Parallel Computing
Why Parallel Computing?

- Most processors have multiple CPUs on chip
  - 4 – 16 or more today
- Operating system can distribute jobs
  - Different processors work on different tasks
- How to optimize a single program?
  - Distribute work among different processors
  - Different “threads” of control cooperate to do job
Parallel model used

- Multiple processors on chip (cores)
- All access the same memory space
  - Called shared memory
- Communication among processors via memory
  - Read/write to same memory location
Parallel Shared Memory

- Multiple processors work on a series of related jobs, numbered 0, 1, 2, ..., n
- How to coordinate assignment of work?
- How to balance load?
Static assignment

- Solution 1, divide jobs statically (k processors)
  - proc 0 does jobs 0, k, 2k, 3k, etc
  - proc 1 does jobs 1, k+1, 2k+1, 3k+1, etc
  - proc 3 does jobs 2, k+2, 2k+2, 3k+2, etc
Dynamic assignment

- Static assignment can be bad if the jobs have varied execution times – a few processors may carry most of the computational load

- Dynamic assignment
  - each processor starts with a job, same num as proc
  - job counter is a memory loc set to next job to be done
  - when a proc finishes a job, it gets the job count, increments it, and saves it back to memory, then executes that job (unless the count has reached the end of work)
Race condition

Same code on different processors
Assume X20 holds address of job counter,
X19 holds the job number of this processor

```
1a, 2a jobstart: ldur X19, [X20, #0]
1b, 2b add X9, X19, 1
1c, 2c stur X9, [X20, #0]
1d, 2d <start work>
...
1z, 2z b jobstart
```

Consider the execution sequence:
1a, 1b, 2a, 1c, 1d, 2b, 2c, 2d, .... 1z, ...

Both processors load the same value into X19 and therefore do the same job.
Mutual Exclusion

- Need a method to allow only one processor to execute a block of code (the load, increment, save sequence)
- Several primitive synchronization methods are used in different systems:
  - test and set
  - register exchange
  - fetch and increment
- These must be atomic operations – no intervening instruction can occur
ARMv8 solution

- two coordinated instructions
  - `ldxr rd,[rn, #0]` load exclusive register
  - `stxr rt, rm, [rn]` store exclusive register

  - Always paired, `ldxr` followed by `stxr` for the same memory location (same `rn`)
  - `stxr` fails if the memory location has been changed after the `ldxr` instruction and before the `stxr` instruction (by another processor or thread)
  - `stxr` success, set `rm` to 0
  - `stxr` failure, set `rm` to 1
Race Solution

Same code on different processors
(assume $s1 holds address of job counter)

1a, 2a jobstart:  ldxr X19, [x20, #0]
1b, 2b add X9, X19, 1
1c, 2c stxr X9, X10, [x20]
1cc, 2cc cbnz X10, jobstart
1d, 2d <start work>

1z, 2z j jobstart

Consider the execution sequence:
1a, 1b, 2a, 1c, 1cc – (memory value has not been changed so no branch) – 1d, 2b, 2c, 1e, 2cc – (between 2a ldxr and 2c stxr, instruction 1c stxr has changed the memory value, so branch) – 2a, 1f, 1g, 2b, 2c, 2cc – (memory value has not been changed this time, no branch) – 2d, ... 1z, etc.
The two processors get different job values.
The code sequence used to avoid the previous race condition is an example of a fetch-and-increment instruction implemented with ldxr and stxr. It is not an atomic assembly instruction but it has the same effect – a processor cannot move past this sequence unless it has executed in a way logically equivalent to its being atomic.

1a, 2a  jobstart: ldxr   X19, [x20, #0]
1b, 2b  add       X9, X19, 1
1c, 2c  stxr      X9, X10, [x20]
1cc, 2cc cbnz      X10, jobstart
Test-and-Set

How would you use ARMv8 ldxr and stxr instructions to implement the equivalent of an atomic test-and-set?

Test-and-set atomically checks whether a memory location is set to 1, rather than 0, and at the same time sets it to 1.
Another atomic control instruction is register-exchange. For this instruction the value in a given register is exchanged with the value at the designated memory location.

How would you implement the logical equivalent of an atomic register exchange instruction using ARMv8 ldxr and stxr?