Not that concurrent!

Øyvind Teig
www.teigfam.net/oyvind/home

@CPA 2015 fringe http://www.wotug.org/cpa2015/

)

Autronica Fire and Security AS, Trondheim, Norway. Autronica is a part of UTC Building & Industrial Systems, a unit of United Technologies Corporation

But less or more concurrent?

Based on the blog note «How much concurrency?" by Øyvind Teig.
See http://www.teigfam.net/oyvind/home/technology/093-how-much-concurrency/

Abstract (1/2)

1. Concurrency is, in the literature often used as an adjective: there is *more* or *less* concurrency, it is more or less *limited* - it may even be seen described as *complete.*
2. In trying to discuss multi-threaded programming with programmers who state that they program single-threaded it is important to communicate that they may program *less* concurrent, but probably *not* as non-concurrent as they believe.
3. What are the factors that increase concurrency and which factors are orthogonal to degree of concurrency?

Abstract (2/2)

4. Does a golang goroutine increase it and is a C++ object orthogonal?
5. Will the CSP paradigm generally enable increased concurrency?
   - Is the CSP paradigm of *communication by synchronisation* itself orthogonal to degree of concurrency?
6. It is also important to understand the term *parallel slackness*, does it introduce or enable more concurrency?
7. And what about atomicity?
8. This presentation will raise more questions that it is able to answer. However, some lines of reasoning are suggested.
9. Finally; is it at all meaningful to raise the awareness of *concurrent* as an adjective?

«More concurrency»

The first time I discovered about this “degreeness” was in the Clojure concurrency documentation:

***commute**. Must be called in a transaction. ... Thus fun should be commutative, or, failing that, you must accept last-one-in-wins behavior. commute allows for **more concurrency** than ref-set*

Metric of concurrency?

Then I asked about what the “metric of concurrency” was in a Clojure discussion group thread. Here’s from the answer by Leon Grapenthin:

How much happens at the same time in contrast to being queued up.
Remember that only successful transactions affect the outside world. E. g.: Of two successful transactions, one had to be restarted because of the other having completed during its first try => The two successful transactions didn’t happen in parallel. ***Using commute reduces the potential of a retry and thus allows more successful transactions in parallel => more concurrency***

«Limited concurrency»

From Kogan and Petrank's paper "Wait-free queues with multiple enqueuers and dequeuers":

***Abstract**: The queue data structure is fundamental and ubiquitous. Lock-free versions of the queue are well known. However, an important open question is whether practical wait-free queues exist. Until now, only versions with **limited concurrency** were proposed.*

«Complete concurrency»

In a paper by Micheal and Scott:

Lamport [.] presents a wait-free algorithm that restricts concurrency to a single enqueuer and a single dequeuer. ... Valois [..] presents a list-based non-blocking algorithm that avoids the contention caused by the snapshots of Prakash et al.'s algorithm and allows ***more concurrency*** *by keeping a dummy node at the head.. ...The algorithm employs separate Head and Tail locks, to allow* ***complete concurrency*** *between enqueues and dequeues.*

«Architectural» levels?

- Has degree of concurrency?
- Degree of concurrency is no issue?
- Degree of concurrency is void?

- Has degree of concurrency?
- Degree of concurrency is no issue?
- Degree of concurrency is void?

It's about semantics?!

parallel?

It's about semantics?!

concurrent?

Has degree of concurrency?

1. **A single-threaded** application where one work task blocks progression of other tasks has **less concurrency** than one the never blocks (also related to how long)
2. A **multi-threaded** application has **more concurrency** the more the number of thread relates to number of work tasks
3. An implementation of a **CSP** systems **does not need to do any busy-polling** to build the primitives. So it is **more concurrent** than one system that uses busy polling
4. Java has **notifyAll** which means that a Java thread may be notified just in case: there may not be anything important to be notified about. The system wouldn't know. So it is **less concurrent** than one system that hits all the time
5. **SDL** and some other message driven systems are not allowed to stop an event when the task is not really able to treat it, so it has to be stored with a save command. So it is **less concurrent** than systems where messages may be ltered in a dierent way
6. **occam** and **Go** may stop listening on channels to keep disturbances out. So they are **more concurrent** than systems that can't

Has degree of concurrency?

7. **Erlang** and **core.async** has ways to filter even on contents. So they are **more concurrent** than systems that can't
8. A **global critical region** to implement something is **less concurrent** than if only parts of the system was necessary to protect
9. **Not needing a critical region** is **more concurrent** than the opposite
10. A **non-blocking type atomic variable** where one is not **100% sure** that an unprotected change was correct at first trial is **less concurrent** than one where one would be certain
11. A system that has much **caching** will be **less concurrent** than one where the same problem is solved without caching (speed, only the degree to which one disturbs when it would be better not to disturb)
12. A system that has **stop-the-world garbage collectors** (as in Go < 1.5) is **less concurrent** than (..one that doesn't use that kind of garbage collection?)
13. A system that needs to **tune its garbage collection** (like Java) to get performance is **less concurrent** than one that (...) doesn't use dynamic memory
14. A system that **doesn't create garbage** is **more concurrent** than one that does

Has degree of concurrency?

15. A serial line interrupt that delivers **a character per interrupt** is **more concurrent** than one that treats frames and delivers a full message. (Less time per interrupt and receiver has time to pick up each byte vs. more time spent in interrupt because scheduling of receiver (driver, process, thread, task) is rather late, so needs a framed message)
16. Rewrite of some of the quotes above
    1. Clojure concurrent programming commute allows for more concurrency than ref-set.
    2. Using commute reduces the potential of a retry and thus allows more successful
    3. transactions in parallel => more concurrency
    4. Wait-free queues give more concurrency than lock-free queues
    5. A system without explicit locks at application level but requiring locks at lower level is
    6. more concurrent than one that has locks at application level (only) (Erlang quote)
    7. A non-blocking algorithm that avoids the contention caused by the snapshots of Prakash
    8. et al.'s algorithm allows more concurrency (Lamport, Valois)

Degree of concurrency is no issue?

1. **CSP with channels** that are non-buffered (synchronous) or buffered (asynchronous until full) (the process algebra concept)
2. **Formally verified** also with respect functional, but also temporal requirements (some blog here)
3. Systems with **deterministic guarantees** with regard to response times, like rate-monotonic scheduling (hard real-time systems)
4. **State transitions systems** (includes SDL and CSP)
5. Rewrite of a quote from above
   - The algorithm employs separate Head and Tail locks, to allow complete concurrency between enqueues and dequeues (Lamport, Valois) (Complete but with two locks!)

Degree of concurrency is void?

1. The property of being able to not fulll the specication
2. The property of being able to deadlock
3. The property of being able to livelock
4. The property of being able to not meeting a deadline

Is it about the semantics of parallels?

1. Synchronous parallel
2. Alphabetized parallel (in CSPm)
3. Interleaving / unsynchronized parallel (in CSPm)
4. Generalized parallel / interface parallel / sharing (in CSPm)
5. Linked parallel (in CSPm)
6. occam PAR

Is it about the semantics of parallels?

1. Synchronous parallel
2. Alphabetized parallel (in CSPm)
3. Interleaving / unsynchronized parallel (in CSPm)
4. Generalized parallel / interface parallel / sharing (in CSPm)
5. Linked parallel (in CSPm)
6. occam PAR
7. «Degree of concurrency?»