

A Simple, Fast and Scalable Non-Blocking Concurrent FIFO Queue for Shared Memory Multiprocessor Systems

Philippas Tsigas Yi Zhang

Department of Computing Science
Chalmers University of Technology

Talk Outline

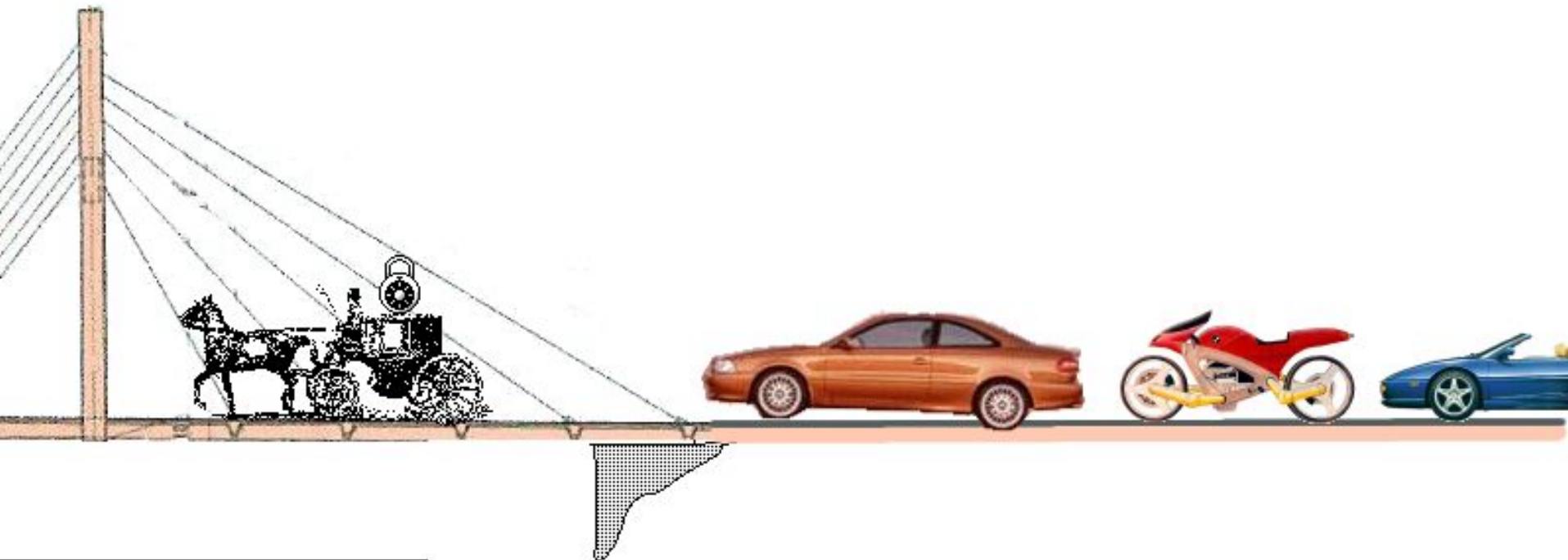
- Synchronization in shared memory multiprocessors
- Non-blocking Queue
 - ABA Problem
 - Performance issues
- Conclusions

Mutual Exclusion

- Traditional way for synchronization.
- Performance degradation under high contention.
 - Network contention
 - Lock convoy
- Complex (pessimistic) scheduling analysis
 - Priority inversion
 - Deadlock

Lock Convoy

* *Slowdown of one process may cause the whole system slowdown*



Non-blocking Synchronization

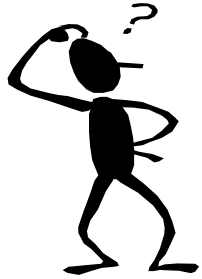
- An alternative approach for synchronization
- Lock-free and Wait-free
- Better performance in multiprocessor systems
 - No lock convoy
 - No priority inversion
 - No deadlock

Non-blocking Queue

Previous Results

- Designed for Asynchronous Shared Memory Multiprocessors
- Previous work
 - Lamport (1983)
 -
 - Michael and Scott (1998)
 -





Non-blocking Queue

Our Results



- The new non-blocking queue outperforms the best known alternative implementation.
 - + the new solution to the ABA problem together with
 - + the lazy pointer updating to improve performance
 - + the algorithmic design of the queue as a cyclic array

ABA or Pointer Recycling Problem

- Occurs when read-modify-write is used in lock-free computing with CAS atomic primitive
- Drawback of the CAS atomic primitive
- A Lot of overhead introduced to solve it

The Specification of CAS

```
Boolean CAS(int *mem, register old, new)
{ temp = *mem;
  if (temp == old) {
    *mem = new;
    return (TRUE);}
  else
    return FALSE;
}
```

ABA Problem (Example)

- Array-based Queue (Enqueue)
 1. Loop
 2. `head = Queue.head`
 3.
 4. if `CAS(Queue.array[head],NULL,data)`
 5.
 6. End loop

ABA Problem (Example)

- Array-based Queue (Dequeue)
 1. Loop
 2. `tail = Queue.tail`
 3.
 4. `if CAS(Queue.array[tail],data,NULL)`
 5.
 6. End loop

ABA Problem (Example)

Execution History (Empty Queue)

- P1
 - Enqueue 2
 -
 - (Preempted)
 -
 - Enqueue 4
- P2
 - Enqueue
 -
 - Dequeue

ABA Problem

Traditional Solution

- Using version number
 - Splits word into two parts
 - Uses one part of the word as a version number
 - Increases the version number of the word whenever updating the word
- Not a complete solution, ABA can still happen, when the version number runs out of space.

ABA Problem

Traditional Solution (Drawbacks)

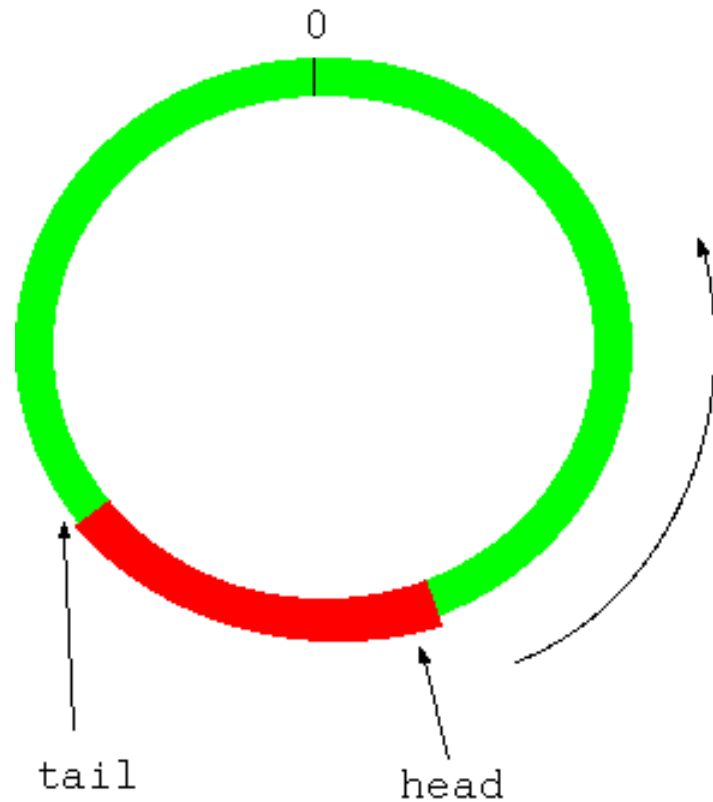
- The actual pointer length is smaller than the system pointer length
 - Programmers must manage the pointer (memory) themselves
 - Limits the memory that can be accessed
- Tag operations introduce extra overhead

ABA Problem

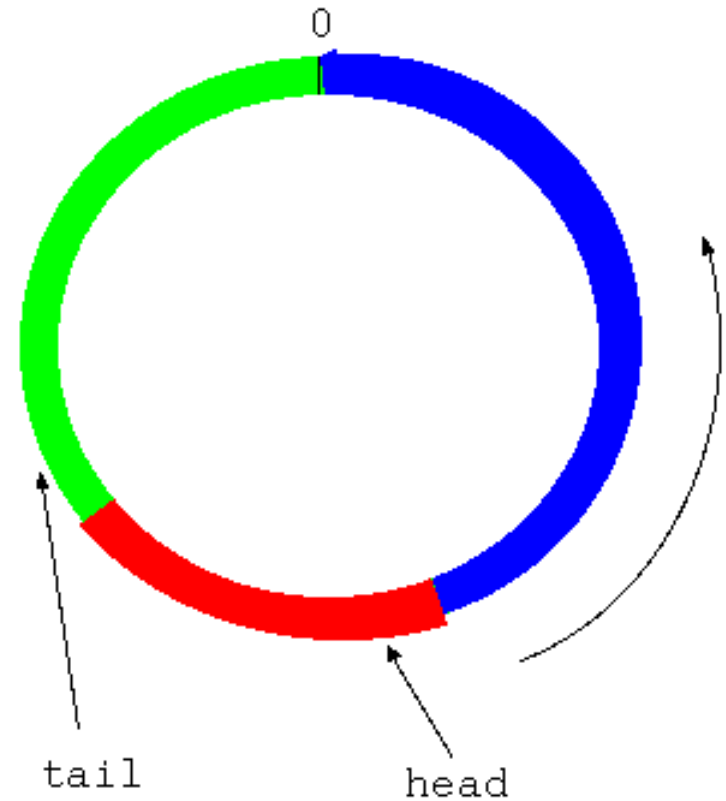
Our Solution

- Introduce a ghost copy of each value and turn ABA to ABA'B' A
- For example, NULL means empty in the Queue implementation
- Using NULL(0) and NULL(1) mean empty cell
- Recycle the NULL values
- More NULL values can be introduced.


Queue using Cyclical Array



Normal Cyclic Queue



Cyclic Queue with two Null

 Occupied Cells

  Empty Cells

ABA Problem

Execution History (Empty Queue)

- P1
 - Enqueue 2

 - (Preempted)

 - Enqueue 4
- P2
 - Enqueue at pos A
 -
 - Dequeue at pos A
 -
 - Enqueue at pos A
 -
 - Dequeue at pos A
 -

Performance Issues of Synchronization

- Network contention
 - Access to shared memory
 - Spinning on shared memory
 - Cache coherent protocols
- Lock convoys

Mutual Exclusion Based Solutions for Performance Issues

- Avoiding network contention
 - Ticket lock
 - MCS Queue lock
- Avoiding lock-convoy effect
 - Scheduler-conscious synchronization

Non-blocking and Performance

- Avoiding the performance problems of lock-convoy from the beginning
- No much consideration about network contention

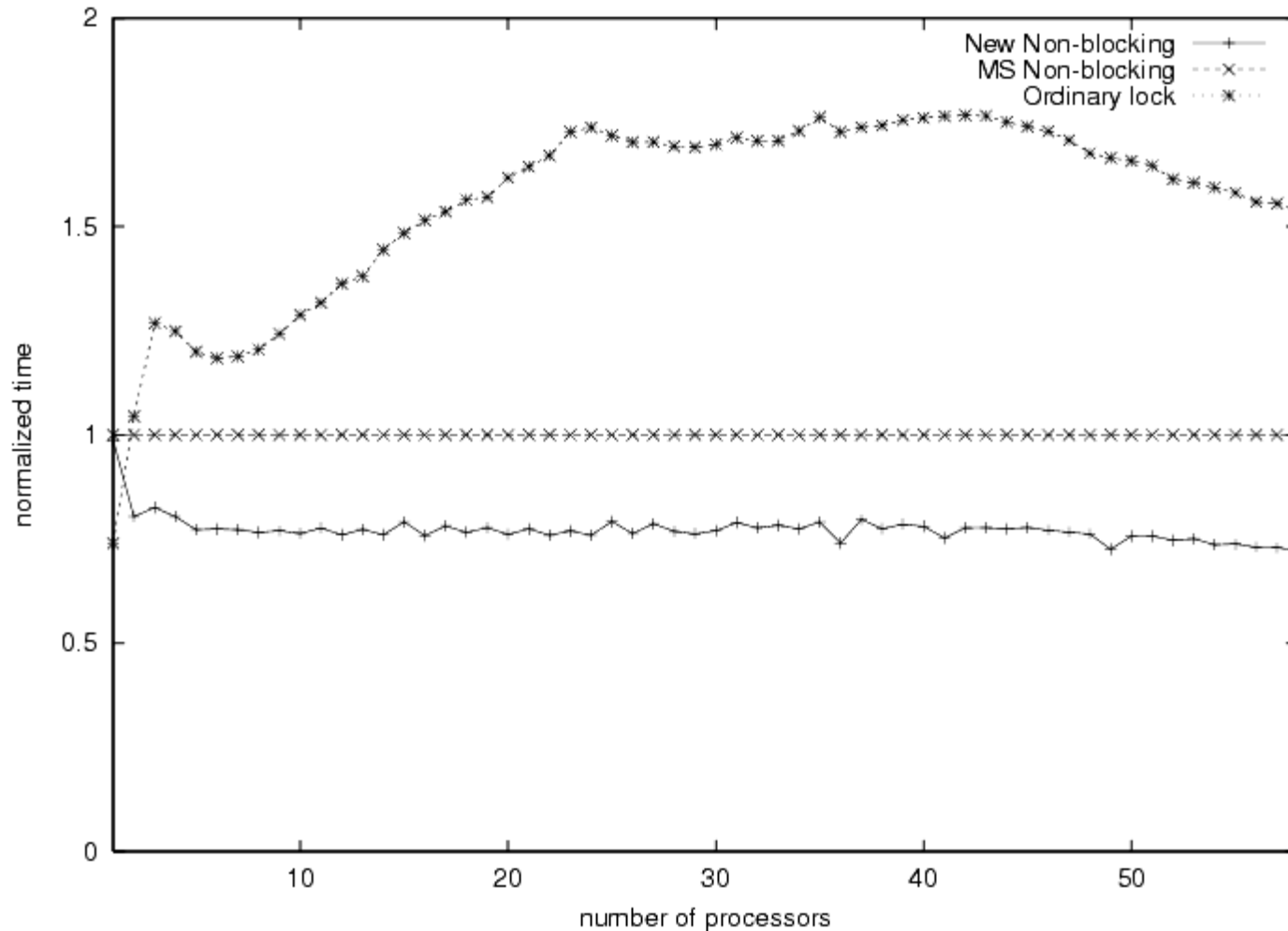
Observations on Queue operations

- CAS operations are network operations; they generate a lot of traffic
- CAS operations are used when changing the head and tail of the queue
- Head and tail of the queue do not have to always point to the actual head and tail of the queue in a non-blocking implementation (Does it?)

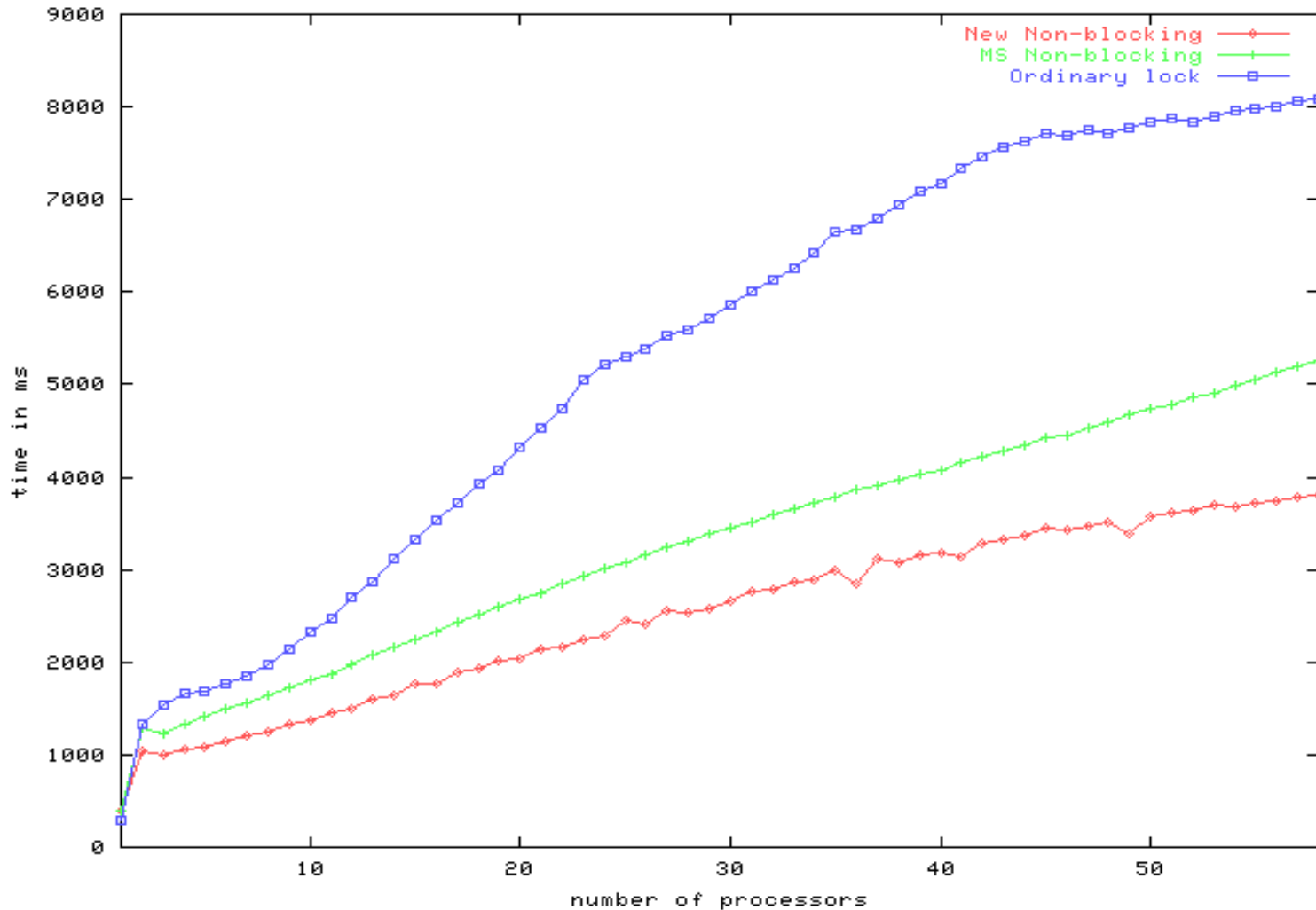
Our Approach: Sketch

- Let the head and tail pointers lag behind the actual head and tail
- Use computation to calculate the actual head and tail
- Trade-off between the computation time for finding the actual head/tail and the synchronization time for keeping head/tail lag not to lag much behind.

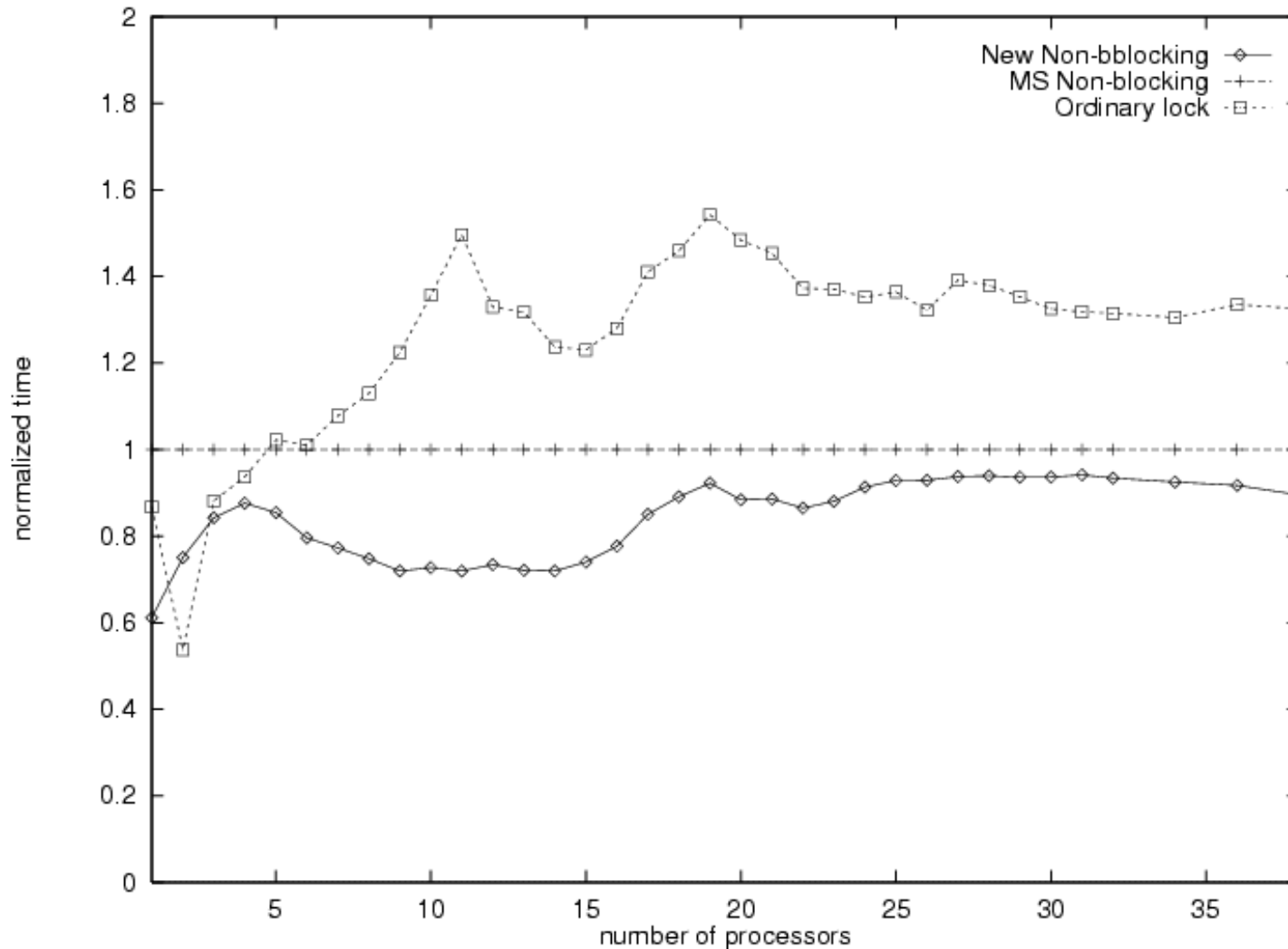
Results on SUN Enterprise 10000 with Full Contention



Results on SUN Enterprise 10000 with Full Contention



Results on SGI Origin 2000 with Full Contention



Conclusion

- A new non-blocking concurrent FIFO queue algorithm is present.
 - A simple mechanism for easing ABA problem is proposed.
 - A mechanism to lower the contention of non-blocking operations is introduced
- ew algorithm perform very well under UMA MA and ccNUMA machines.