Why do we want many cores on a chip?
Why is parallel programming hard?
What is transactional memory?

• Delimit regions of code accessing shared data
• Execute regions atomically and in isolation
• Buffer results and restart if isolation is violated
• “Serializable schedule of atomic transactions”
Paper “TxLinux”

• Semaphores & spinlocks
  – “busy waiting”
• Deadlock & livelock

• Primitives
  – xbegin, xend, xrestart
  – xpush, xpop
  – xcas, xtest, xgettxid
• Conflict: $\emptyset \neq \{W_a\} \cap \{R_b \cup W_b\}$
Paper “TxLinux”

• Asymmetric conflicts; complex conflicts
• Transactional vs. non-transactional thread
• Spinlocks vs. HTM
  – Contention low/high
• “Output commit problem” (I/O, low memory)
• Cooperative transactional spinlocks (cxspinlocks)
Paper “TxLinux”

- Critical sections use locks OR transactions
  - Most critical sections attempt transactions
  - Rollback and lock if a CS attempts I/O (or cost of restart would be high)
  - Locks optimize CS that always does I/O

- cx_optimistic: try to use transactions; if mutual exclusion needed, restart and obtain exclusive lock
- cx_exclusive: for paths requiring mutual exclusion (e.g., I/O)

Paper “DThreads”

- Multithreaded programming hard
- Enforce deterministic execution (but be efficient)
- Heisenbugs
- **Same program + same inputs = always same outputs**
- Goals of Dthreads: deterministic execution, easy to deploy, robust to changes in input/architectures/code, eliminates cache-line *false sharing*, efficient.
- How: turn multithreaded apps into multiple processes with private *copy-on-write* mappings to shared memory
Paper “DThreads”

- Pthread: race conditions (Figure 1)
- DThreads: deterministic output (Figure 2)
- Synchronization points
- Last-writer wins protocol
- Deterministic thread index
- Memory mapped files
- Global token (serialization, locks, condition variables, barriers,