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

by Anderson, Lazowska, Levy, IEEE TC, 1989

- Data structures and algorithm altearnatives for thread management
- Fine grain parallelism, small differences can make big differences in performance
- What may degrade performance
  - data structures per processor
  - how to queue for locks (critical resources)
- New algorithms for thread management

What is a Thread?

- "lightweight" process
- executes a portion of a program, cooperates with other threads, shares the same address space
- maintains separate program counter, stack, and control block
- on multiprocessors, exploit parallelism

Thread Operations and Issues

- Even if you do not have an idle processor, you spawn a thread, (i.e., create parallelism for the future)
- Space-consuming: got to allocate space...
- Data structures: stack free list, ready queue, control block free list

- No Contention: minimize latency $\rightarrow$ cost of thread management
- With Contention: maximize TPUT $\rightarrow$ rate at whoch threads can be created, started, etc.
- Five management strategies

1) Simple Lock: central data structure protected by a single lock
   - straight forward
2) Multiple Locks: central data structures, each protected by a separate lock
   - better throughput, but latency more costly

3) Local Free List: per-processor free lists without locks; a central locked ready queue

- each processor maintains its own free list of control block and stack
- single shared ready queue
- better performance than algorithm 2)
- local free lists should be balanced (extra work, may trade space for time)

4) Idle Queue: a central queue of idle processors, per processor free lists

- takes advantage of parallelism
- cost of thread creation reduces when idle processors do the work in parallel with the creating processor
- central queue of idle processors (along with central queue of threads
- thread creation: dequeues an idle processor, does all initialization
- no benefit if all processors are busy

5) Local Ready Queue: per processor ready queue, per processor free lists

- one ready queue per processor
- danger for imbalance
  - solution: randomly choose a queue for each newly ready thread
  - increases the number of queues to be scanned when there are only few ready threads
  - this cost may outweigh the benefit...

Overall...

- Adding even a single lock aquisition can increase latency
- Per processor data structures are necessary to decrease latency and TPUT
- Thread management routines must be kept simple – enhancements that add complexity do not help
- When all data are kept under a single lock TPUT is limited by contention on the lock
- Only local ready queues can support higher TPUT

Spin-Lock Management Alternatives

- Spinning on a lock
- Hidden cost: executing processor can be slowed down because of bus contention
- implementation: availability of Xchgb operation
- atomically reads a memory location and updates its value
- 1: lock, 0: free

Approaches

- Spin on Xchgb
  - simplest
  - consumes bus
  - as more spin on a lock, the holder is slowed down
- Spin on Read
  - spinning in cache, avoiding bus traffic
  - not very efficient for short critical sections
- Ethernet-style Back-off
  - try to lock only when the odds are good
  - estimate from experience
  - if tried and failed many times, slow down
  - couls take longer to acquire a lock
  - but still effective...

Conclusions

- Simplicity is important for efficient implementation
- Small data structure changes: huge effect
- Per processor data structure can improve TPUT
- Spin-waiting slows down everybody
- Ethernet-style back-off