* 1. Each unit of work depends on at most one other unit of work, which itself has no dependencies. 6 cores, 5 of them doing 2 units of work and the other doing 1. The available parallelism is 11/2 =5.5
  2. Each unit of work depends on the previous one, which in turn depends on all the previous ones. This is inherently serial.
  3. ?

Client A:

v = ?c2

c1!1

Client B:

v = ?c1

c2!2

* 1. ?
  2. If on the turn of the hour, the interrupt handler first resets the minute variable from 59 to 0, and then increments the hour variable by 1 (or indeed vice versa) but is pre-empted in between these two operations by a reader, the time read will be off by 59 minutes (too slow if the minute is set first, or too fast if the hour is set first).

lastHour := hour

lastMinute := minute

lastSecond := second

lastMillisecond := millisecond

currentTimeValid := false

update(hour)

update(minute)

update(second)

update(millisecond)

currentTimevalid := true

When the reader wants to read the time, they check the currentTimeValid flag, which, if true, indicates that they should read the time from the hour, minute, second, and millisecond variables. If the flag is false, then the reader should read from the lastHour, lastMinute, lastSecond, and lastMillisecond variables.

* 1. BEGIN TRANSACTION

update(hour)

update(minute)

update(second)

update(millisecond)

COMMIT

Whether or not this can be considered lock-free programming depends on how the transactional memory is implemented. If it is implemented by letting the transactor modify a copy of the data until the commit, at which point it is copied over, then this is indeed lock-free and equivalent to the code from part a. If instead the transactional memory blocks reads until any outstanding writes have been committed, then this is not truly lock-free but instead just uses a lower-level lock.

* 1. DO

v := READ(addr)

success := CAS(addr, v, v + 1)

WHILE success == false





This can be implemented with lock-free programming because since the data is immutable, there will be no conflicting writes (in fact no writes at all). All of the tree data will have to be copied anyway to create the new tree, and so each call to the method will be operating on its own copy.