Are locks dead?

SCOOL Panel session
San Diego, 16 October 2005

- Victor Luchangco, Sun Labs
- Satnam Singh, Microsoft
- Robert Ennals, Intel
- Maged Michael, IBM

- Michael L. Scott, Rochester (moderator)

Parallelism for the Masses

- 2004 a watershed year: can't make faster superscalars and still cool them with air
- Multiprocessors about to become commodities
- Lots of recent focus on nonblocking synchronization and transactions
  - Will they replace locks?

Locks Have Problems

- Semantics: vulnerable to
  - » Thread failure, preemption, page faults
  - » Priority inversion
  - » Deadlock
- Convenience/performance tradeoff:
  - » Coarse-grain locks convenient but slow
  - » Fine-grain locks fast but very hard to use (correctness issues, deadlock)

Solutions?

- Ad-hoc nonblocking synch addresses semantic issues, but is at least as hard to use as fine-grain locks
- Transactions address convenience and usually performance
  - » Ousterhout'95 ("Why Threads Are a Bad Idea") lists 8 problems, at least 4 of which are arguably solved by transactions
- Note that nonblocking synch ≠ transactions

But . . .

- Memory-level transactions still have problems:
  - Poor common case SW performance (so far)
  - Immature HW support
  - Condition synchronization
  - Non-transactional data
  - Irreversible operations, compensation
  - Nesting
  - Legacy code (need to interoperate with locks?)

So . . .

- What does the future hold?
- Perhaps our panelists can tell us.
  (or you can tell them!)

  [5 min/panelist, followed by
  35 min of discussion]

Are locks dead?

A panel discussion at
the OOPSLA Workshop on
Synchronization and Concurrency in Object-Oriented Languages

Victor Luchangco
Sun Labs

Locks are not dead

Locks
vs.
Transactions

Locks

vs.

Transactions

False dichotomy

Why not locks?

Why not locks?

not composable

Why not locks?

not composable

“easy concurrency”

Why not locks?

not composable

deadlock, etc.

“easy concurrency”

Why not locks?

not composable

deadlock, etc.

“easy concurrency”

???

Obstruction-freedom

guarantee progress when running alone

Obstruction-freedom

easier to program

better implementations

Obstruction-freedom

obstruction-free transactional memory

Locks have their uses

Locks have their uses and their problems...

We can do better!

Need more work

conditional waiting

exceptions

input/output

Need more work

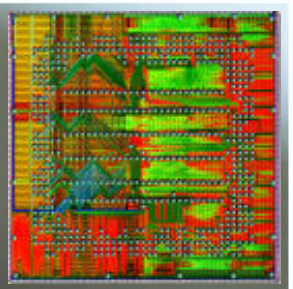
conditional waiting

exceptions

input/output

Let's do it!

Satnam Singh, Microsoft



- Locks are not dead
  - specific low-level uses
- Transactions
  - seem promising for application code
  - composability
  - good initial results from JSR-166 -> STM work
- Non-blocking code:
  - we've tried it: it's mind bending!
- Important distinction:
  - blocking / threads as a **programming abstraction**
  - non-blocking / **CPS as an implementation**

How I learned to stop worrying and love the lock

Rob Ennals

Intel Research, Cambridge

intel

Free of Locks
≠
Lock Free

Lock freedom – a very useful concept

But the name confuses the hell out of people

Atomic
≈
One Big Fat Lock

An easy way to explain transactions

And often the best way to implement them

Sometimes Blocking is OK

If transactions always short

If long transactions in separate process

If Mix and match blocking and non-blocking

Performance Matters

Transactions aid parallelism =>

parallelism often for performance.

Blocking Perf ≥ Non-Blocking Perf

"blocking" means we allow blocking,
not that we require blocking.

Respecting priorities can also reduce performance.

Would Hardware make Non-Blocking as Fast as Blocking?

It might, but then again, it might not.

Hardware speeds up small transactions

But if small, then blocking is ok

What do people actually need?

- Simplicity
- No deadlock
- Non blocking?
- Explicit rollback?
- Lock-Free?
- Respecting priorities? (not compatible with lock freedom)
- IO in transactions? (easy if allow blocking)
- Nested transactions?

Give the people what they want, but no more.

Conclusions

- Sometimes Non-Blocking is useful
  - But sometimes it isn't
- Sometimes Non-Blocking is as efficient as Blocking
  - But sometimes it isn't
- Atomicity does not require non-blocking
  - Don't carry the bathwater round with the baby

IBM Research

Panel: Are Locks Dead?
The Role of Blocking in Concurrency Control

Maged M. Michael

IBM Research

Are Locks Dead?

- No. Not yet. Probably never.
- Probably will be overtaken by transactions
- Architectural trends favor transactions
  - More concurrency, more efficient implementations of atomic and memory ordering instructions
- Problems with transactions remain
  - Efficient non-transactional access
  - I/O and irreversible side-effects
  - Composition, nesting

- Locks are not dead … But their role will be diminished

Blocking vs Non-Blocking?

- Non-blocking progress is of primary importance only in certain cases
- Should transactions guarantee non-blocking progress?
  - Not necessarily always … Need to figure out how to combine non-blocking and blocking transactional interfaces to the same objects efficiently
- Does the implementation of transactions have to be purely non-blocking?
  - No. Can use blocking mechanisms with efficient VM support.
- Non-blocking progress is preferable in general. But …
- There is place for blocking implementations in the VM and application levels

Conclusion

- Problems with transactions remain to be solved
- Eventually transactions will (hopefully) overtake locks as the most common synchronization mechanism
- There is always a place for blocking