Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Problem Tutorial

- The *transpose* of a matrix **`M`** is a matrix **`T`**, where:
  `T[i,j] = M[j,i]` for all **`i`**, **`j`**.
- Consider a sequential program to transpose a matrix.
- How would you parallelise it?
- What problems/issues do you foresee?

Running Sequential Code

- Ran sequential code for a large matrix, with and without **`printf`** statements
- Source code: ***`seq-transpose-m2t.c`***
- Sizing: **`#define MSIZE 3000`**
- Compiling: **`cc -o seq  seq-transpose-m2t.c`**
- Running (and timing): **`time ./seq`**

Running Concurrent Code

- Developed code to use a thread per row.
- Ran concurrent code for a large matrix, with and without **`printf`** statements
- Source code: ***`row-transpose-m2t.c`***
- Sizing: **`#define MSIZE 3000`**
- Compiling: **`cc -o row  row-transpose-m2t.c`**
- Running (and timing): **`time ./row`**

Timing Comparison

- We have four runs, size=3000,
  sequential and row-concurrent, with and without **`printf`** output

prog.	printf?	real	user	sys	
`seq`	yes	24.169	9.459	6.687	
`seq`	no	0.125	0.106	0.018	
`row`	yes	52.813	28.246	201.822	4.35
`row`	no	0.166	0.154	0.283	2.63

4Ghz Intel i7, 32Gb, OS X Mojave

Printing takes a lot of time !

Why is sys time greater than real time for prog **`row`**?

Problem Tutorial (revisited)

- Assume that matrix **`M`** is square
- Consider a program that transposes this matrix ***in place***.
  - `M[i,j] = M[j,i]` for all `i`, `j`.
- How would you parallelise it now?
- What additional problems/issues do you foresee?

In-place update discussion

- No code was developed - just a general discussion of issues
- Looks fully independent at first glance, so how about a thread t(i,j) for each value of i and j each doing `M[i][j] = M[j][i];` ?
  - MSIZE*MSIZE threads! Too much thread create/memory/join overhead
  - if t(i,j) runs before t(j,i) the final outcome is M[i,j] = M[j,i] = initial value of M[j,i]
- Another solution: threads t(i,j) where i < j, that swap M[i,j] and M[j,i]
  - (MSIZE*(MSIZE-1))/2 threads!
- More like the **`row`** program: a thread t(i) that works for all j > i?
  - better, but some threads do 1 or 2 swaps, while others do almost MSIZE swaps
  - Suggestion: t(ia,ib) that does both t(ia) and t(ib) as above, but ia+ib = MSIZE-2