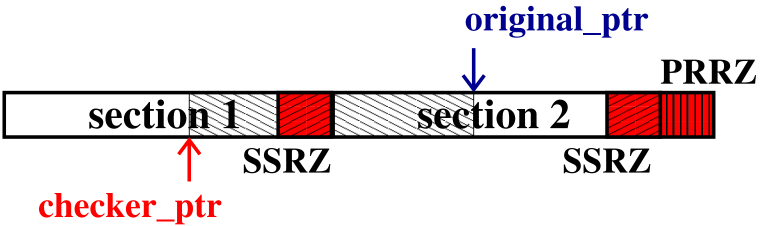


Lynx is a SP/SC queue which relies on memory protection system. In this way, each enqueue/dequeue operation has **2** instructions overhead.

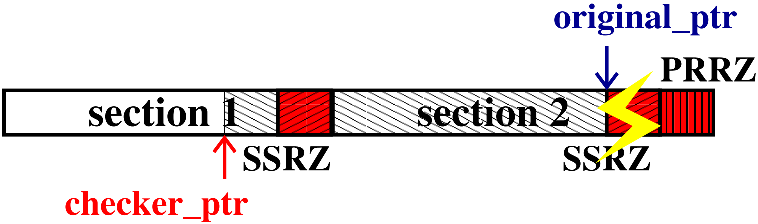


- SSRZ: Section Synchronisation Red Zone
- PRRZ: Pointer Rotation Red Zone

Reduce Communication Overhead



Reduce Communication Overhead



- The exception is captured by Lynx's handler
- Lynx's handler does the job of the synchronization code

Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

Reduce Communication Overhead

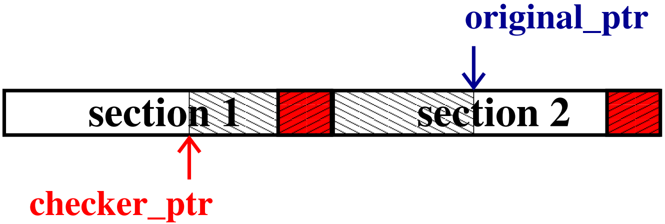
COMET optimises Lynx queue by applying the following compiler optimisations:

- Simplify Lynx design by using a fixed register (R15) for the enqueue/dequeue pointers:

Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

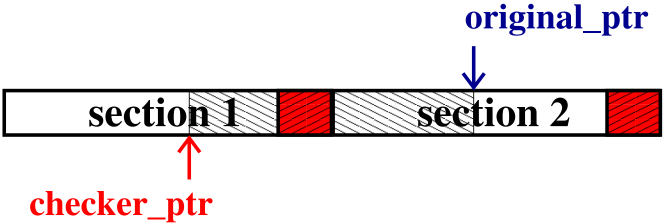
- Simplify Lynx design by using a fixed register (R15) for the enqueue/dequeue pointers:
  - The queue has two fixed red-zones



Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

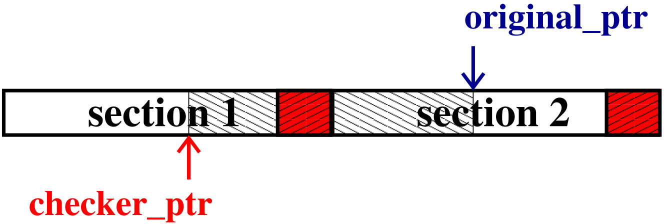
- Simplify Lynx design by using a fixed register (R15) for the enqueue/dequeue pointers:
  - The queue has two fixed red-zones
  - The handler has less things to do



Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

- Simplify Lynx design by using a fixed register (R15) for the enqueue/dequeue pointers:
  - The queue has two fixed red-zones
  - The handler has less things to do
  - We can apply more optimisations



Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

- Simplify Lynx design

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
call enqueue(r1)	r1 = call dequeue()
r2 = r2 + 100	cmp r1, r1'
call enqueue(r2)	jmp
store (r1), r2	r2' = r2' + 100
	r2 = call dequeue()
	cmp r2, r2'
	jmp

error detection code with enqueue and dequeue calls

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
store (r15), r1	load r1, (r15)
r15 = r15 + 8	r15 = r15 + 8
r2 = r2 + 100	cmp r1, r1'
store (r15), r2	jmp
r15 = r15 + 8	r2' = r2' + 100
store (r1), r2	load r2, (r15)
	r15 = r15 + 8
	cmp r2, r2'
	jmp

inlined code with COMET

Reduce Communication Overhead

COMET optimises Lynx queue by applying the following compiler optimisations:

- Simplify Lynx design
- Address offset fusion optimisation

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
store (r15), r1	load r1, (r15)
r15 = r15 + 8	r15 = r15 + 8
r2 = r2 + 100	cmp r1, r1'
store (r15), r2	jmp
r15 = r15 + 8	r2' = r2' + 100
store (r1), r2	load r2, (r15)
	r15 = r15 + 8
	cmp r2, r2'
	jmp

inlined code with COMET

original thread	checker thread
r1 = r1 + 16	r1' = r1' + 16
store (r15), r1	load r1, (r15)
r2 = r2 + 100	cmp r1, r1'
store (r15 + 8), r2	jmp
store (r1), r2	r2' = r2' + 100
	load r2, (r15 + 8)
	cmp r2, r2'
	jmp

inlined code with COMET and offset optimisation

Reduce Communication Frequency

Packed Checking: COMET packs the communication operations for each store instruction:

```
r1 = r1 + 16              r1' = r1' + 16
store (r15), r1           load r1, (r15)
r2 = r2 + 100             cmp r1, r1'
store (r15 + 8), r2       jmp
store (r1), r2            r2' = r2' + 100
                          load r2, (r15 + 8)
                          cmp r2, r2'
                          jmp

original thread           checker thread
```

inlined code with COMET and offset optimisation

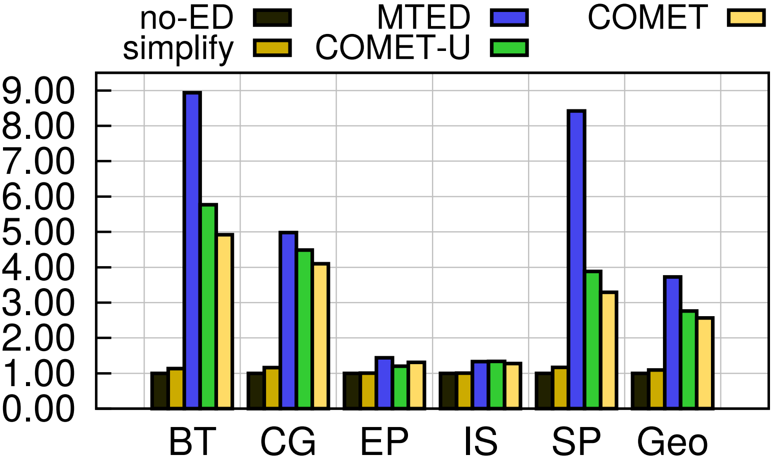
```
r1 = r1 + 16              r1' = r1' + 16
r2 = r2 + 100             r2' = r2' + 100
r2 = r1 XOR r2            r2' = r1' XOR r2'
store (r15), r2           load r2, (r15)
store (r1), r2            cmp r2, r2'
                          jmp

original thread           checker thread
```

COMET with packed checking optimisation

Experimental Setup

- Compiler:
    - GCC-4.9.0
- Performance evaluation:
    - Intel Core i5-4570 @ 3.2GHz desktop machine
- Fault-coverage evaluation:
    - in-house tool based on gdb
- Benchmark suite:
    - NAS benchmarks

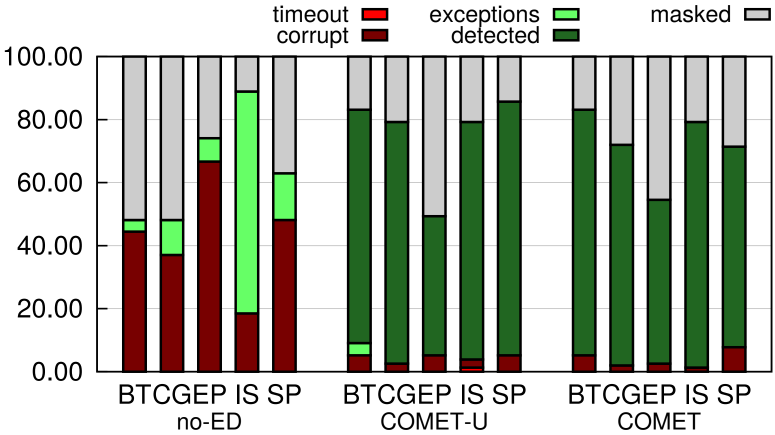
Performance Evaluation



Fault Coverage Evaluation

- Single-Event Upset (SEU) fault model
- Monte Carlo simulation:
  1. count the dynamic instructions
  2. randomly pick one instruction
  3. randomly pick one bit of the instruction's output
  4. run the program
  5. repeat steps 2 to 4 for 300 times
- Type of errors:
  - *detected errors* are the ones that COMET detects
  - *masked errors* do not alter program's output
  - *exceptions* can be detected by a specialised exception handler
  - *corrupt errors* change program's output
  - *timeout errors* result in infinite execution of the program

Fault Coverage Evaluation



Summary

- The communication is the main performance bottleneck of redundant multi-threaded error detection

Summary

- The communication is the main performance bottleneck of redundant multi-threaded error detection
- COMET can potentially reduce the communication overhead down to one instruction

Summary

- The communication is the main performance bottleneck of redundant multi-threaded error detection
- COMET can potentially reduce the communication overhead down to one instruction
- COMET reduces the communication frequency

Summary

- The communication is the main performance bottleneck of redundant multi-threaded error detection
- COMET can potentially reduce the communication overhead down to one instruction
- COMET reduces the communication frequency
- COMET improves performance by 31.4% on average

Summary

- The communication is the main performance bottleneck of redundant multi-threaded error detection
- COMET can potentially reduce the communication overhead down to one instruction
- COMET reduces the communication frequency
- COMET improves performance by 31.4% on average
- The proposed optimisations do not affect the fault-coverage