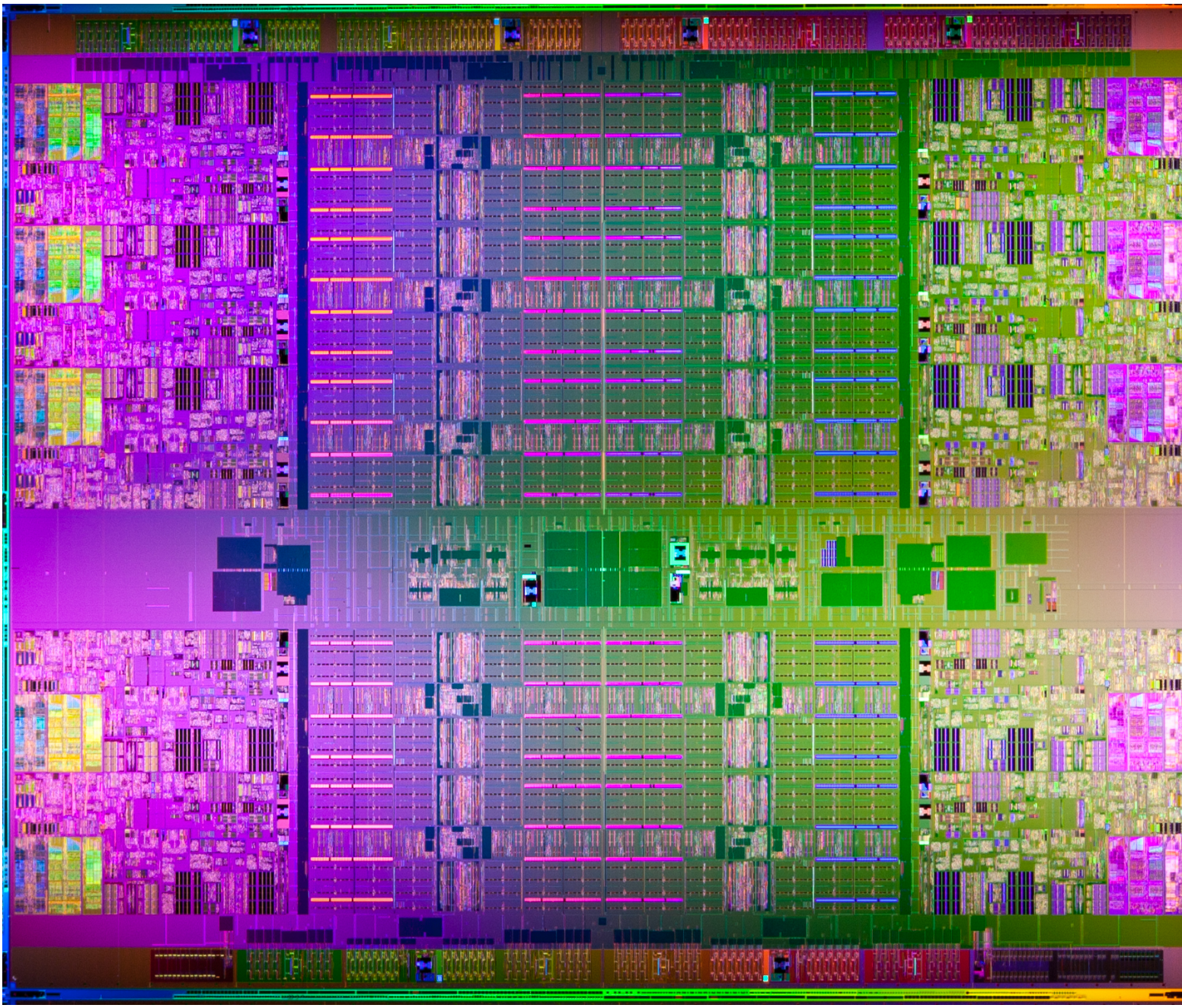


60007

Theory and Practice of Concurrent Programming

Imperial College London

Contents

Chapter 1

Introduction

1.1 Course Structure & Logistics

1.1.1 Structure

Dr Azalea Raad

Prof Alastair Donaldson

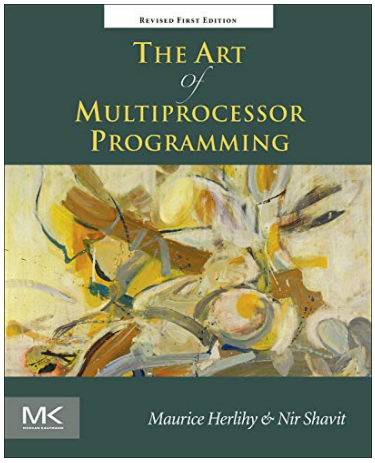
Theory For weeks 2 → 5:

- Intro to synchronisation paradigms (mutual exclusion, readers-writers, producer-consumer)
- Low-level concurrent semantics (sequential consistency, Intel-x86)
- High-level concurrent semantics (concurrent objects, linearisability)
- Transactional memory (serialisability)

Practical For weeks 5 → 8:

- Threads and locks in C++
- Implementing locks
- Concurrency in Haskell
- Race-free concurrency in Rust
- Dynamic data-race detection

1.1.2 Extra Materials



The Art of Multiprocessor Programming
About 65% of the theory course.

1.2 Preface for Concurrency

1.2.1 Moore's Law

Moore's Law — **Definition 1.2.1**

An empirical (supported by observation) law that states the density of transistors in an integrated circuit will double approximately every two years.

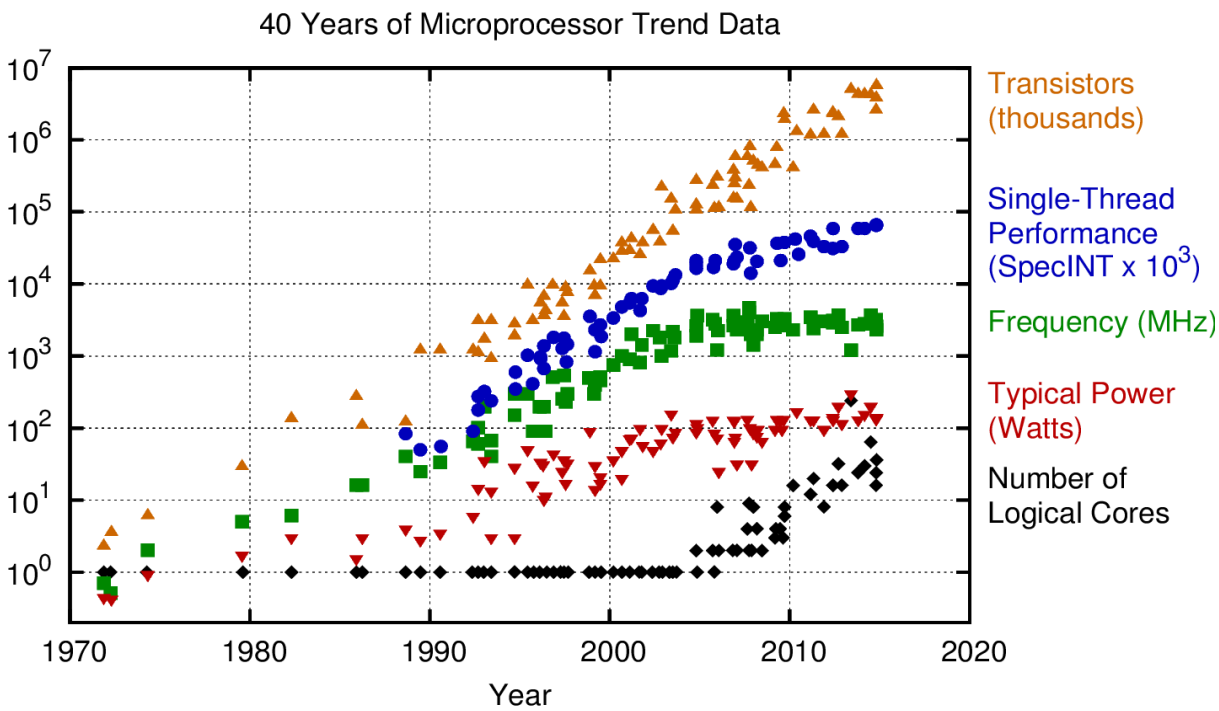
- The observation is named after Gordon Moore (co-founder and later CEO of Intel).
- This law no longer holds, and sequential performance improvements have declined.

Dennard/MOSFET Scaling — **Definition 1.2.2**

$$\text{Power} \propto \text{Transistor Size}$$

A scaling law stating that as transistor density increases, the power requirements stay constant.

- Increasing transistor density results in power staying constant (less power per transistor) and lower circuit delay.
- This allows for higher switching frequency ⇒ higher clocks frequencies ⇒ better sequential performance).



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

The performance improvements typically expected yearly (moore's law and Dennard scaling) no longer apply.

- Sequential (Single-Thread) performance improvements have declined.
- Parallelism is being exploited to improve performance (uniprocessors are virtually extinct).
- Shared-memory multiprocessor systems have lost out to multicore processors.

Amdahl's Law — **Definition 1.2.3**

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \text{ where } p = \text{parallel portion and } s = \text{threads}$$

Amdahl's law describes the speedup of a program, associated with the number of threads.

- Can be applied to other resources.
- Versions of the equation exist for different proportions using different numbers of threads.
- As the number of threads increases the sequential part of the program becomes a bottleneck.

1.2.2 Concurrency Difficulties

Writing correct, concurrent code is difficult.

race Condition — **Definition 1.2.4**

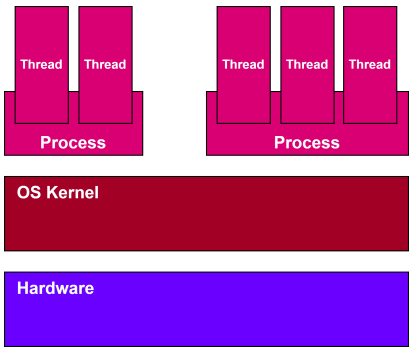
A potential for a situation where the result of a program depends on the non-deterministic timing or interleaving of threads.

- Where multiple threads access data (non-atomically) and at least one writes.
- Where a lack of enforced ordering on some events causes differing results (e.g output to user)

- A process can have multiple threads executing in parallel.
- Cannot determine at compile time the relative speed of execution of threads (many delays are unpredictable; cache misses, page faults, interrupts).
- Cannot predict how long threads will be blocked (e.g I/O) or when threads will be scheduled (or use up their time quantum).

Hence we must use synchronisation mechanisms to regulate accesses to shared data that can result in a race condition.

1.2.3 OS Concepts



- Operating system provides process and thread abstractions.
- A process contains one or more threads (streams of instructions being executed).
- A process has its own address space, all threads in the process share this address space.
- The OS kernel contains a scheduler which schedules processes & their threads.

Chapter 2

Concurrency In C++

2.1 Locks and Threads