

Spin Lock Alternatives

“The Performance of Spin Lock Alternatives for
Shared-Memory Multiprocessors”

Thomas E Anderson

Introduction

- Spinlocks are commonly used to protect small critical regions
- Problem: On multi-processor machines, spinning consumes memory bandwidth, slowing other processors

Introduction

- Question: Are there more efficient algorithms for spin waiting that work on existing hardware?

Simple approaches to spin waiting

- spin on test-and-set

```
lock := CLEAR;
while (TestAndSet(lock) = BUSY);
/* critical section */
lock := CLEAR;
```

- spin on read

```
lock := CLEAR;
while (lock = BUSY or TestAndSet(lock) = BUSY);
/* critical section */
lock := CLEAR;
```

- ran simulation on Sequent Symmetry Model B
(20 80386 processors)

Simple approaches to spin waiting

performance degrades badly with many processors
spin-waiting

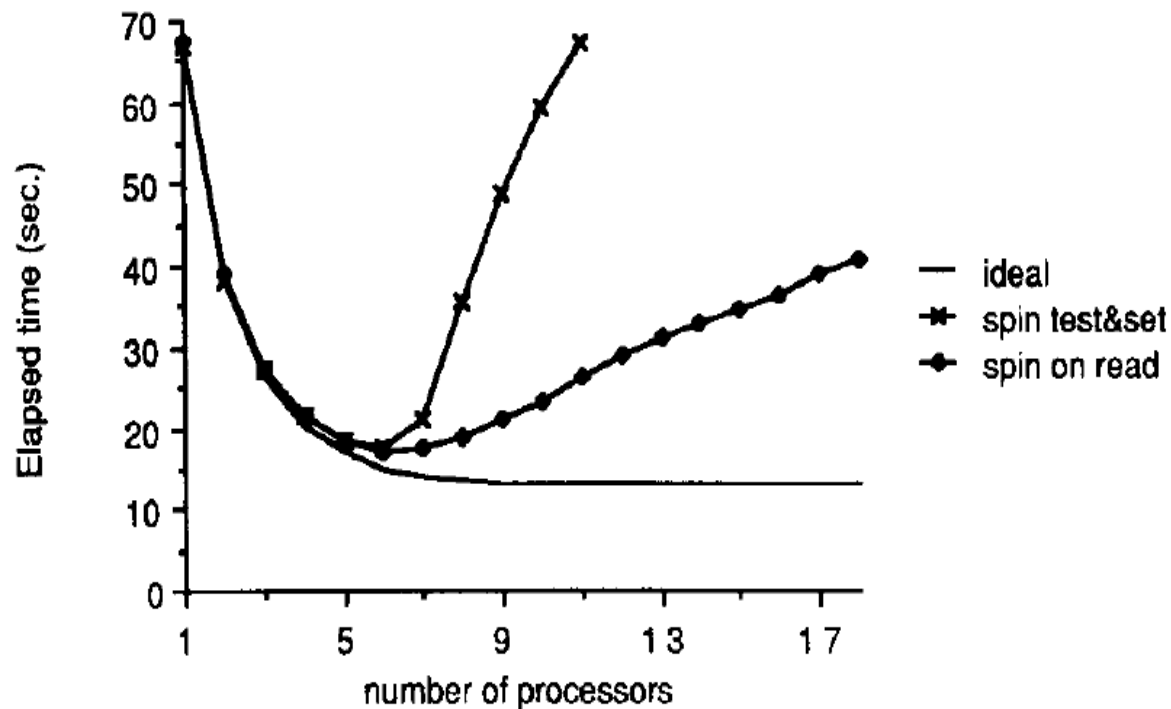


Fig. 1. Principal performance comparison: elapsed time (second) to execute benchmark (measured). Each processor loops one million/ P times: acquire lock, do critical section, release lock, and compute.

Software alternative 1

- delay after spinning processor notices lock has been released
- idea: reduce number of unsuccessful test-and-set ops
- statically assign each processor a delay from 0 to P
- dynamically adjust delay depending on how many collisions experienced (similar to Ethernet)

Software alternative 1

```
while (lock = BUSY or TestAndSet(lock) = BUSY)
  begin
    while (lock = BUSY);
    Delay();
  end;
```

Software alternative 2

- delay after every read of the lock
- idea: limit total communication bandwidth of spinning
- statically or dynamically set delays, similar to alternative 1

Software alternative 2

```
while (lock = BUSY or TestAndSet(lock) = BUSY)  
    Delay();
```

Software alternative 3

- queueing in shared memory
- idea: reduce cache invalidations from spinning processors
- each waiting processor enqueues itself and spins on separate location in memory
- processor leaving critical section notifies next processor in line
- high latency with small number of processors

Software alternative 3

```
Init      flags[0] := HAS_LOCK;
          flags[1...P-1] := MUST_WAIT;
          queueLast := 0;

Lock      myPlace := ReadAndIncrement(queueLast);
          while (flags[myPlace mod P] = MUST_WAIT)
              ;
          flags[myPlace mod P] := MUST_WAIT;

Unlock    flags[(myPlace + 1) mod P] := HAS_LOCK;
```

Performance of software alternatives

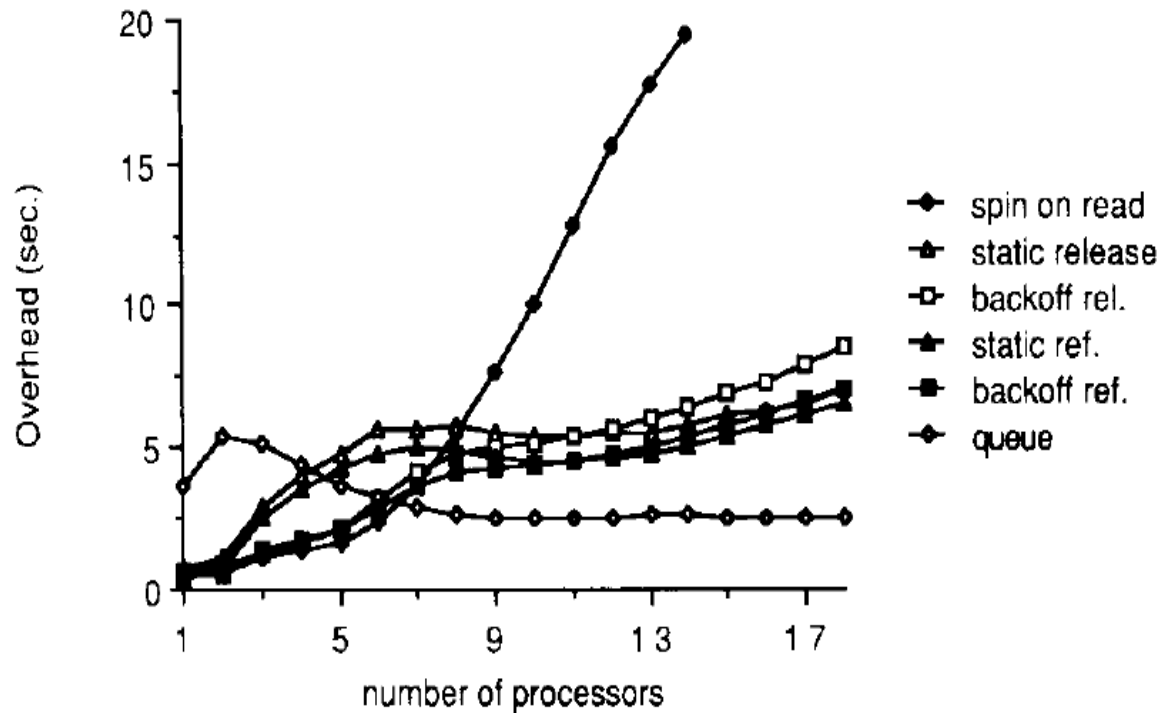


Fig. 3. Principal performance comparison: spin-waiting overhead (seconds) in executing the benchmark (measured). Each processor loops one million/ P times: acquire lock, do critical section, release lock, and compute.

Overhead vs number of slots

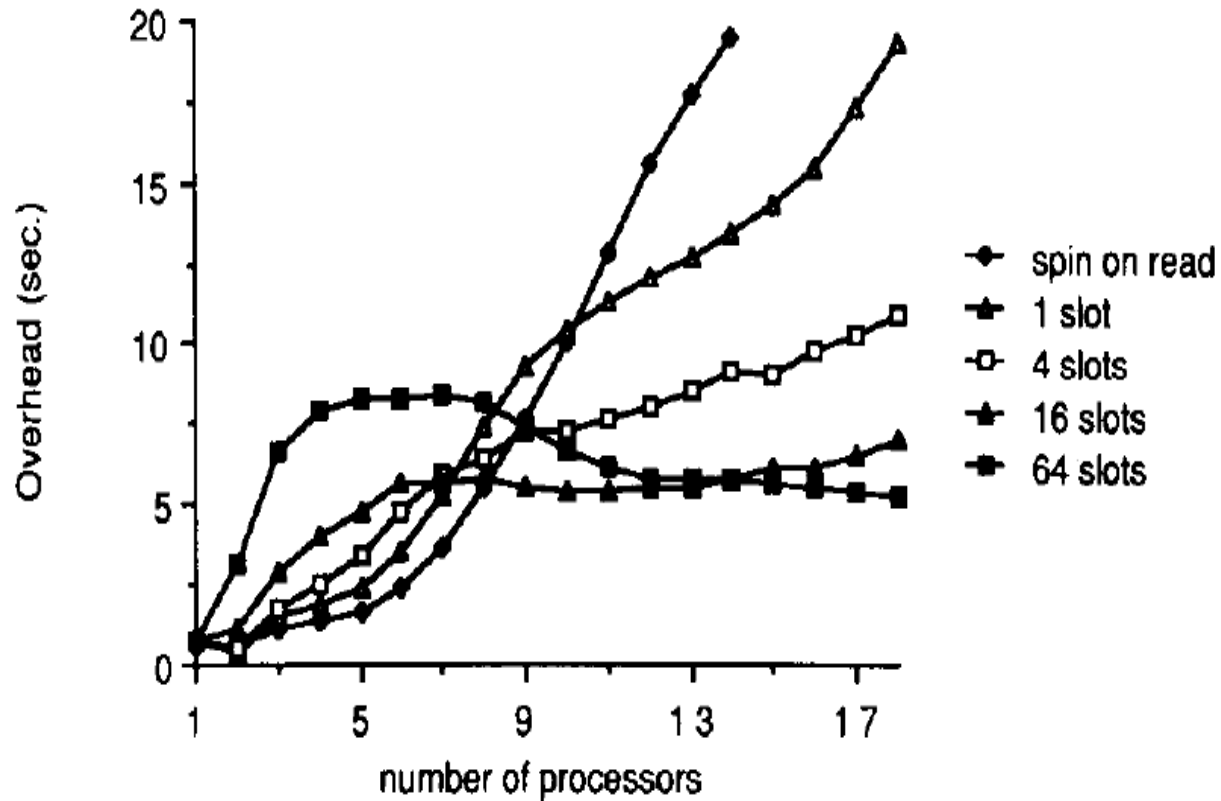


Fig. 4. Spin-waiting overhead (seconds) versus number of slots.