
The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors

T.E. Anderson, E.D. Lazowska, H.M. Levy

Presented by Davor Capalija

Topics in the Design and Implementation of
Operating Systems

Feb 8, 2007

February 8, 2007

CSC 2227

1

Overview

- Historical context
- Problem
- Approach
- Results
- Conclusion
- Influence

February 8, 2007

CSC 2227

2

Historical context: Threads for parallelism

- Threads - *Lightweight* processes
- Thread packages – common element of new languages and OSES for multiprocessors
 - Mach, Topaz, Psyche, DYNIX, UNIX
- Programs on multiprocessors use threads to exploit parallelism
 - DYNIX – process creation takes 25 ms

February 8, 2007

CSC 2227

3

Historical context: Problem

- DYNIX processes can be used only for coarse grained parallelism
- Topaz kernel provides faster thread mgmt
 - Bottleneck: routines protected by single lock
- Presto thread package (same authors)
 - application-level run-time library
 - kernel only for processor allocation and memory management
 - order of magnitude better than DYNIX processes
- The thread package views the process as a *virtual processor*
 - does not require modifications of the kernel

February 8, 2007

CSC 2227

4

Historical context: Solution user-level threads

- FastThreads - the outcome of this work
 - An order of magnitude better than Presto
 - Another comparison form subsequent paper
- Related Work on user-level threads
 - CThreads (Mach)
 - WorkCrews (Topaz)
 - Portable Common Runtime

Table I: Thread Operation Latencies (µsec.)

Operation	FastThreads	Topaz threads	Ultrix processes
Null Fork	34	948	11300
Signal-Wait	37	441	1840

February 8, 2007

CSC 2227

5

Approach – main idea

- *Small changes* in organization of data structures and corresponding locks
- Algorithm to queue for locks
- Thread's data structures
 - Program counter
 - Stack
 - Thread Control Block – state for thread mngmt
- Ready Queue
 - Keeps threads ready to run

February 8, 2007

CSC 2227

6

Approach – basic optimizations

- Thread creation
 - Creating parallelism for future
 - The stack need not be allocated until startup
 - Allocated conservatively large amount (no dynamic stack mngmt)
- Store deallocated TCBs and stacks in free lists
 - Avoids scanning/coalescing heap
 - Both allocation and deallocation are simple list operations

February 8, 2007

CSC 2227

7

Approach – thread management alternatives

- Effect of alternatives on
 - Creation, startup and finish
- Locking of shared data structures
 - Impacts latency and throughput
- 5 alternative mngmt strategies
- A and B – basic alternatives
- A – single lock
- B – multiple locks

February 8, 2007

CSC 2227

8

Approach – thread management alternatives

- C – Local Free Lists
 - Per-processor free lists without locks
 - Central locked ready queue
- Local free lists need to be balanced
- Global locked pool, local lists with maximum size of T
- Local free stack list, $T=1$
- Local free TCB list, $T=P$

February 8, 2007

CSC 2227

9

Approach – thread management alternatives

- E – Local ready queue
- Per-processor ready queues
- Lock-free local queues and global pool
 - Inefficient balancing
- Locked local queues, idle processors scan ready queues
 - Easy to avoid degradation to single queue case
- One-to-one correspondence between queues and processors for locality

February 8, 2007

CSC 2227

10

Results

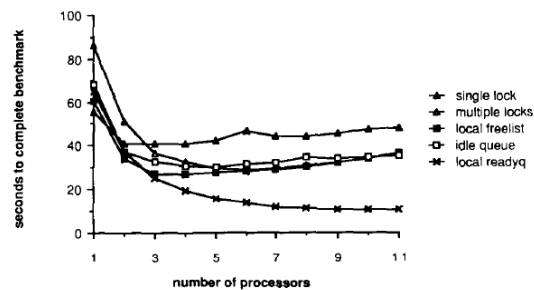


Fig. 1. Principal results for thread management—elapsed time to create, start, and finish 1 000 000 null threads (measured).

February 8, 2007

CSC 2227

11

Results

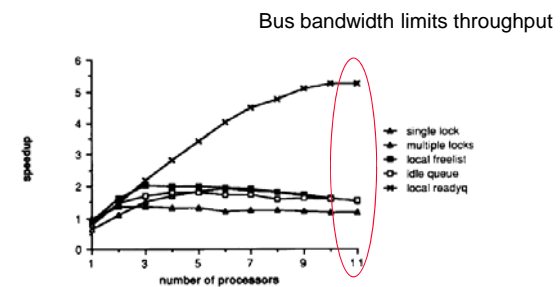


Fig. 2. Speedup to create, start, and finish 1 000 000 null threads (measured).

February 8, 2007

CSC 2227

12

Results

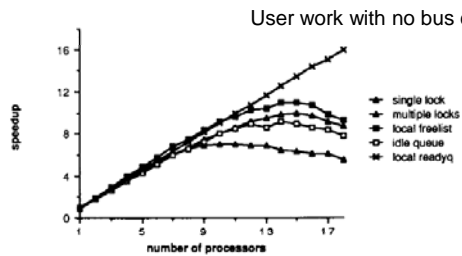


Fig. 3. Speedup, user work = 300 μ s (measured).

Fine-grained parallelism

Spin-lock management alternatives

- Busy lock
 - Spin wait
 - Relinquish the processor
- Overhead of context switch is prohibitive
- Spin-waiting has hidden cost
 - Bus contention

Hardware platform

- Per processor cache
- Write-through protocol
- Test-and-set (xchgb instruction)
- Xchgb is atomic, enforced by the bus
- Value in other caches is invalidated
 - Always, regardless if the value has changed

Spin-lock management alternatives

- A – Spin on Xchgb
 - Every instruction consumes bus resources
 - Uniformly degrades bus performance
- B – Spin on read
 - Coherent caches support spinning without using the bus
 - Cascade of repeated invalidation when lock released
 - Works with coarse-grained critical sections

Spin-lock management alternatives

- C – Ethernet-Style Backoff
 - An attempt to acquire a lock is costly
 - Try only when probability is high
 - Wait a time period dependent on the number of past failures
 - Possibility of indefinite starvation exists

Results

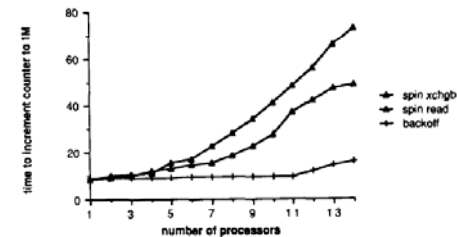


Fig. 6. Principal results for spin-waiting: elapsed time in seconds to increment a shared counter to 1 000 000 (measured).

Results

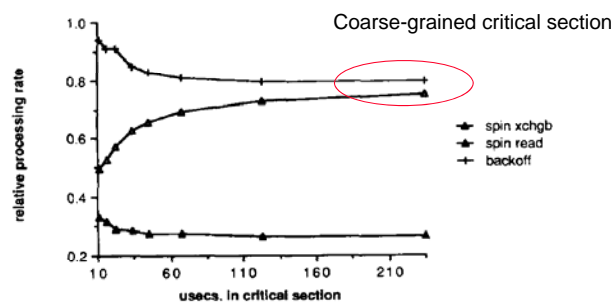


Fig. 7. Relative processor speed (8 processors to 1 processor) versus critical section size (measured).

Conclusions

- Proof that fast thread package can be implemented
- Per-processor data structures can be used to improve throughput
- The cost of spin waiting can be reduced by using an Ethernet-style back off algorithm
- Downsides, no evaluation using real-world benchmarks
- Problems with user-level threads
 - Only unprogrammed applications, no I/O
- What about the scalability of other OS data structures and services
 - Memory management, IPC, I/O

Influence

- Scheduling and resource management for multiprocessors
 - Guidance for the design of multiprocessor scheduler, processor allocation and control
 - Locality-based thread scheduling (child, parent-same CPU)
 - Work stealing
 - Affinity scheduling (cache locality)
- Kernel Support for Effective User-level thread management (same authors)
 - New kernel interface: Scheduler Activations
- Dynamic computation migration in multiprocessors
 - Instead migrating waiting threads, migrate active threads (or a portion of it)

Influence

- Pthreads
 - Pre-caching of TCBs and stacks into memory pool
- Dynamic task granularity control
 - Dynamic run-time information to select task size
- Multiprocessing implementation in many OSes
 - Exokernel OS (2000)
- Configurable locks
 - Large scale NUMA machines (Anderson et al. – UMA)