

Concurrent Object Oriented Languages

CSE 6490A

wiki.cse.yorku.ca/course/6490A

- **Name:** Franck van Breugel
- **Email:** franck@cse.yorku.ca
- **Office:** Computer Science and Engineering Building, room 3046
- **Office hours:** Mondays and Wednesday, 13:00–14:00

- concurrent algorithms
- concurrent programming in Java
- verification of concurrent Java code

- searching the literature
 guest lecture by librarian John Dupuis
- writing
 guest lecture by Susan Visser (Publishing Program
 Manager, IBM)
- presenting

Evaluation

- 3 assignments (20% each)
- 3 presentations (5% each)
- paper (15%)
- participation (10%)

Assignments

- 1 Find a nontrivial concurrent algorithm in the literature.
- 2 Implement your algorithm in Java.
- 3 Check some properties of your Java implementation using the model checker Java PathFinder.

For each assignment, you are expected to write a report.

Present your assignments.

- Combine your assignments into a paper
- of 15 pages
- using \LaTeX
- in Springer's Lecture Notes in Computer Science format.

For example,

- contributions to the wiki
- contributions to the forum
- discussion in class
- questions after presentations
- be on time
- ...

In which systems/applications do you find concurrency today?

- operating systems
- data bases
- graphical user interfaces
- Internet applications

In which systems/applications will you find concurrency in the near future?

- Everywhere?

“Concurrency has long been touted as the “next big thing” and “the way of the future,” but for the past 30 years, mainstream software development has been able to ignore it. Our parallel future has finally arrived: new machines will be parallel machines, and this will require major changes in the way we develop software.”

Herb Sutter and James Larus. Software and the Concurrency Revolution. *Queue*, 3(7):54-62, September 2005.

Moore's Law

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years."

Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.



source: www.rentoid.com

Carver Mead dubbed this Moore's Law.



Moore's Law

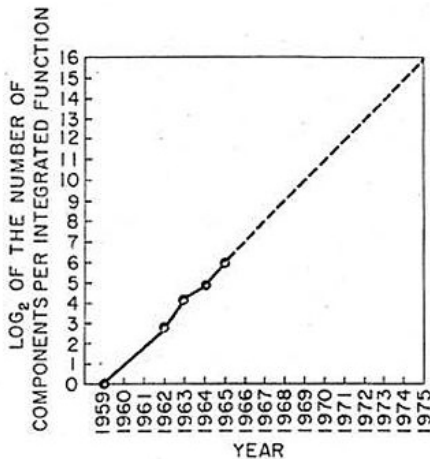


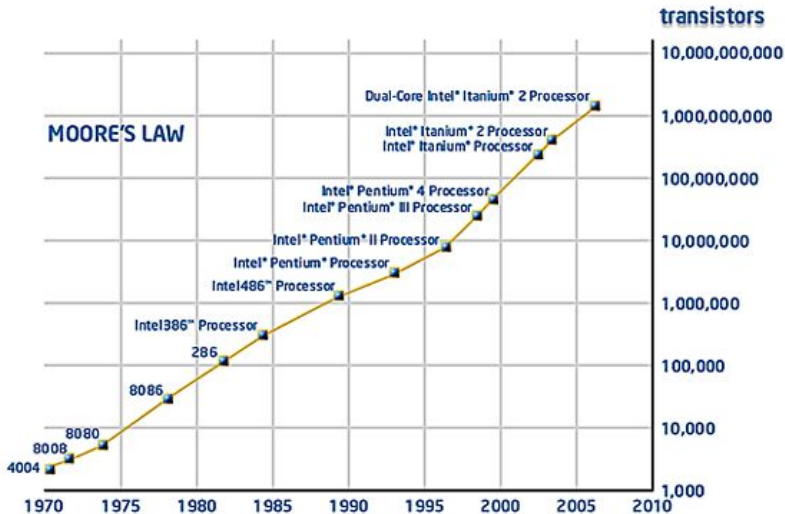
Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

source: www.intel.com

“So the original one was doubling every year in complexity now in 1975, I had to go back and revisit this... So then I changed it to looking forward, we'd only be doubling every couple of years...”

Excerpts from a conversation with Gordon Moore (source: www.intel.com)

Moore's Law



source: www.intel.com

This complexity (which doubled every two years) is strongly correlated with

- processor speed
- memory capacity

The End of Moore's Law?

“The combination of limited instruction parallelism suitable for superscalar issue, practical limits to pipelining, and a “power ceiling” limited by practical cooling limitations has limited future speed increases within conventional processor cores to the basic Moore’s law improvement rate of the underlying transistors.”

Kunle Olukotun and Lance Hammond. The Future of Microprocessors. *Queue*, 3(7):26–29, September 2005.

“Chip multicore processors implement two or more conventional superscalar processors together on a single die.”

Kunle Olukotun and Lance Hammond. The Future of Microprocessors. *Queue*, 3(7):26–29, September 2005.

The End of Moore's Law?

“While Moore’s Law continues to hold, due to both intractable physical limitations and practical engineering considerations, that increasing density is no longer being spent on boosting clock rate, but rather on putting multiple CPU cores on a single CPU die. ... most code can (and should) achieve concurrency without explicit parallelism ...”

Bryan Cantrill and Jeff Bonwick. Real-world concurrency. *Communications of the ACM*, 51(11):34-39, November 2008.

Concurrency is Hard

“But concurrency is hard. Not only are today’s languages and tools inadequate to transform applications into parallel programs, but also it is difficult to find parallelism in mainstream applications, and—worst of all—concurrency requires programmers to think in a way humans find difficult.”

Herb Sutter and James Larus. Software and the Concurrency Revolution. *Queue*, 3(7):54-62, September 2005.

Concurrency is Hard

“Multicore architectures will (finally) bring parallel computing into the mainstream. To effectively exploit them, legions of programmers must emphasize concurrency.”

“Nontrivial software written with threads, semaphores, and mutexes are incomprehensible to humans and cannot and should not be trusted!”

Edward A. Lee. Making Concurrency Mainstream. Presentation at the *17th International Conference on Concurrency Theory*, Bonn, Germany, August 27, 2006.

Amdahl's Law

“... the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.”

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Joint Computer Conferences*, pages 483–485, Atlantic City, NJ, USA, April 1967. ACM.



source: Perry Kivolowitz

