MEMORIA TRANSACCIONAL "IN-CORE" CON CERROJOS EN LEX ORDER

Álvaro Rubira García

UNIVERSIDAD DE MURCIA

Índice

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

contador: 0

Hilo 1

- rA = contador
- rB = rA + 1
- contador = rB

Hilo 2

- rA = contador
- rB = rA + 1
- contador = rB

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

contador: 0

Hilo 1

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0

Hilo 2

- rA = contador
- rB = rA + 1
- contador = rB

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

contador: 0

Hilo 1

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0

Hilo 2

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

contador: 0

Hilo 1

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0
rB: 1

Hilo 2

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0
rB: 1

Estado del arte

- Procesadores multinúcleo
- Comunicación a través de memoria compartida

contador: 1

Hilo 1

- rA = contador
- rB = rA + 1
- contador = rB

rA: 0
rB: 1

Hilo 2

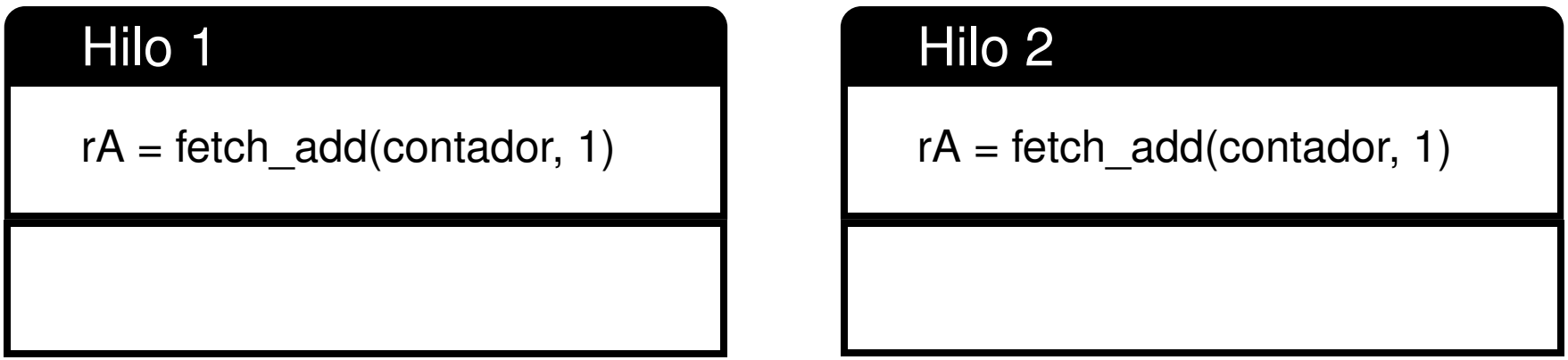
- rA = contador
- rB = rA + 1
- contador = rB

rA: 0
rB: 1

Estado del arte

- ISA incluye atómicos para operaciones simples

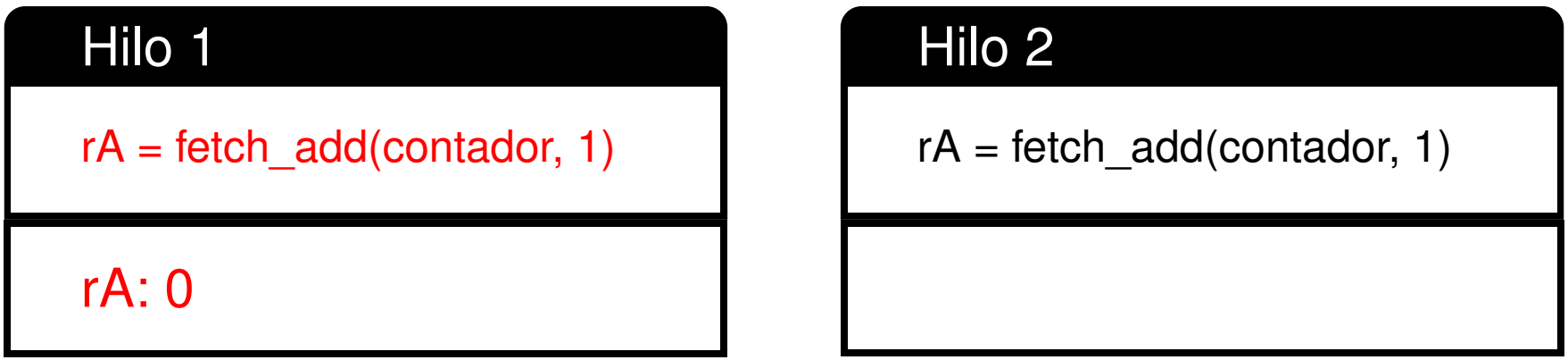
contador: 0



Estado del arte

- ISA incluye atómicos para operaciones simples

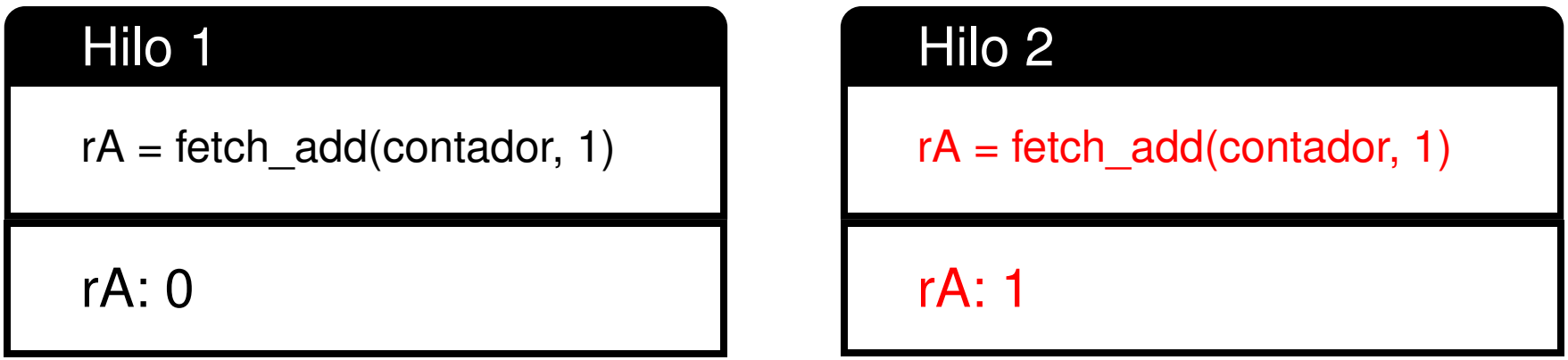
contador: 1



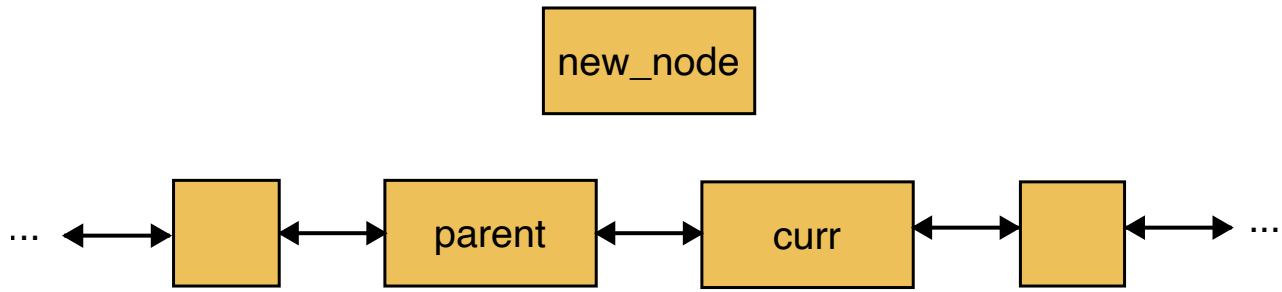
Estado del arte

- ISA incluye atómicos para operaciones simples

contador: 2



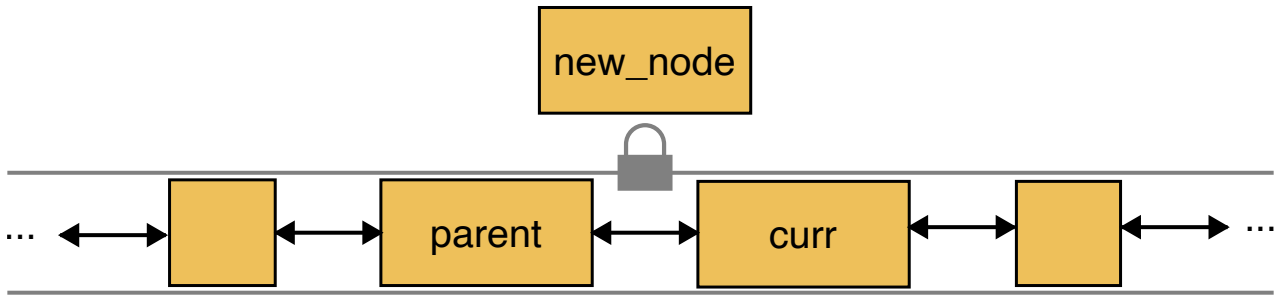
ESTADO DEL ARTE




```
1    new_node->next = curr;
2    new_node->prev = parent;
3    parent->next = new_node;
4    curr->prev = new_node;
```

ESTADO DEL ARTE

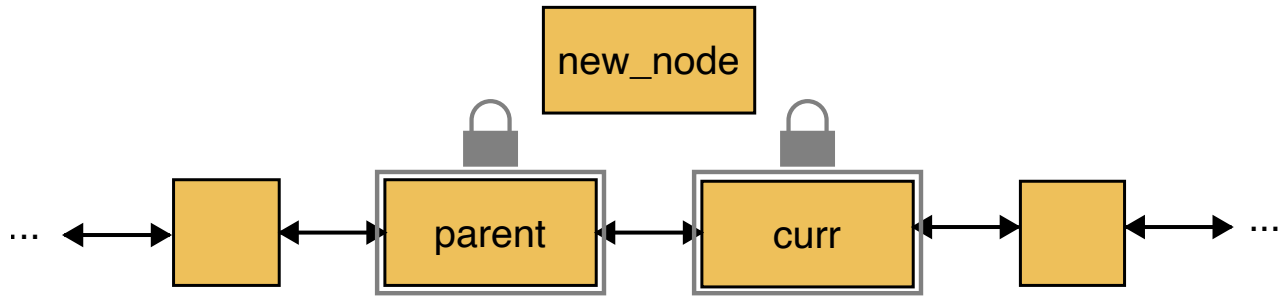
CERROJOS DE GRANO GRUESO




```
1  lock();
2  new_node->next = curr;
3  new_node->prev = parent;
4  parent->next = new_node;
5  curr->prev = new_node;
6  unlock();
```

ESTADO DEL ARTE

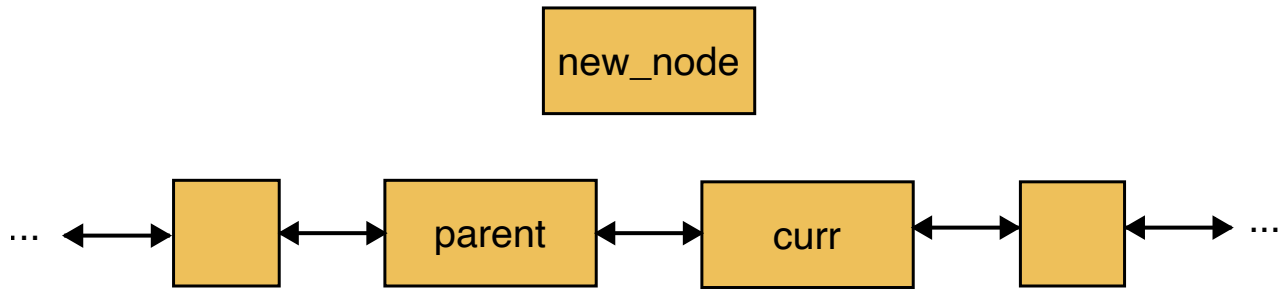
CERROJOS DE GRANO FINO




```
1  parent->lock();
2  curr->lock();
3  new_node->next = curr;
4  new_node->prev = parent;
5  parent->next = new_node;
6  curr->prev = new_node;
7  parent->unlock();
8  curr->unlock();
```

Estado del arte

Memoria transaccional




```
1  TM_BEGIN();
2  new_node->next = curr;
3  new_node->prev = parent;
4  parent->next = new_node;
5  curr->prev = new_node;
6  TM_END();
```

Estado del arte
Memoria transaccional

- Transacciones se ejecutan especulativamente
- Detección de conflictos
- *Aborts* y reintentos
- Hardware dedicado (HTM)
  - Memoria transaccional software es muy flexible
  - Difícil soportar transacciones complejas en hardware

Objetivos e implementación

- Los objetivos de este trabajo son:

Objetivos e implementación

- Los objetivos de este trabajo son:
  1. Implementar HTM *constrained* con pocas modificaciones al hardware existente del procesador

Objetivos e implementación

- Los objetivos de este trabajo son:
  1. Implementar HTM *constrained* con pocas modificaciones al hardware existente del procesador
  2. Evaluar propuesta para reducir conflictos mediante *cache line locking* dinámico

Objetivos e implementación

- Los objetivos de este trabajo son:
  1. Implementar HTM *constrained* con pocas modificaciones al hardware existente del procesador
  2. Evaluar propuesta para reducir conflictos mediante *cache line locking* dinámico
- Simulador gem5

Objetivos e implementación

HTM in-core

- *Speculative lock elision*[1] propone dos implementaciones

[1] Ravi Rajwar and James R Goodman. *Speculative lock elision: Enabling highly concurrent multithreaded execution*, MICRO-34, 2001.

Objetivos e implementación

HTM in-core

- *Speculative lock elision*[1] propone dos implementaciones
- *Out-of-core*
  - *Checkpoint* de registros
  - Marca líneas de caché

[1] Ravi Rajwar and James R Goodman. *Speculative lock elision: Enabling highly concurrent multithreaded execution*, MICRO-34, 2001.

Objetivos e implementación

HTM in-core

- *Speculative lock elision*[1] propone dos implementaciones
- *Out-of-core*
  - *Checkpoint* de registros
  - Marca líneas de caché
- *In-core*
  - Reutiliza hardware existente para ejecución especulativa
  - Límites más estrictos sobre complejidad de transacciones

[1] Ravi Rajwar and James R Goodman. *Speculative lock elision: Enabling highly concurrent multithreaded execution*, MICRO-34, 2001.

Objetivos e implementación

HTM in-core

- *Buffering* de ejecución especulativa en el ROB
  - Límite de instrucciones

Objetivos e implementación

HTM in-core

- *Buffering* de ejecución especulativa en el ROB
  - Límite de instrucciones
- Bloques accedidos se leen a caché de primer nivel
  - Prefetches exclusivos para escrituras
  - Límite de líneas de caché accedidas

Objetivos e implementación

HTM in-core

- *Buffering* de ejecución especulativa en el ROB
  - Límite de instrucciones
- Bloques accedidos se leen a caché de primer nivel
  - Prefetches exclusivos para escrituras
  - Límite de líneas de caché accedidas
- Detección de conflictos mediante LQ y SQ
  - *Backoff* en hardware

Objetivos e implementación

HTM in-core

- *Buffering* de ejecución especulativa en el ROB
  - Límite de instrucciones
- Bloques accedidos se leen a caché de primer nivel
  - Prefetches exclusivos para escrituras
  - Límite de líneas de caché accedidas
- Detección de conflictos mediante LQ y SQ
  - *Backoff* en hardware
- *Commit* de transacción
  - Escritura atómica de modificaciones

Objetivos e implementación

Cerrojos en lex order

- Propuesta anterior[2] emplea cerrojos estáticos

[2] Eduardo José Gómez-Hernández, Juan M Cebrian, Stefanos Kaxiras, and Alberto Ros. *Bounding speculative execution of atomic regions to a single retry*, ASPLOS-29, 2024.

[3] Alberto Ros and Stefanos Kaxiras. *Non-speculative store coalescing in total store order*, ISCA-45, 2018.

Objetivos e implementación

Cerrojos en lex order

- Propuesta anterior[2] emplea cerrojos estáticos
- Cerrojos de líneas de caché en lexicographical order[3] (lex order)
  - Subconjunto del orden de direcciones de memoria
  - Evita deadlocks en estructuras privadas (L1, L2) y compartidas (L3, directorio *sparse*)

[2] Eduardo José Gómez-Hernández, Juan M Cebrian, Stefanos Kaxiras, and Alberto Ros. *Bounding speculative execution of atomic regions to a single retry*, ASPLOS-29, 2024.

[3] Alberto Ros and Stefanos Kaxiras. *Non-speculative store coalescing in total store order*, ISCA-45, 2018.

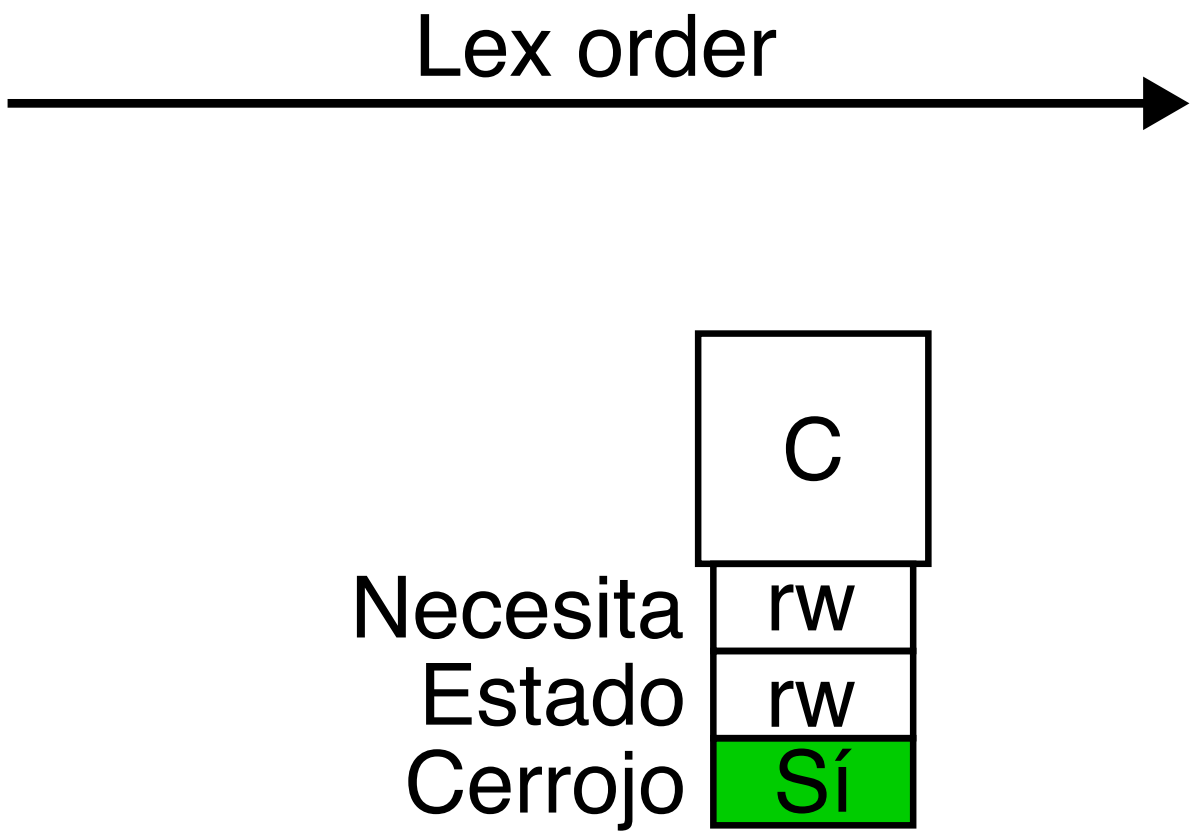
Objetivos e implementación

Cerrojos en lex order

- Propuesta anterior[2] emplea cerrojos estáticos
- Cerrojos de líneas de caché en lexicographical order[3] (lex order)
  - Subconjunto del orden de direcciones de memoria
  - Evita deadlocks en estructuras privadas (L1, L2) y compartidas (L3, directorio *sparse*)
- Activamos cerrojos dinámicos

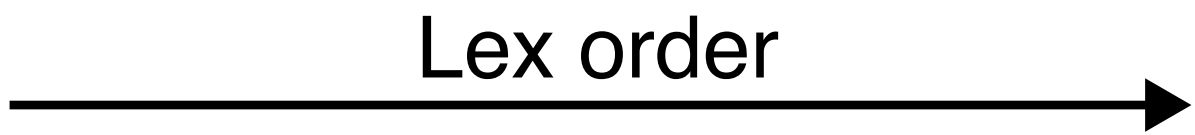
[2] Eduardo José Gómez-Hernández, Juan M Cebrian, Stefanos Kaxiras, and Alberto Ros. *Bounding speculative execution of atomic regions to a single retry*, ASPLOS-29, 2024.

[3] Alberto Ros and Stefanos Kaxiras. *Non-speculative store coalescing in total store order*, ISCA-45, 2018.


Lex order
C
Necesita
rw
Estado
rw
Cerrojo
Sí

Objetivos e implementación

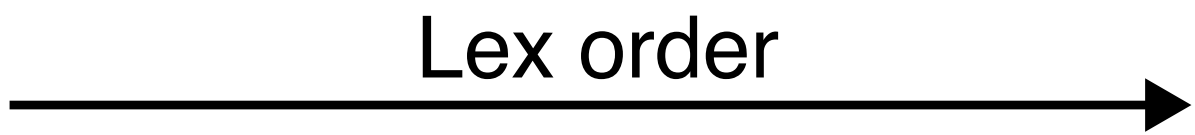
Cerrojos dinámicos en lex order



	C	D
Necesita	rw	r
Estado	rw	r
Cerrojo	Sí	Sí

Objetivos e implementación

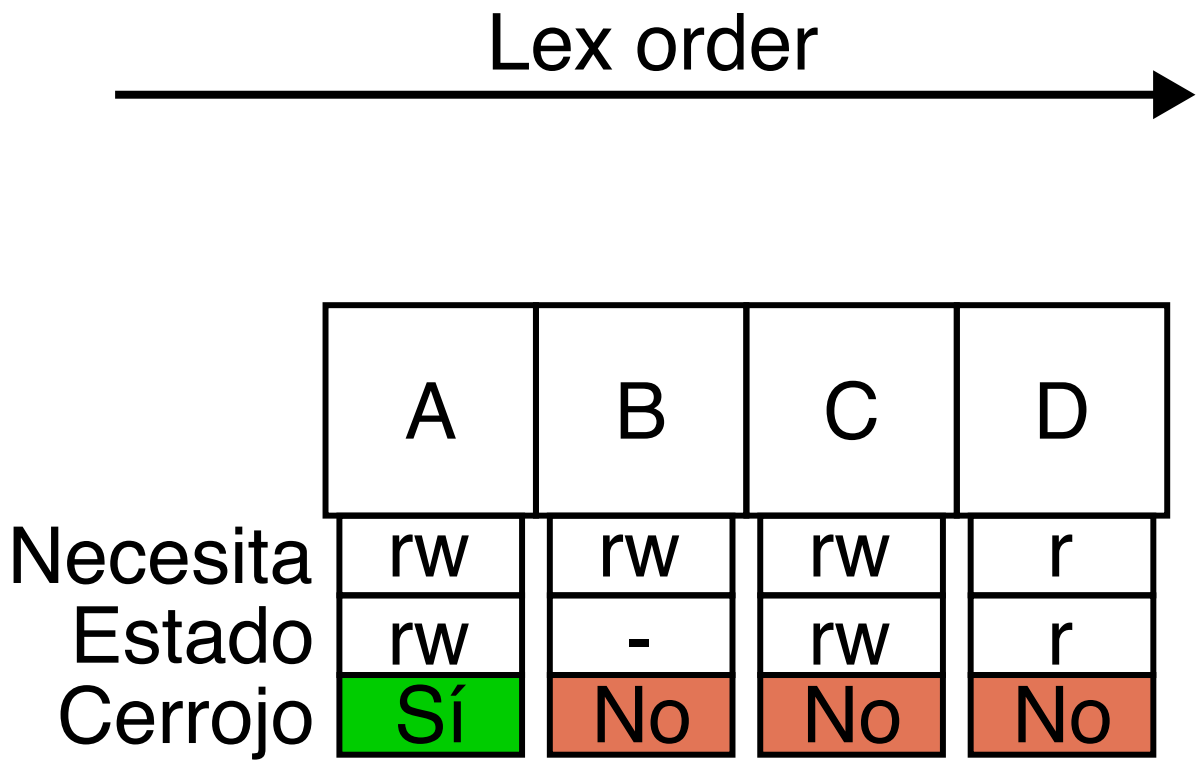
Cerrojos dinámicos en lex order



	B	C	D
Necesita	rw	rw	r
Estado	-	rw	r
Cerrojo	No	No	No

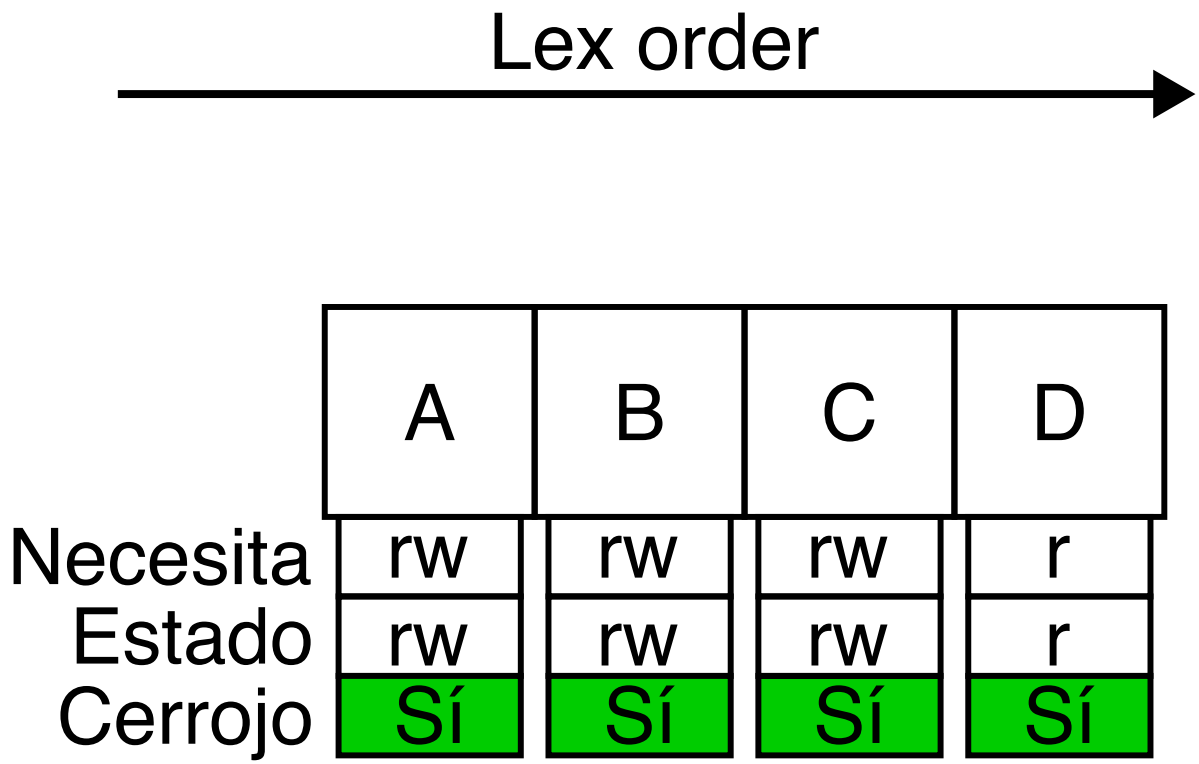
Objetivos e implementación

Cerrojos dinámicos en lex order



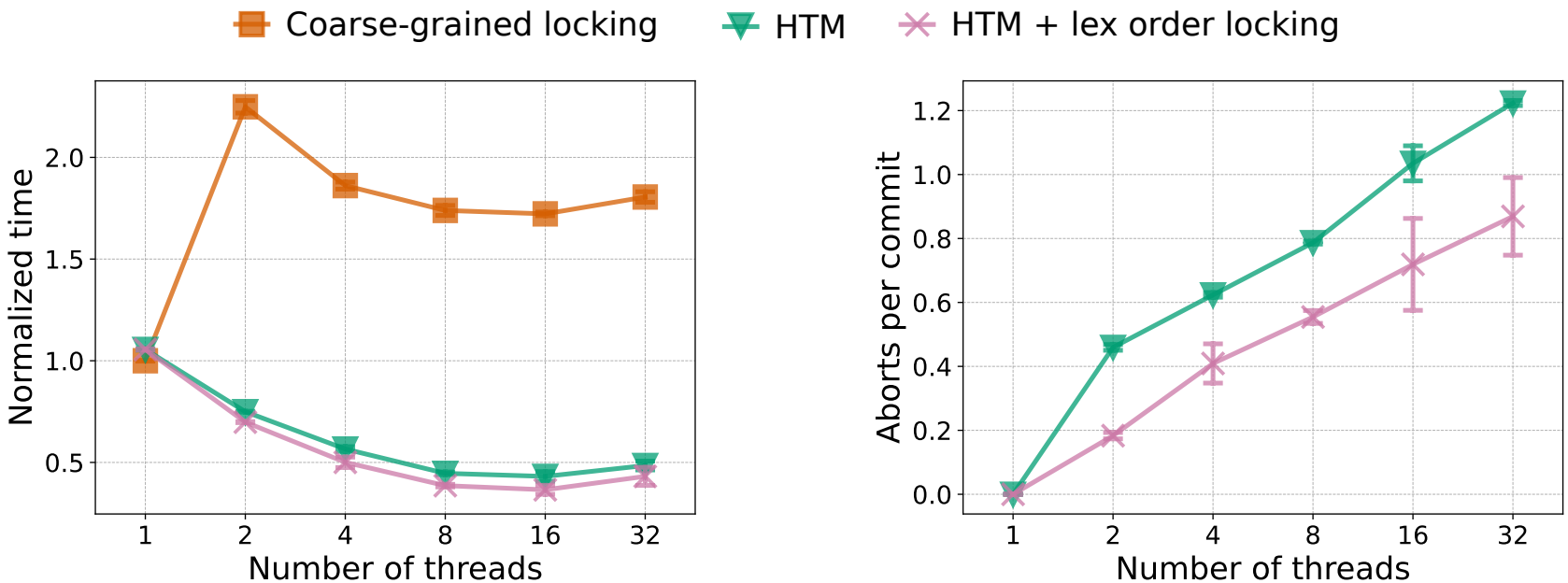
Objetivos e implementación

Cerrojos dinámicos en lex order



Evaluación y resultados

Estructuras de datos

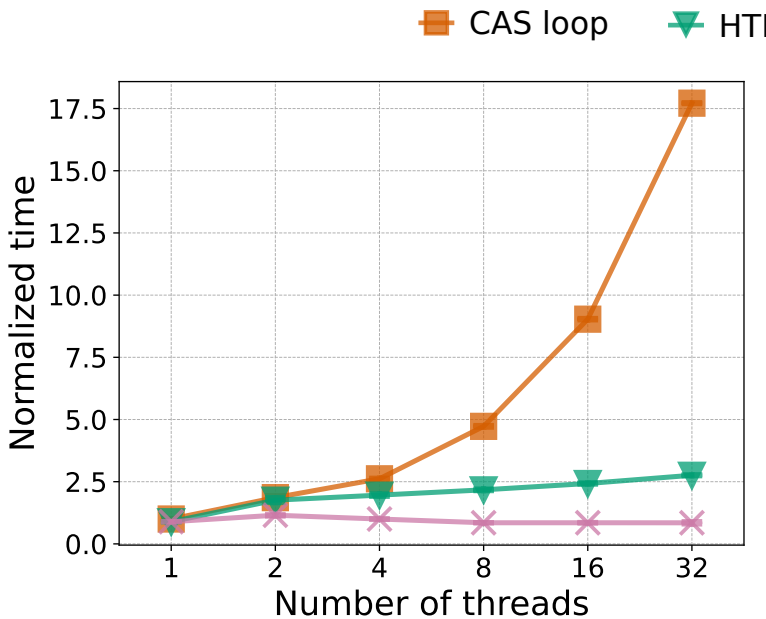


(A) Media geométrica de tiempos de ejecución normalizados

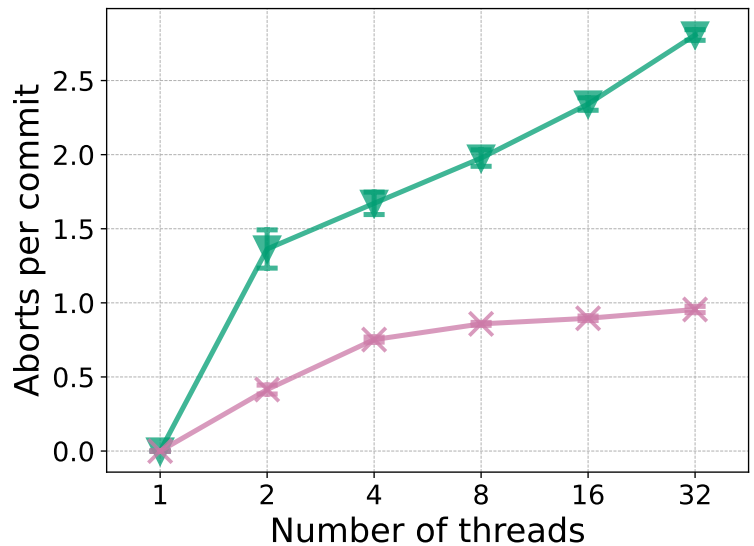
(B) Media de *aborts* por cada *commit*

Evaluación y resultados

Fetch-add double



(A) Tiempo de ejecución normalizado



(B) Número de *aborts* por *commit*

Conclusión

- Implementación requiere pocas modificaciones

Conclusión

- Implementación requiere pocas modificaciones
- Eficiente en transacciones pequeñas

Conclusión

- Implementación requiere pocas modificaciones
- Eficiente en transacciones pequeñas
- Cerrojos en lex order reducen *aborts* significativamente

MEMORIA TRANSACCIONAL "IN-CORE" CON CERROJOS EN LEX ORDER

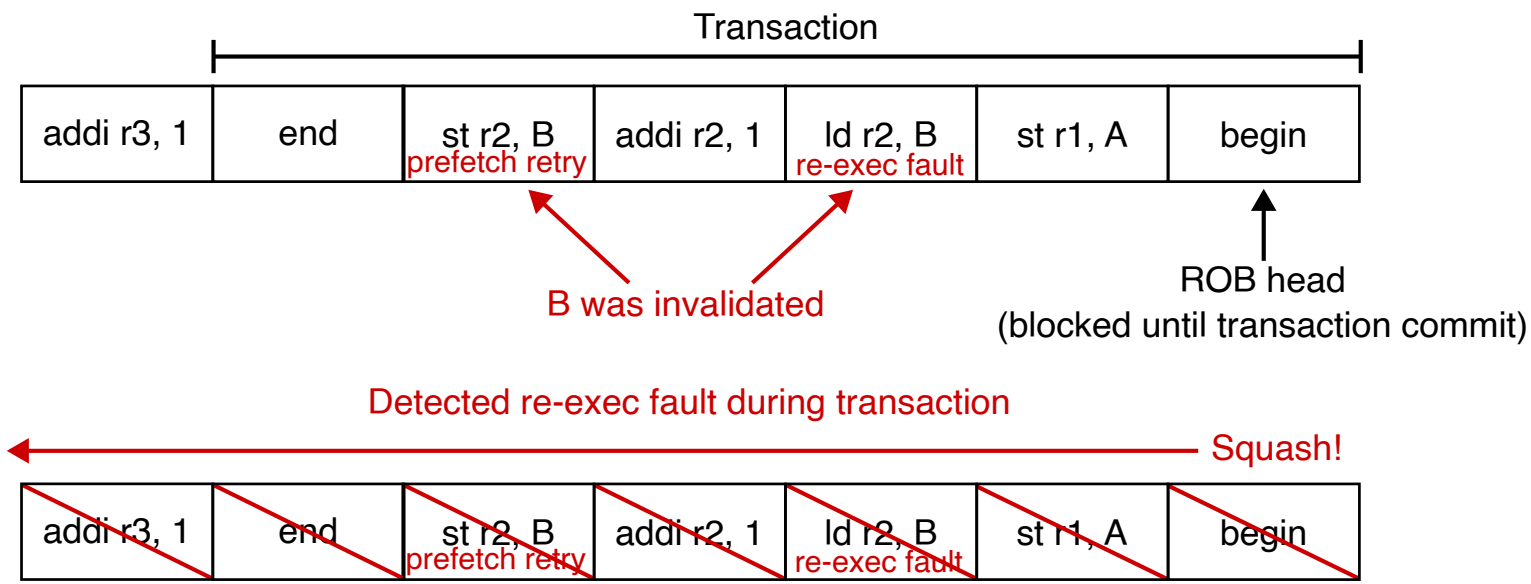
Álvaro Rubira García

UNIVERSIDAD DE MURCIA

Camino alternativo (fallback)

- HTM no garantiza que la transacción se vaya a completar (es *best-effort*)
  - Excepciones
  - Contención
- Se suele incluir un camino alternativo para garantizar el progreso por software
  - Cerrojo convencional que serializa la ejecución

ABORT

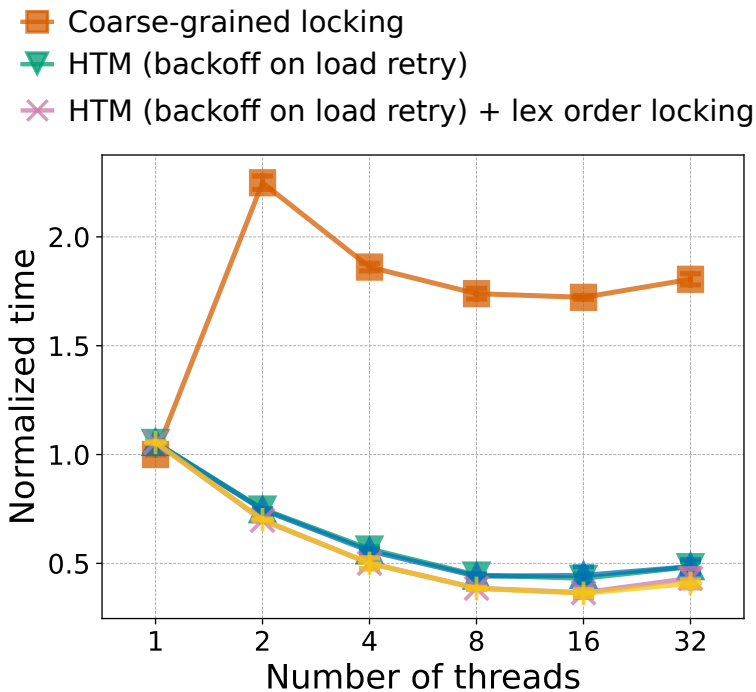

Transaction
addi r3, 1
end
st r2, B
prefetch retry
addi r2, 1
ld r2, B
re-exec fault
st r1, A
begin
B was invalidated
ROB head
(blocked until transaction commit)
Detected re-exec fault during transaction
Squash!
addi r3, 1
end
st r2, B
prefetch retry
addi r2, 1
ld r2, B
re-exec fault
st r1, A
begin

Benchmarks

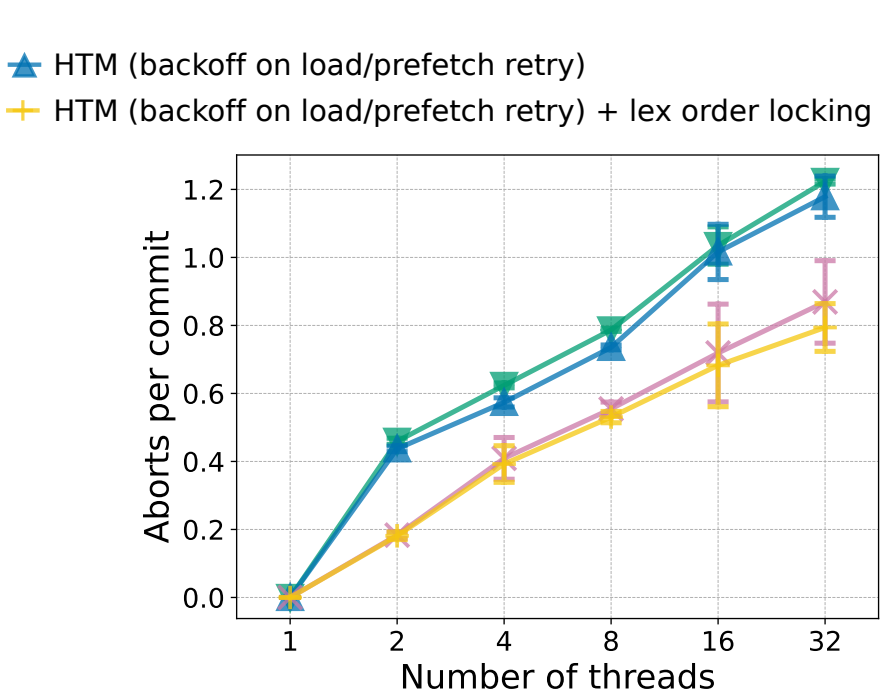
- Estructuras de datos concurrentes
  - Una única transacción por operación
    - *Deque*
    - *Queue*
    - *Stack*
    - *Array* (intercambio atómico de elementos)
  - Varias transacciones por operación
    - Lista enlazada
    - *Hash-map*
    - Árbol binario de búsqueda
- Atómicos especializados
  - *Fetch-add double*
  - *Atomic max double*

BENCHMARKS

Resumen estructuras de datos



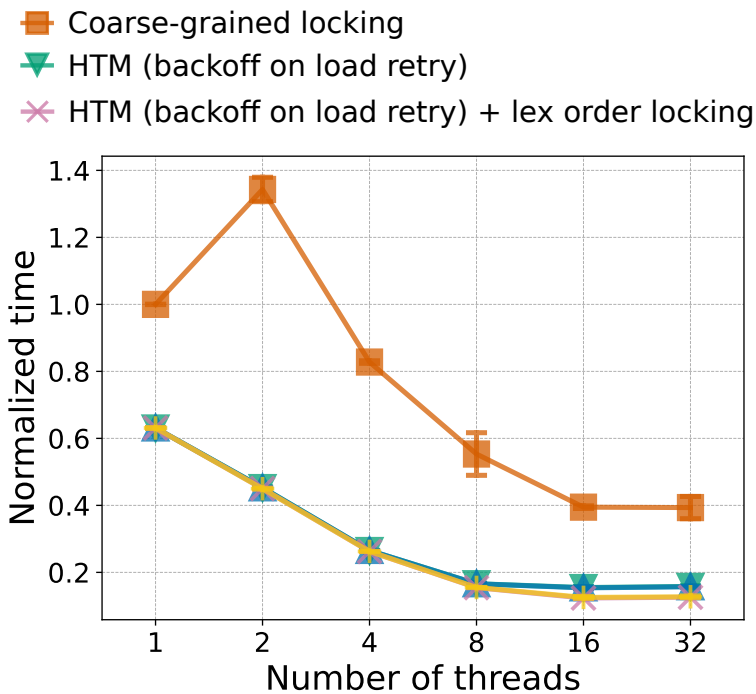
(A) Tiempo de ejecución normalizado



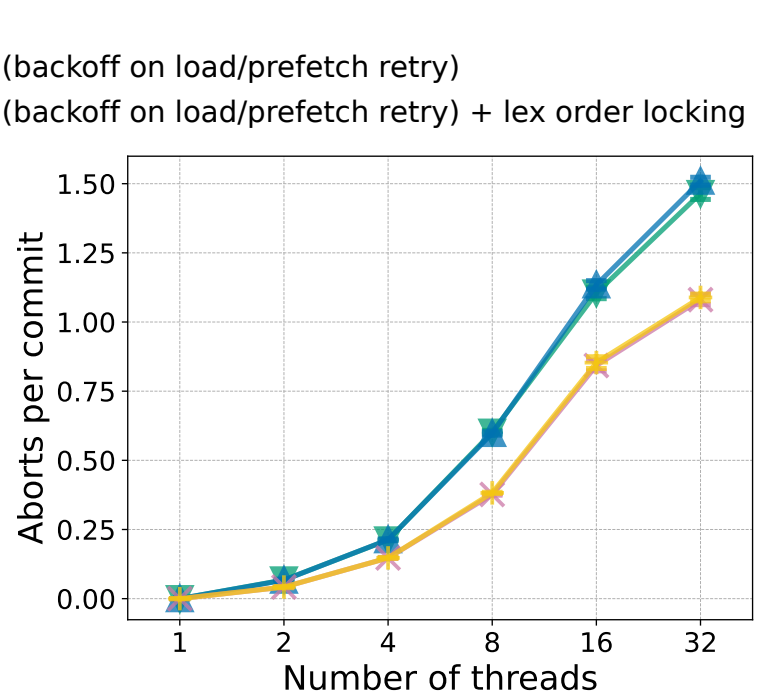
(B) Número de *aborts* por *commit*

BENCHMARKS

Arrayswap



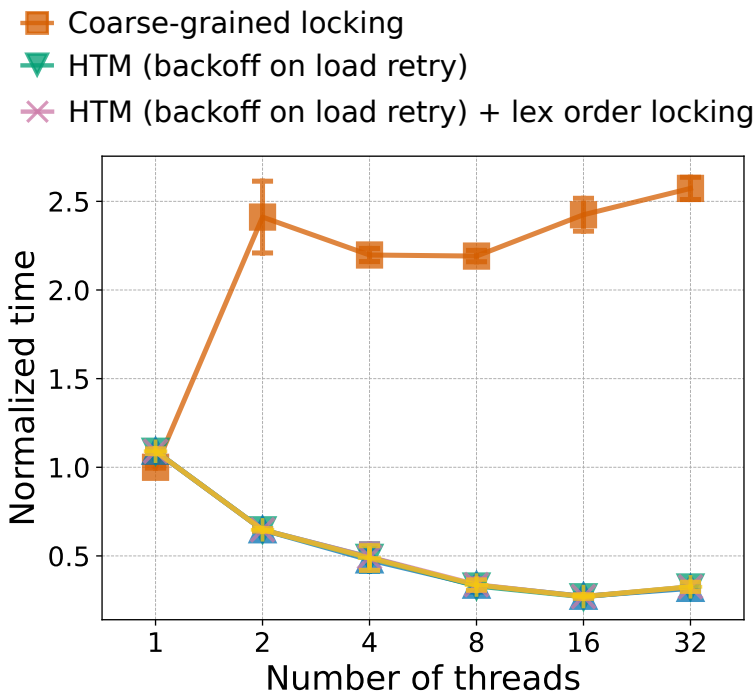
(A) Tiempo de ejecución normalizado



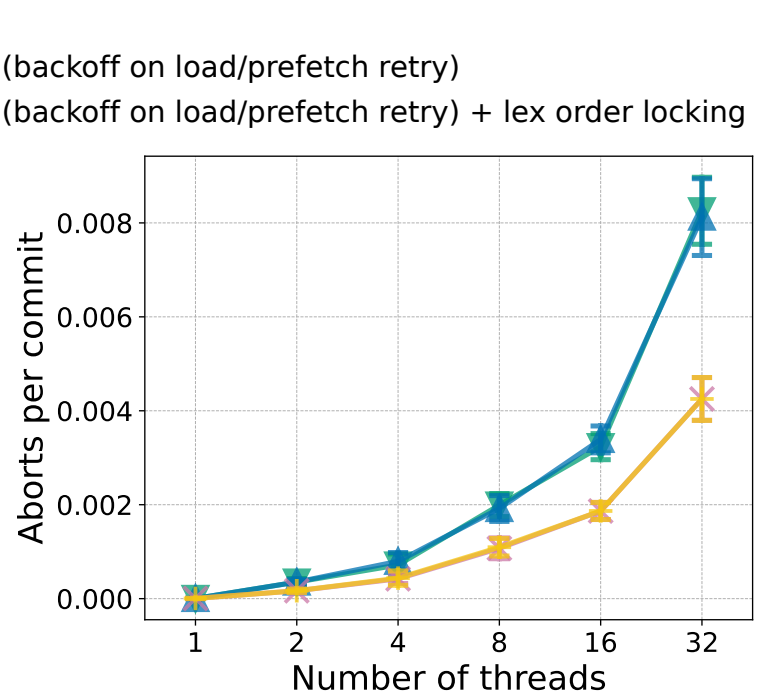
(B) Número de *aborts* por *commit*

BENCHMARKS

Árbol de búsqueda binario



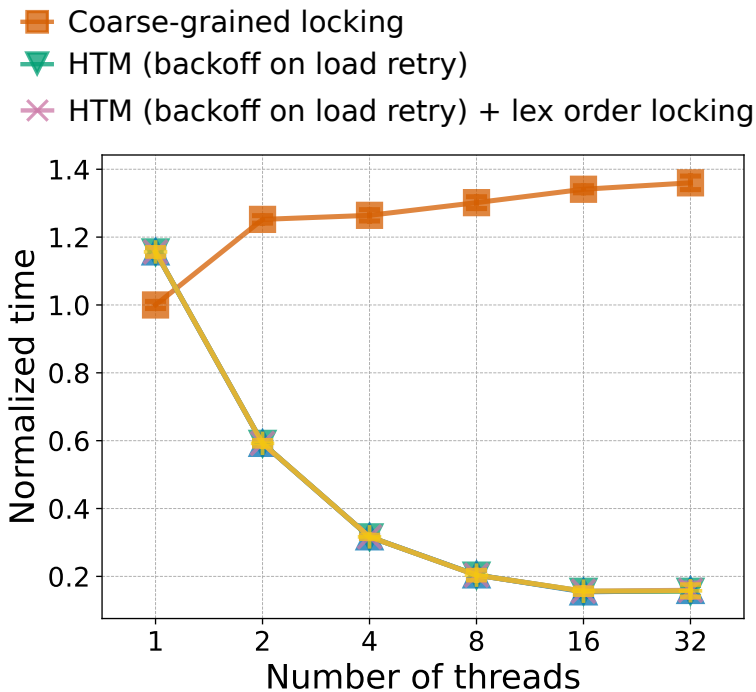
(A) Tiempo de ejecución normalizado



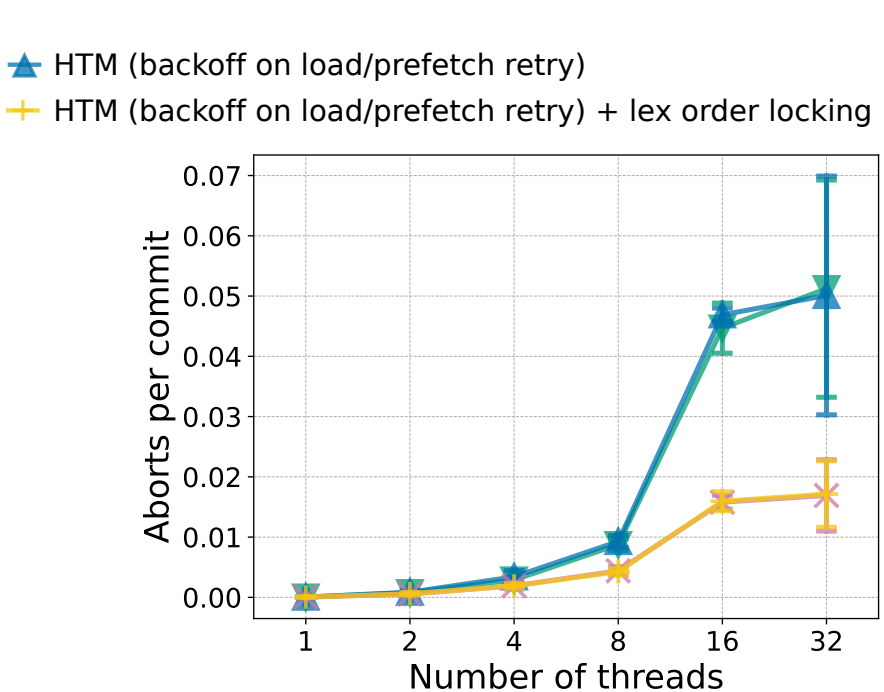
(B) Número de *aborts* por *commit*

BENCHMARKS

Lista ordenada doblemente enlazada



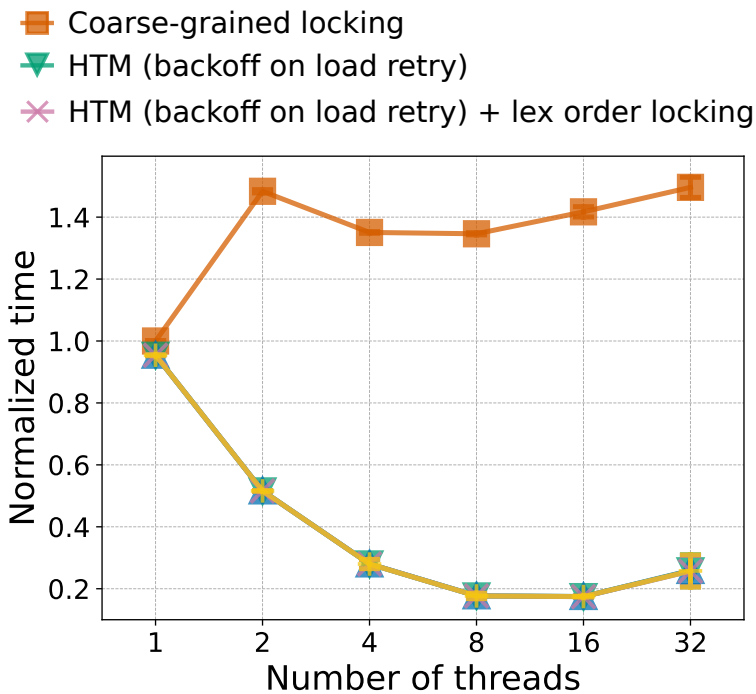
(A) Tiempo de ejecución normalizado



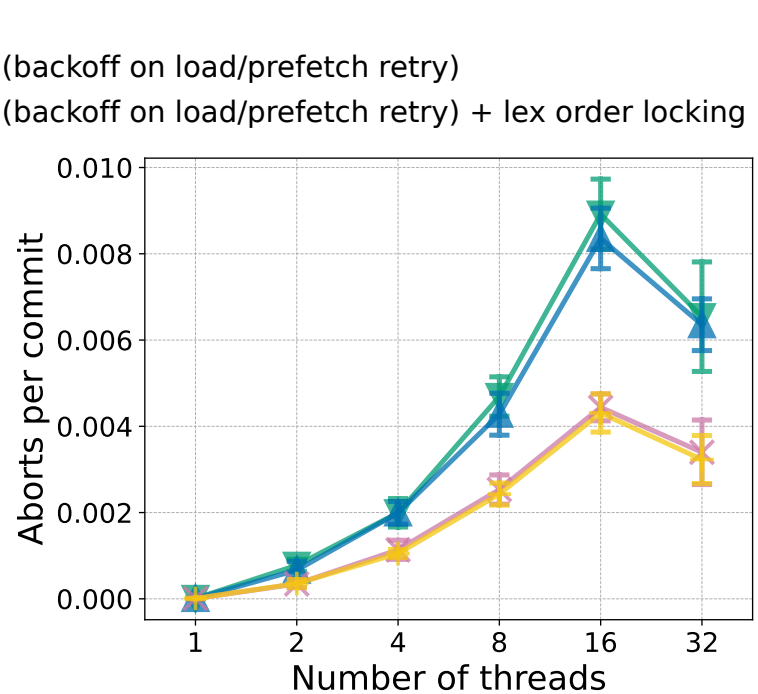
(B) Número de *aborts* por *commit*

BENCHMARKS

Hash map



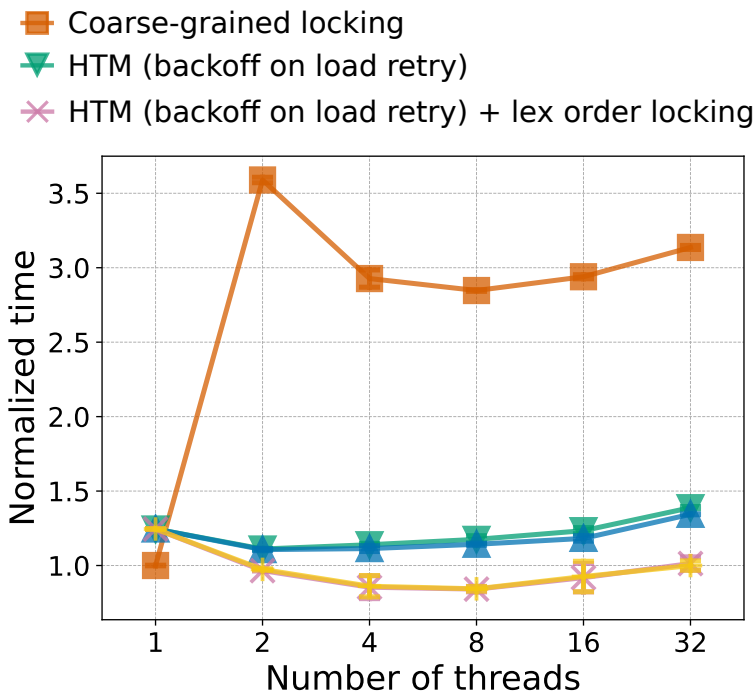
(A) Tiempo de ejecución normalizado



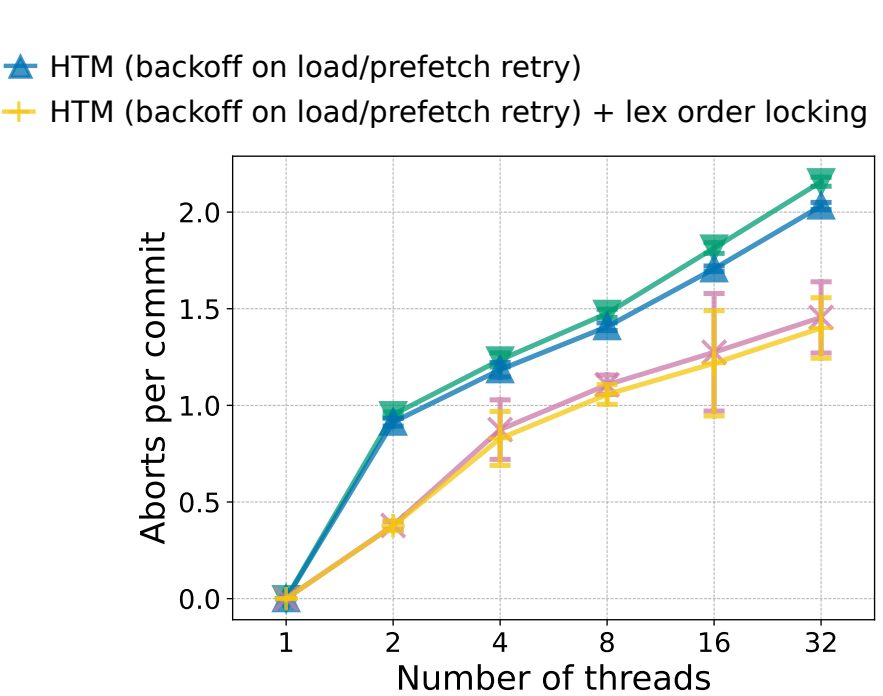
(B) Número de *aborts* por *commit*

BENCHMARKS

Deque



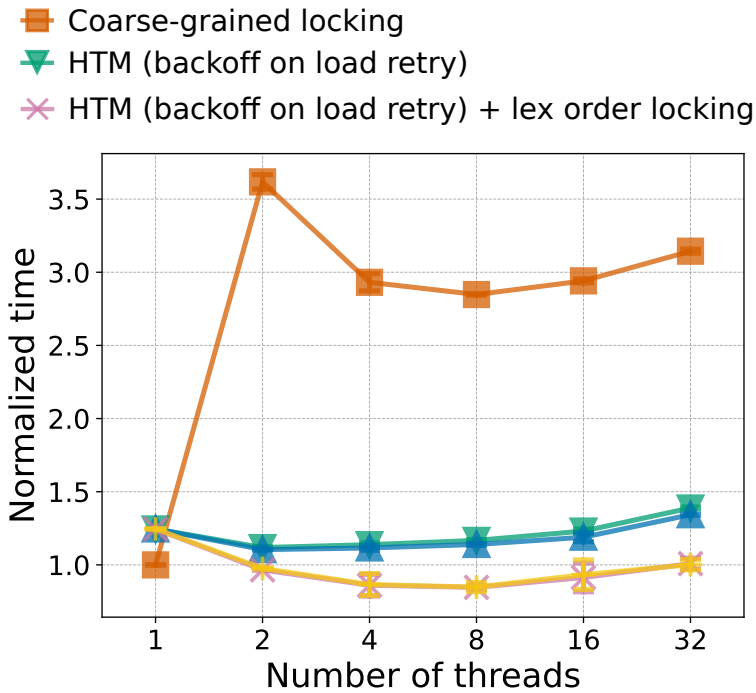
(A) Tiempo de ejecución normalizado



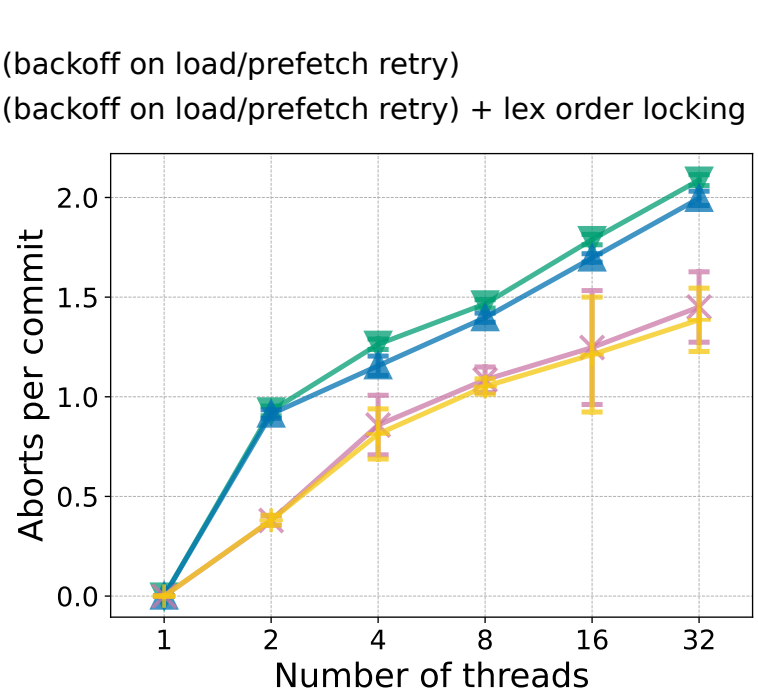
(B) Número de *aborts* por *commit*

BENCHMARKS

Queue



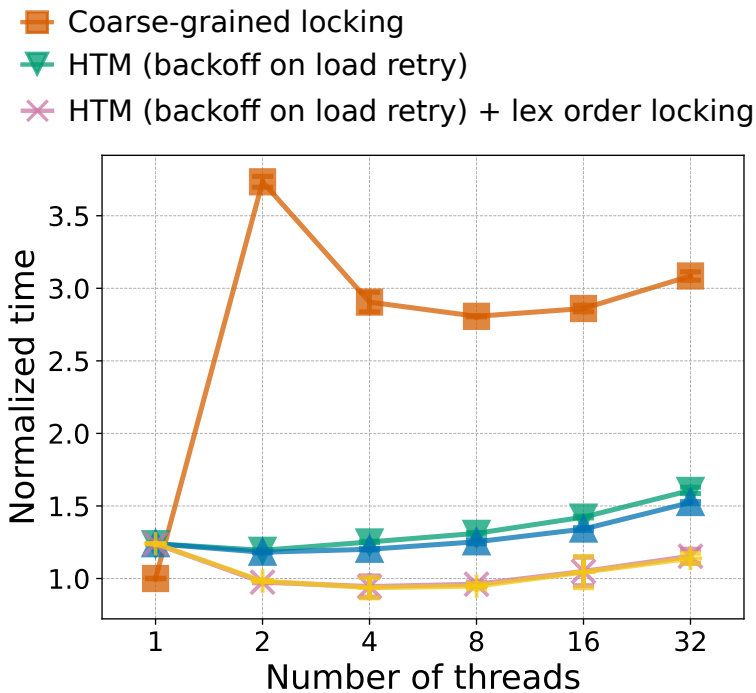
(A) Tiempo de ejecución normalizado



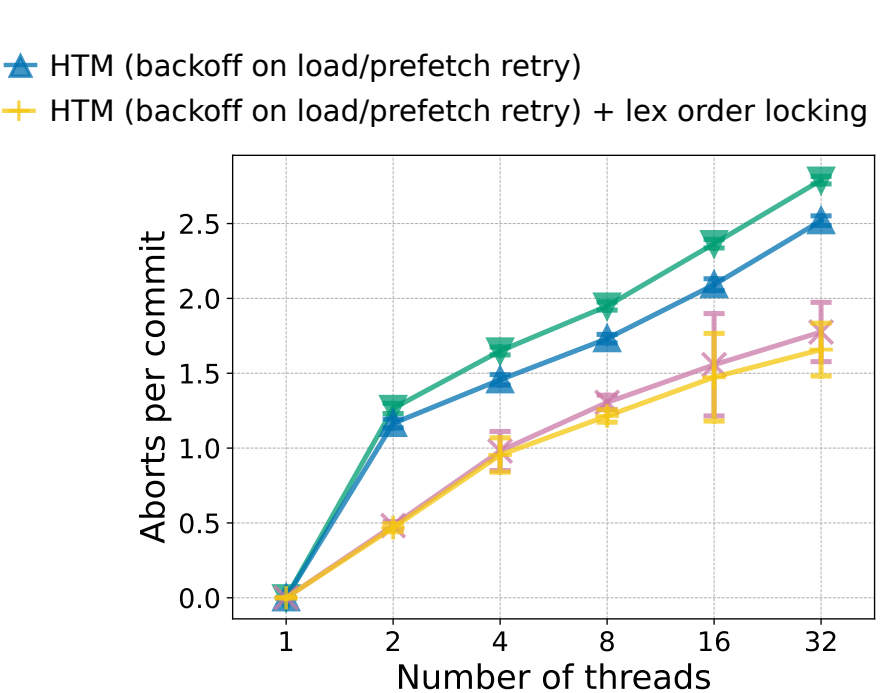
(B) Número de *aborts* por *commit*

BENCHMARKS

Stack



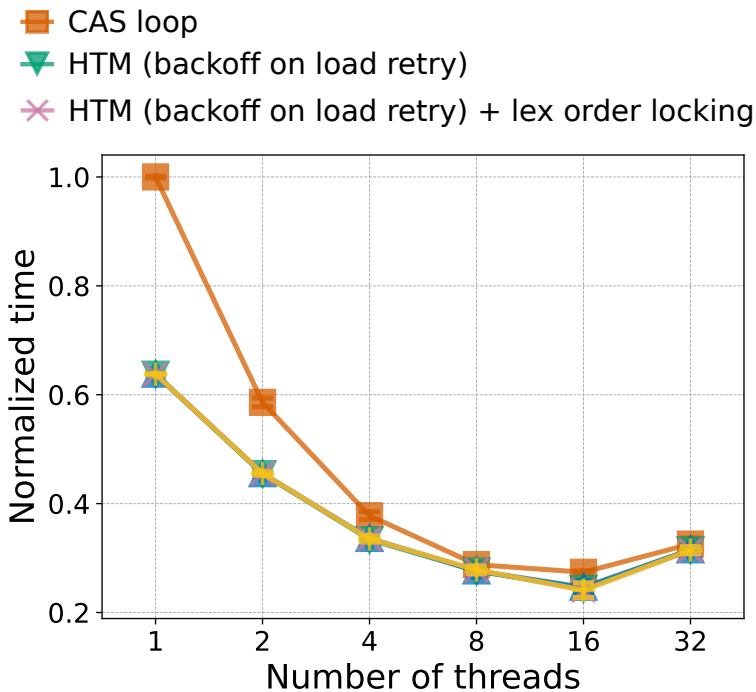
(A) Tiempo de ejecución normalizado



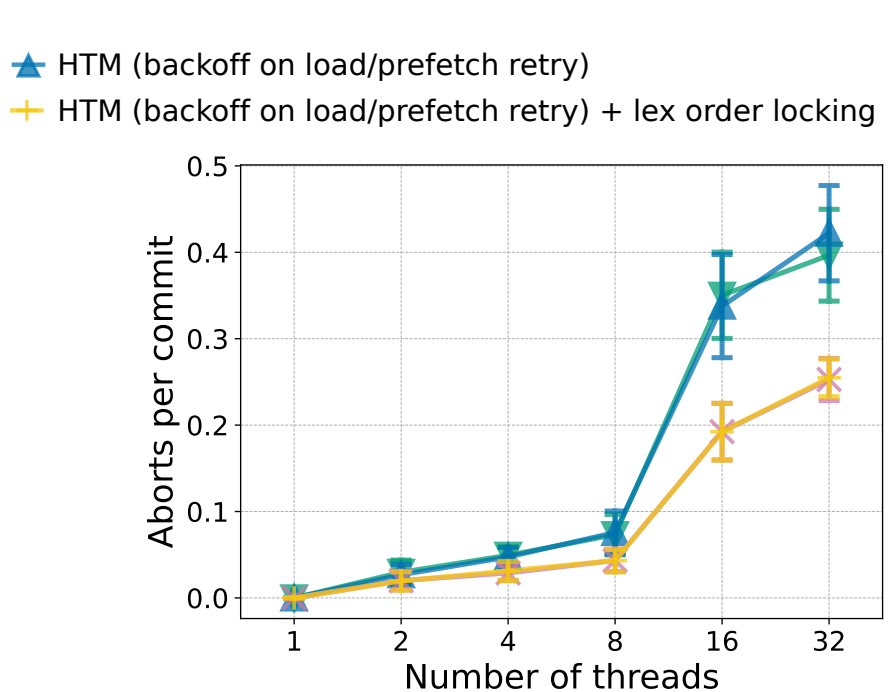
(B) Número de *aborts* por *commit*

BENCHMARKS

Atomic max double



(A) Tiempo de ejecución normalizado



(B) Número de *aborts* por *commit*