CS4021/4521 Advanced Computer Architecture II

Dr Jeremy Jones

Office: O’Reilly Institute F.38

Email: jones@scss.tcd.ie {jones@tcd.ie}
## BACS / MCS Year 4 Timetable

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.00 – 11.00</td>
<td>MT: CS4061: Lect LB120</td>
<td>MT: CS4053: Lect LB08</td>
<td>MT: CS4051: Lect LB107</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.00 – 12.00</td>
<td>MT: CS4032: Lect LB04</td>
<td></td>
<td></td>
<td>MT: CS4001: Lect LB08</td>
<td></td>
</tr>
<tr>
<td>13.00 – 14.00</td>
<td></td>
<td>MT: CS4053: Lect LB04</td>
<td>MT: CS4053: Lect LB01 / ICT Lab 1</td>
<td>MT: CS4001: Lect Synge</td>
<td></td>
</tr>
<tr>
<td>14.00 – 15.00</td>
<td><strong>MT: CS4021: Lect LB08</strong></td>
<td><strong>MT: CS4051: Lect Joly</strong></td>
<td></td>
<td></td>
<td><strong>MT: CS4004: Lect LB107</strong></td>
</tr>
<tr>
<td>15.00 – 16.00</td>
<td>MT: CS4052: Lect M21</td>
<td></td>
<td>MT: CS4071: Lect LB01</td>
<td>MT: CS4032 Lect LB120 (2hrs)</td>
<td></td>
</tr>
<tr>
<td>16.00 – 17.00</td>
<td>MT: CS4051: Lect LB08</td>
<td>MT: CS4071: Lect LB01</td>
<td><strong>MT: CS4021: Lect LB04</strong></td>
<td>MT: CS4032 Lect/Lab: LB120/ICT2</td>
<td></td>
</tr>
<tr>
<td>17.00 – 18.00</td>
<td>MT: CS4061: Lect LB01</td>
<td>MT: CS4004: LB120</td>
<td>MT: CS4081: Lect LB01</td>
<td>MT: CS4004: Lect LB120</td>
<td></td>
</tr>
</tbody>
</table>

© 2015 jones@scss.tcd.ie School of Computer Science and Statistics, Trinity College Dublin 28-Sep-15
### MAI Year 5 Timetable

#### YEAR 5 MAI, 2015/16 - ELECTRONIC/COMPUTER ENGINEERING

<table>
<thead>
<tr>
<th>DAY</th>
<th>TYPE</th>
<th>0900 - 1000</th>
<th>1000 - 1100</th>
<th>1100 - 1200</th>
<th>1200 - 1300</th>
<th>1300 - 1400</th>
<th>1400 - 1500</th>
<th>1500 - 1600</th>
<th>1600 - 1700</th>
<th>1700 - 1800</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Second semester</td>
<td>CS7033 [LB1.07]</td>
<td>5C1 laboratory session [CADLAB]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Electronic/Computer Engineering Research Projects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Second semester</td>
<td>CS7046 [WS 1.3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Second semester</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRIDAY</td>
<td>First semester</td>
<td>5C3 [M20]</td>
<td></td>
<td></td>
<td>5C4 [PHH/CADLAB]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Second semester</td>
<td>CS7033 [LB1.07]</td>
<td>5C2 [PHH]</td>
<td></td>
<td>CS70033 [WS 1.5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2015 jones@scss.tcd.ie School of Computer Science and Statistics, Trinity College Dublin 28-Sep-15
CONCURRENT PROGRAMMING WITH AND WITHOUT LOCKS

• Peterson and Bakery locks [locks without atomic instructions]
• atomic instructions
• cache coherency [revision]
• memory ordering
• serialising instructions
• cost of sharing data

• Spin model checker [revision]
• lock implementations and performance [TAS, TATAS, MCS, ticket, ...]

• 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 Haswell CPU [2012] + Broadwell + Skylake
  ▪ hardware lock elision (HLE)
  ▪ restricted transactional memory (RTM)
• *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
  - tutorials and coursework
  - miscellaneous materials (eg. papers, documentation, sample code, ...)

© 2015 jones@scss.tcd.ie School of Computer Science and Statistics, Trinity College Dublin 28-Sep-15
ASSESSMENT [5 ECTS]

Coursework  20%

• 3 or 4 coursework projects

Examination  80%

• 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 has a bug

Linux (Debian)

For coursework
VPN access for remote access
Why lockless algorithms?

- clock rate of a single CPU core currently limited to $\approx 4\text{GHz}$
- 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]
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!

  many 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
- conflicts 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 if conflicts can be detected and resolved
- protecting tree with a lock is pessimistic as it assumes conflicts will occur - NO parallelism
- a lockless algorithm is optimistic as it assumes conflicts occur rarely and when detected they are resolved – allows parallelism while there are no conflicts [which hopefully is most of the time]