CS4021/4521 Advanced Computer Architecture II

Prof Jeremy Jones

Rm 4.16 top floor South Leinster St (SLS)

jones@scss.tcd.ie
**Timetable Slots**

- Mon  @ 5  Salmon
- Thurs @ 4  LB120
- Fri  @ 3  LB08

use Fri @ 3 as a tutorial slot when needed
CONCURRENT PROGRAMMING WITH AND WITHOUT LOCKS

• mixture of theory and practice

• writing parallel programs (bucket sort, suffix array construction, binary search trees, ...)

• Peterson and Bakery locks [locks without atomic instructions]

• Spin model checker [revision?]

• atomic instructions

• serialising instructions

• caches coherency and the cost of sharing data between CPUs

• lock implementations and their performance [TAS, TATAS, ticket, MCS, ...]
CONCURRENT PROGRAMMING WITH AND WITHOUT LOCKS

• lockless data structures and algorithms
  ▪ CAS based
  ▪ LIFOs, FIFOs, linked, lists, trees, hash tables, ...
  ▪ memory management [eg. hazard pointers]

• hardware transactional memory [HTM]
  ▪ Herlihy and Moss [1993]
  ▪ Intel Transactional Synchronisation Extensions (TSX)
  ▪ hardware lock elision (HLE)
  ▪ restricted transactional memory (RTM)
USEFUL BOOKS

• *The Art of Multiprocessor Programming*
  Maurice Herlihy and Nir Shavit

• *The Spin Model Checker: Primer and Reference Manual*
  Gerald J. Holzmann

• *Principles of the Spin Model Checker*
  Mordechai Ben-Ari

  
  ▪ lecture notes
  ▪ coursework (3 or 4 exercises)
  ▪ miscellaneous materials (papers, documentation, sample code, ...)

CS4021/4521 © 2018 jones@scss.tcd.ie School of Computer Science and Statistics, Trinity College Dublin 11-Sep-18
ASSESSMENT [5 ECTS]

Coursework: 20%

- 3 or 4 coursework projects

Examination: 80%

- Dec 2018
- answer 3 out of 4 questions in 2 hours
MALBEC [malbec.scss.tcd.ie]

Supermicro 1U SuperServer 5018D-FNFT
Intel Xeon D-1540 2.0 GHz Broadwell CPU 45W
8 cores / 16 threads
128GB ECC RDIMM

Transactional Synchronization Extensions (TSX)
Haswell TSX implementation had a bug, Broadwell OK

Linux (Debian)

use for coursework
remote access via macneill or VPN

can use VS2017 “Linux development with C++” component to develop software remotely on malbec from a Windows PC
Why lockless algorithms?

- clock rate of a single CPU core currently limited to ≈ 4GHz
- single CPU core processing power NO longer doubling every 18 months
- Intel, AMD, Sun, IBM, ... producing multicore CPUs instead
- typical desktop has 4 cores with each core capable of executing 2 threads [hyper-threading] giving a total of 8 concurrent threads
- top-of-range desktop 2014 16 threads, 2016 32 threads, ... [Moore's Law and Joy's Law]
- need to be able to exploit *cheap* threads on multicore CPUs
- locked based solutions are simply not scalable as locks INHIBIT parallelism
- need to explore lockless data structures and algorithms
Consider a Binary Search Tree (BST) as an example

- `contains(key)`
  - returns 1 if key in tree

- `add(key)`
  - always adds to a leaf node

- `remove(key)`
  - 3 cases depending if node has zero, one or two children

- operations on tree normally protected by a per tree lock which inhibits parallelism

- why can't operations be performed in parallel?

- how much parallelism is possible?
BST Operations

- add (50) [single pointer updated]
- add(45) [single pointer updated]
- remove(45) – NO children [one pointer updated]
- remove(25) – ONE child [single pointer updated]
- remove(20) – TWO children

  find node (20)
  find smallest key in its right sub tree (22)
  overwrite key 20 with 22
  remove old node 22 (will have zero or one child)
  [key and a pointer updated]

- variations

  find largest key in left sub-tree instead of smallest key in right sub tree
  move node instead of value
Concurrent add operations

- concurrently add(27) and add(50)
  OK if adding to different nodes
- concurrently add(23) and add(24)
  problem as adding to same leaf node
  result depends on how steps of operations are interleaved [pointer updates]
  could work correctly, BUT...
  if there is a conflict ONLY one node may be added [23 or 24, BUT still a valid tree]
Concurrent remove operations

- concurrently remove(21) and remove(27)
  OK as both are leaf nodes [have NO children]
- concurrently remove(20) and remove(22)
  smallest key in 20's right sub tree is 22
  result depends on how steps of operations are interleaved [key and pointer updates]
  could work correctly, BUT ...
  one possible interleave is as follows
  
  both operations find 22
  20 is overwritten with 22
  old node 22 removed [by both operations], BUT
  22 still in tree!

other interleaves possible
Concurrent add and remove operations

- concurrently add(50) and remove(25)
- OK as modifying links in different nodes
Concurrent add and remove operations...

- concurrently add(50) and remove(40)
- result depends on how steps of operations are interleaved [key and pointer updates]
- could work correctly, BUT ...
- one possible interleave as follows
- 40 deleted, BUT ...
- 50 also deleted as attached to 40
Concurrent Operations on a BST

• concurrent operations ARE possible

• probability of a conflict inversely proportional to size of tree

• conflicts proportional to number of concurrent operations

• with a large tree, conflicts between operations will be rare

• with a large tree, should be able to achieve a linear speedup proportional to number of threads provided that conflicts can be detected and resolved

• protecting tree with a single lock is pessimistic as it assumes conflicts will occur resulting in NO parallelism

• a lockless algorithm is optimistic as it assumes conflicts unlikely to occur and, when they are detected, they are resolved – allows parallelism while there are no conflicts which hopefully is most of the time