

Algorithms for Scalable Synchronization on Shared Memory Multiprocessors

John Mellor-Crummey and Michael L. Scott

Greg Wilbur

February 2, 2009

Locks

- `test_and_set` lock
- Ticket Lock
- Array Based Queueing Locks
- MCS Lock

Barriers

- Centralized Barriers
- Software Combining Tree Barrier
- Dissemination Barrier
- Tournament Barriers
- A New Tree-based Barrier

Provide a means for achieving mutual exclusion.

How do you compare Locks?

- Network traffic
- Size
- Fairness
- Single Processor Latency
- Primitives Used
- Scalability

test_and_set lock (p4)

Each processor spins on the same value.

Possible improvements:

- Test-and-test_and_set
- Backoff (constant/linear/exponential?)

test_and_set lock (p4)

Pros:

- $O(1)$ space
- Only test_and_set required

Cons:

- High network traffic
- Not fair

Ticket Lock (p5)

Two integers

- `now_serving`
- `next_ticket`

Backoff

- Exponential BAD
- Constant works well

Ticket Lock (p5)

Pros:

- $O(1)$ space
- FIFO

Cons:

- `fetch_and_increment` required
- High network traffic

Array-based Queueing Locks (p6)

Each processor spins on a different location in a different cache line.

Anderson's uses `fetch_and_increment`.

Graunke and Thakkar's uses `fetch_and_store`.

Array-based Queueing Locks (p6)

Pros:

- $O(1)$ Network traffic (on cache-coherent machines)
- FIFO

Cons:

- `fetch_and_increment` or `fetch_and_store` required
- $O(P)$ space

The MCS Lock (p8)

Keeps track of a queue of the processors that are waiting on a lock.

Why do we need two spins?

The MCS Lock (p8)

Pros:

- $O(1)$ space
- $O(1)$ Network traffic (on cache-coherent machines)
- FIFO

Cons:

- `fetch_and_store` required (`compare_and_swap` recommended)

Scalability of Locks

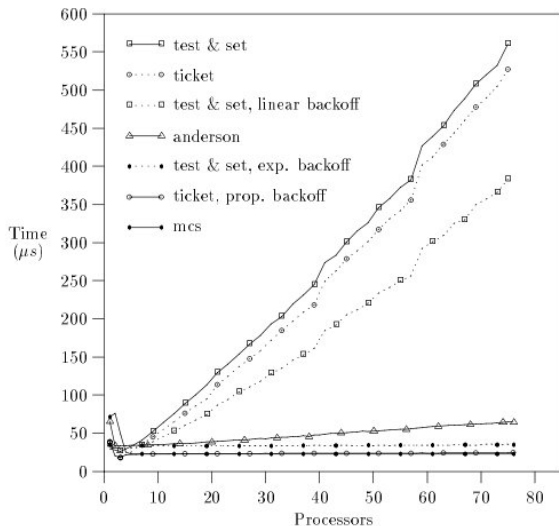


Figure 4: Performance of spin locks on the Butterfly (empty critical section).

Scalability of Locks

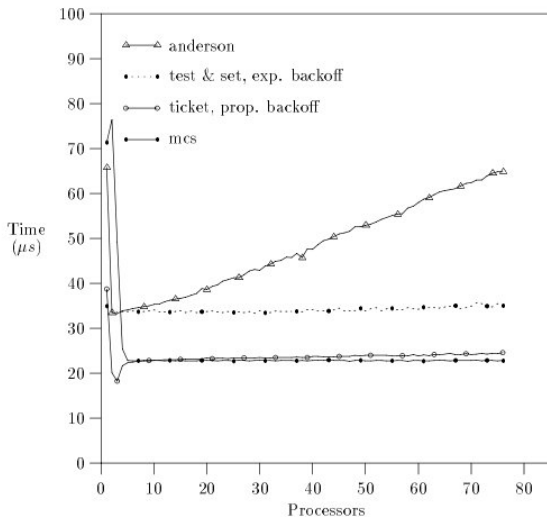


Figure 5: Performance of selected spin locks on the Butterfly (empty critical section).

Scalability of Locks

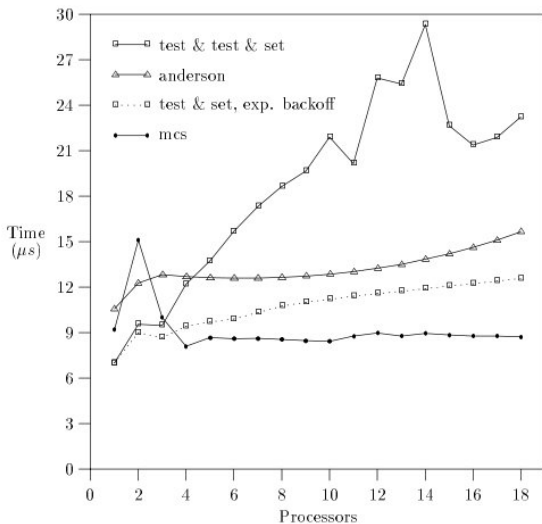


Figure 6: Performance of spin locks on the Symmetry (empty critical section).

Used to make sure that all processes complete one section before any one proceeds to the next.

How do you compare barriers?

- Network traffic
- Size
- Processor Latency
- Primitives Used
- Scalability

Centralized Barriers (p12)

Each processor decrements a number then spins on the same shared variable.

Improvements

- Sense reversing
- Backoff scheme?

Centralized Barriers (p12)

Pros:

- $O(1)$ space required

Cons:

- High network traffic
- $O(P)$ critical path
- `fetch_and_decrement` required

Software Combining Tree Barrier (p13)

Each processor is a leaf in a tree. The last child to arrive at the barrier signals the parent.

Sense reversing helps here too.

Software Combining Tree Barrier (p13)

Pros:

- $O(\log P)$ critical path

Cons:

- $O(P \log P)$ space required
- High network traffic
- `fetch_and_decrement` required

Dissemination Barrier (p15)

In the k th round ($0 \leq k < \log(P) - 1$), processor i first signals processor $i + 2^k$, then waits for a signal from processor $i - 2^k$.

Improvements

- Alternating sets of variables
- Sense reversal

Dissemination Barrier (p15)

Pros:

- $O(\log P)$ critical path
- $O(P \log P)$ Network transactions
- No atomic primitives required.

Cons:

- $O(P \log P)$ space

Tournament Barriers (p17)

Much like the combining tree barrier, except the signaler is determined statically, so there is no need for special atomic primitives.

Tournament Barriers (p17)

Pros:

- $O(\log P)$ critical path
- $O(P)$ Network transactions
- No atomic primitives required

Cons:

- $O(P \log P)$ space

A New Tree-Based Barrier (p19)

Each processor gets a tree node which is part of two trees. One for signaling arrival and one for signaling wakeup.

The arrival tree has four children per parent to allow optimizations on 32 bit hardware.

The wakeup tree has two children per parent to minimize the critical path.

A New Tree-Based Barrier (p19)

Pros:

- $2P - 2$ Network transactions (theoretical minimum)
- Critical path proportional to $\log_4 P + \log_2 P$
- No atomic primitives required

Cons:

- $O(P)$ space

Scalability of Barriers

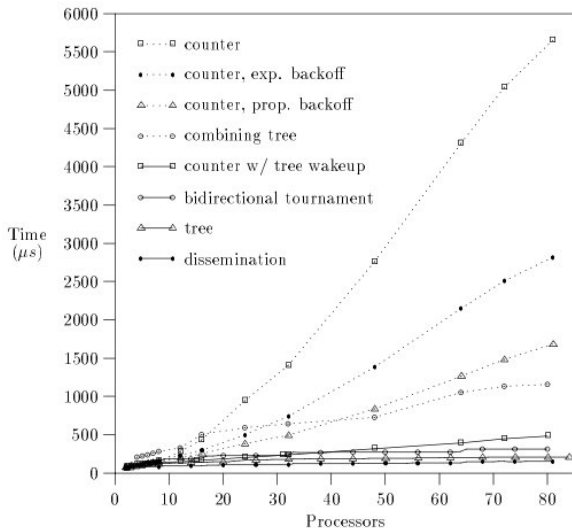


Figure 8: Performance of barriers on the Butterfly.

Scalability of Barriers

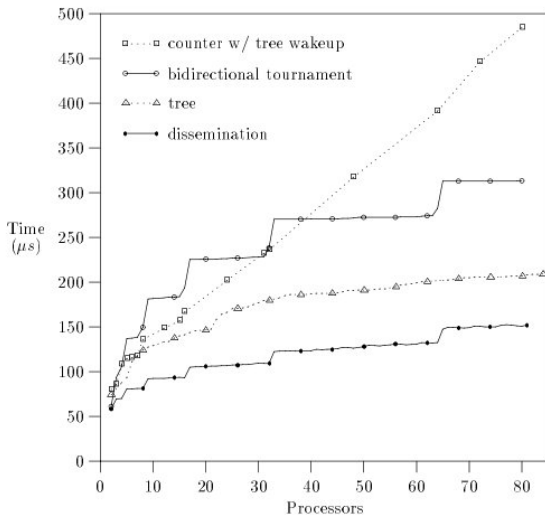


Figure 9: Performance of selected barriers on the Butterfly.

Scalability of Barriers

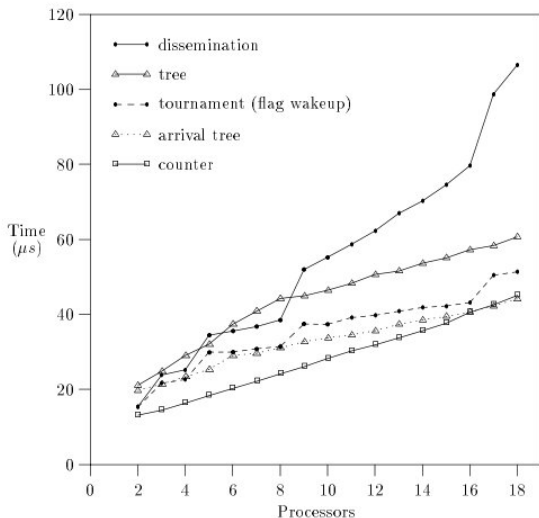


Figure 10: Performance of barriers on the Symmetry.

Conclusion

Locks

- Use the MCS lock
- If you can't use MCS, use the ticket lock
- If you can't use the ticket lock, use the `test_and_set` lock

Barriers

- Cache-coherent: Use a centralized barrier or arrival tree
- Not cache-coherent: Use a dissemination barrier or tree-based barrier