Seminar 236832

Advanced Topics in Concurrent Programming

Erez Petrank

Agenda

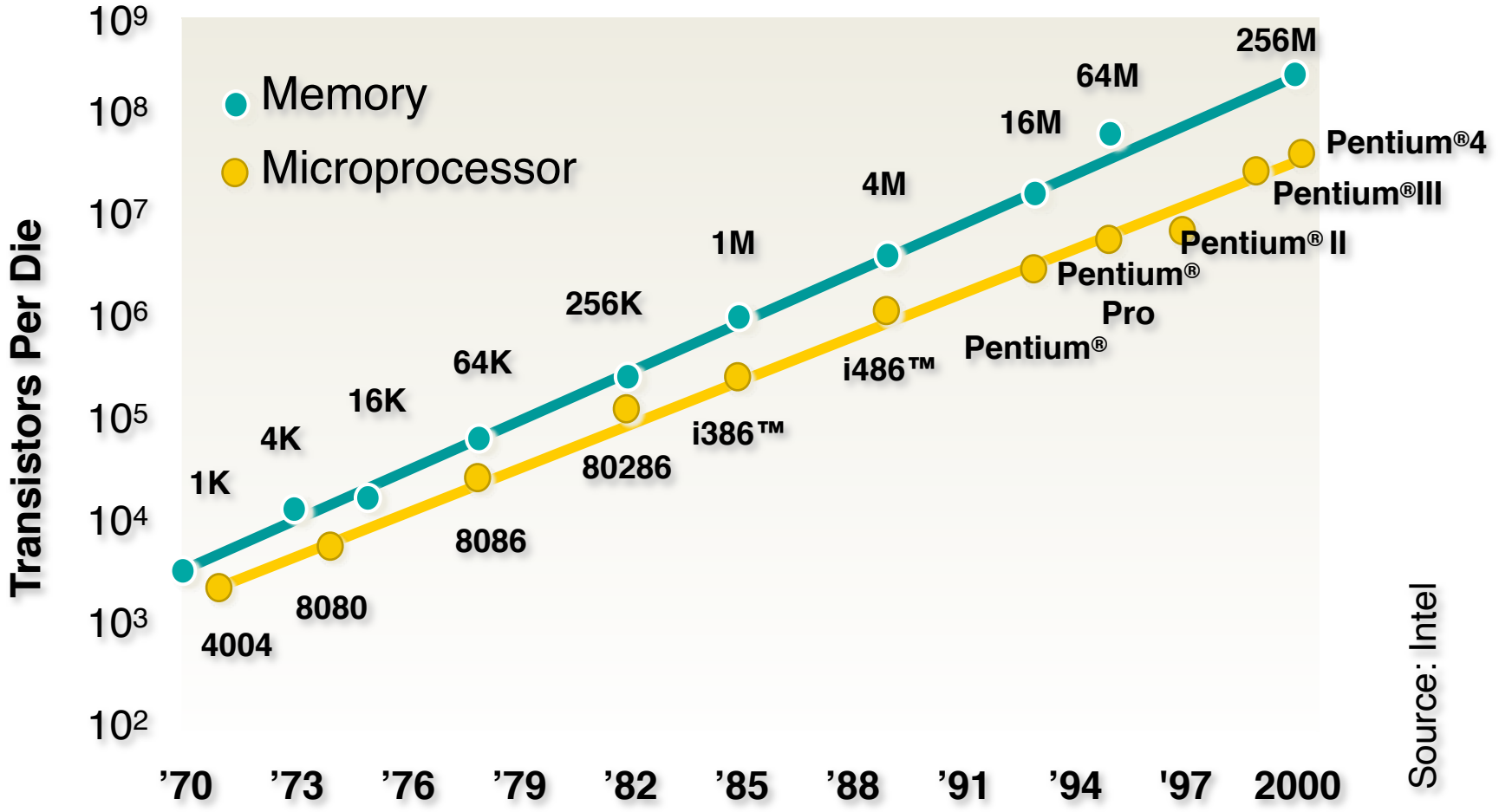
- The (practical) problem of parallelism today.
- Basic terms.
- Administration.

Moore's Law

- The number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years. [Gordon E. Moore, 1965]

- Also applies to: processing speed, memory capacity, and even the resolution of digital cameras.
- Predicted for "at least 10 years", lasted at least 50…

Moore's Law

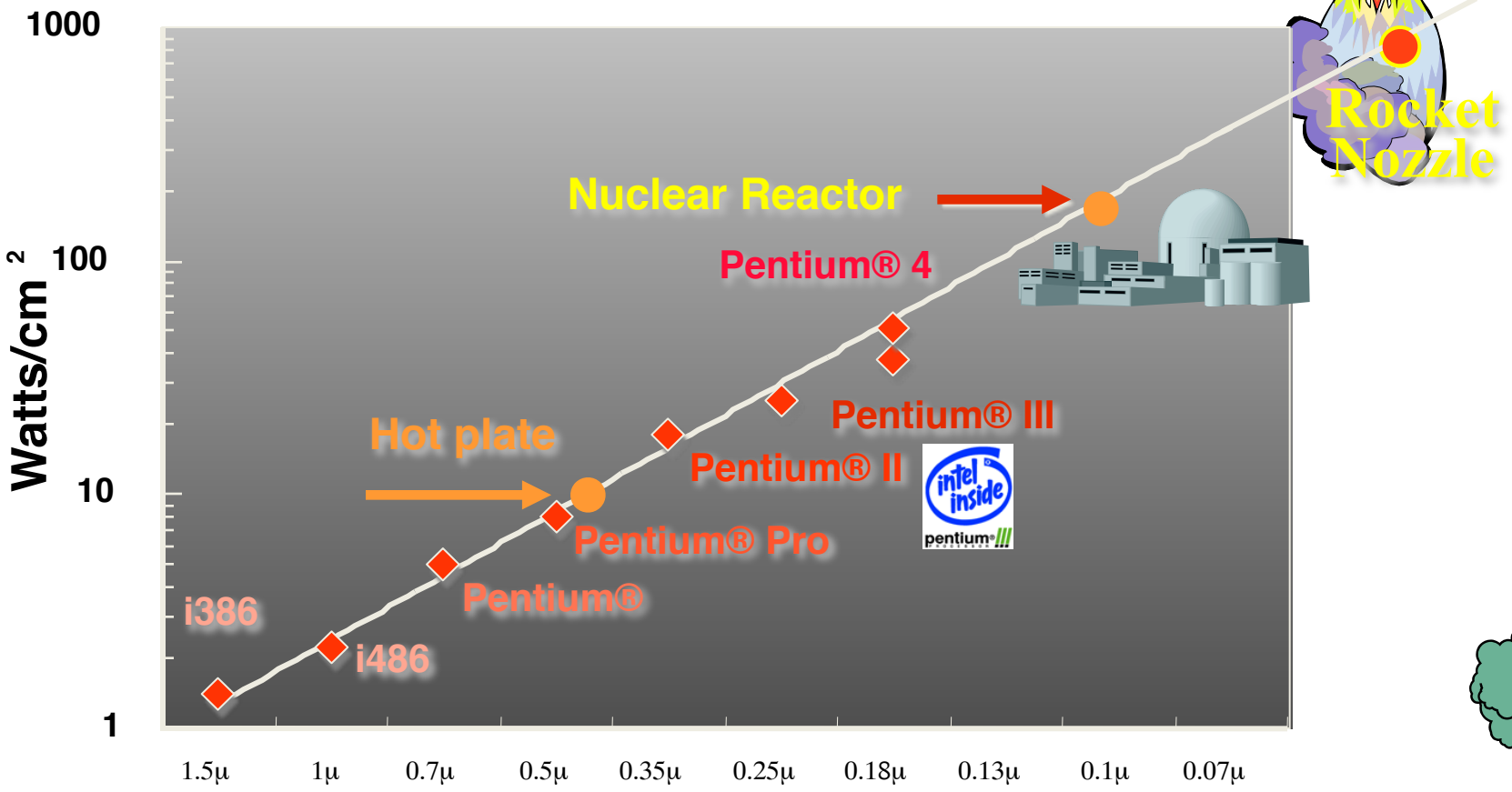

Transistors Per Die
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
Memory
Microprocessor
1K
4K
16K
64K
256K
1M
4M
16M
64M
256M
4004
8080
8086
80286
i386™
i486™
Pentium®
Pentium® Pro
Pentium® II
Pentium®III
Pentium®4
'70
'73
'76
'79
'82
'85
'88
'91
'94
'97
2000
Source: Intel

Transistors and Performance

- Make transistors smaller, and then
- they get closer to each other, and then
- the delay in spreading their signal becomes smaller, and then
- we can increase the operating frequency. Hurrey!

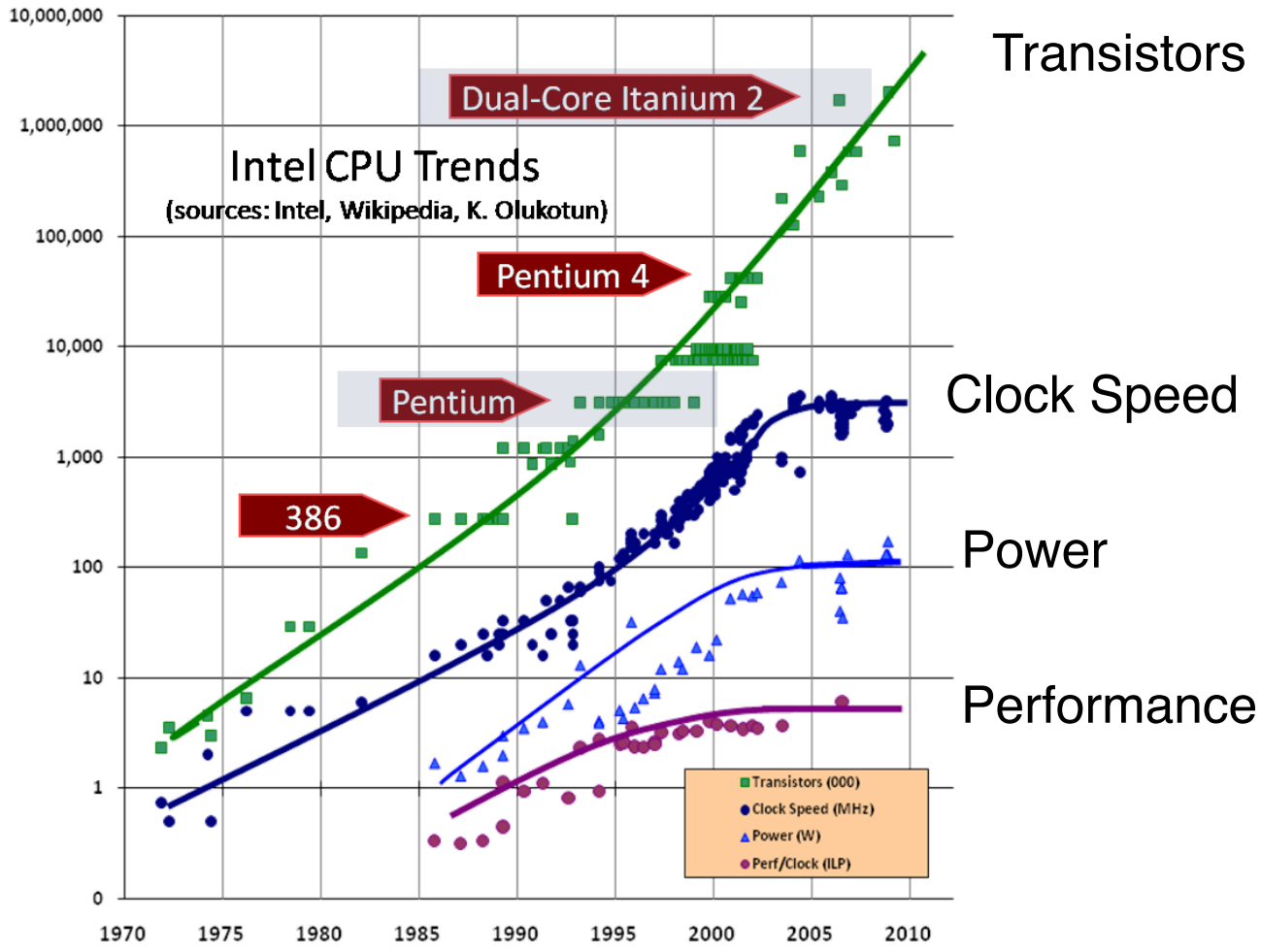
But then, power increases!

Power density



* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999.

Processor Development

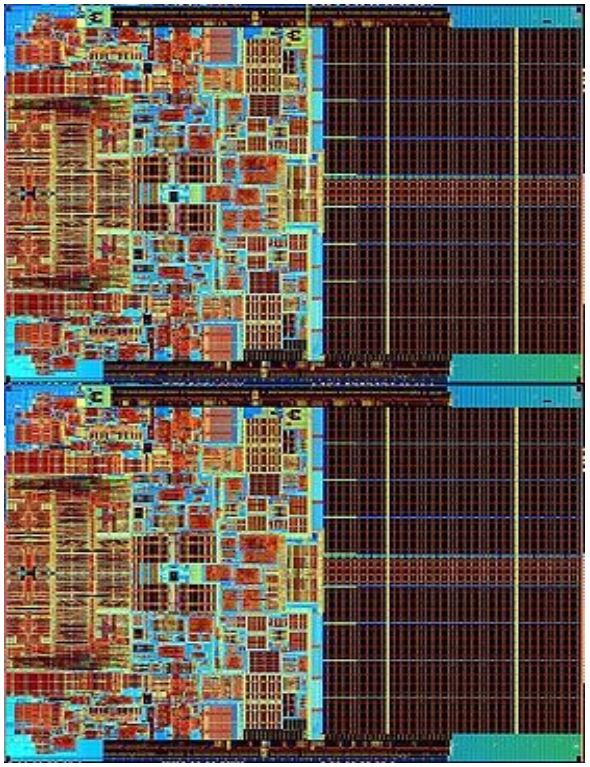


For Software: Free Lunch is Over

- Want faster software? Buy a new computer...
- So software producers did not have to work hard for better performance.

Hardware Change

- Do not increase frequency.
- But there is more space!
- Put several “cores” on the same chip.
- Let the software guys parallelize their programs.



Current Status

- Multicores everywhere.
- But software is not ready!
  - Lots of existing single-threaded code.
  - Difficult to write parallel code.
  - Difficult to debug and ensure reliability.

- This is a major challenge for the computing world today.

Solution: Automatic Paralellization?

- Automatic Parallelization (compiler or source to source).
  - Lots of research and theory, initially mostly for Fortran (no pointers, no recursion)
  - But little success for general code
  - Some success for scientific computing (FFT, Matrix multiplication, etc.).
- Problem: existing dependencies in “typical” code foils parallelism.
- Seeming conclusion: developers must specify parallelism or at least give hints about it.

Solution: Design a New Language?

- Design a new language with an intuitive and robust expressiveness for parallelism.
  - How does one specify parallel execution?
  - How does synchronization go?

- Current programming languages typically use threads. For many programmers threads create synchronization problems, and the shared memory implies data races.
- Difficult to develop and debug.

Parallel Programming Environments in the 90's

ABCPL
ACE
ACT++
Active messages
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*.
"C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL

CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
ECO
Eiffel
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda

GLU
GUARD
HAsL.
Haskell
HPC++
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold

Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
p4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm

Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
pC++
PCN
PCP:
PH
PEACE
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick Threads
Sage++
SCANDAL
SAM

pC++
SCHEDULE
SciTL
POET
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

Taken from: "Patterns for Parallel Programming" T. Mattson and K. Kuetzer, 2004

Design a New Language?

- Some big designs are still in the work:
  - Transactional memories ?
  - ~~SUN/ORACLE's Fortress?~~
  - IBM's X10?

- An open problem.

Help Experts Can Provide

- Build libraries:
- Experts can work hard and design a linked-list with “excellent properties”.
- Non-experts can just use that list in their programs to obtain a good behavior.
- Properties:
  - Efficiency (time, space, scalability, etc.)
  - Progress guarantees

- We will go over algorithm designs.

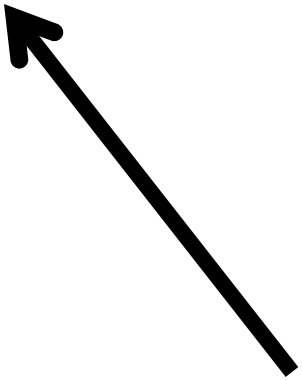
Efficiency

- Parallelism brings a new dimension to efficiency: synchronization overheads and delays.
- What does it mean for a parallel program to be “good”?
- Various theories about lock-freedom, fairness, progress guarantee, etc.
- We will explore some issues in that end, focusing on lock-freedom.

Another Major Problem

- Debugging and Reliability

How difficult is it to find (and fix) a visible bug?



When benchmarks are running well on the developer platform, how many bugs are still hiding?

Debugging Parallel Programs

- Some bugs appear in a specific schedule and become visible only once every *n* runs.
  - How large is *n* ?
- Sometimes bugs produce different behaviors depending on the schedule.
- Can we ever be sure that a program is bug-free?

Debugging Tools

- Replay (same schedule)
- Randomly choose schedules
- Go over all “reasonable” schedules (and check what?)
- Race detection
- Determinism
- Etc.

Verification

- Verify that the program satisfies some properties.
- Very difficult (undecidable in the general case).
- Should we prove each code written?
  Can we verify a set of programs?
- Tools today are not ready for general developer use.
- Various logics: trade-offs between expressiveness and complexity.
- An important first step: type-safety.
- Verify high or low level code?
  - Verify the compiler?
  - Proof carrying code,
  - Typed assembly

Parallelizing Software: a Major Focus of Industry and Academia

- Large companies are trying various directions, (e.g., Intel, Oracle, IBM, Microsoft, Google, VMWare). We will discuss several directions in this seminar.
- Major conferences devote sessions for attempts to deal with the parallelism challenge.
  (See the resurrection of transactional memories.)

Major Focus of Industry and Academia

- Some money is being allocated…
- Example 1: DARPA (Pentagon's Defense Advanced Research Projects Agency) invested a large fund (~500M) for the next generation parallel platform.
  Surviving contractors were IBM and Cray.
- Example 2: "Microsoft and Intel Launch Parallel Computing Research Centers to Accelerate Benefits to Consumers, Businesses" (see http://www.microsoft.com/presspass/press/2008/mar08/03-18UPCRCPR.mspx)
- Microsoft and Intel have committed a combined $20 million to the Berkeley and UIUC research centers over the next five years. An additional $8 million will come from UIUC, and UC Berkeley has applied for $7 million in funds from a state-supported program to match industry grants.

Motivation Summary

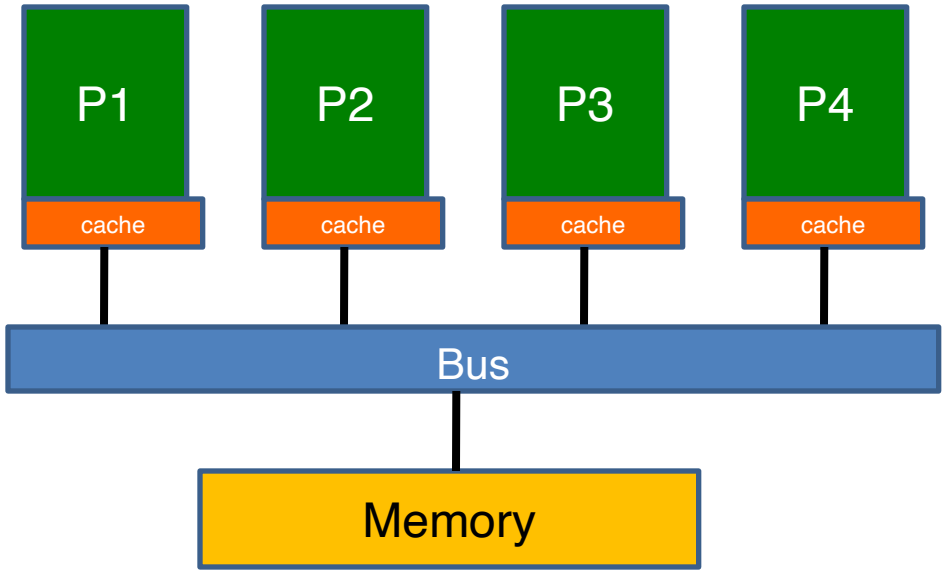
- Parallel platfroms are widely available.
- We do not know how to efficiently develop, debug and verify parallel software.
- A major challenge for the computing world today.
- We will study solution directions,
  - and technical issues in the design of scalable parallel algorithms.

Some Basics

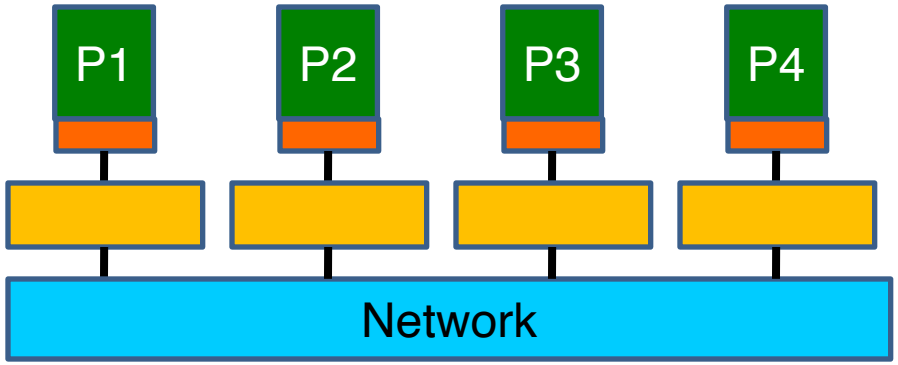
Platforms, Processes, Threads

Hardware

- Multi-processor (or multi-core),
- NUMA,
- Distributed system,
- and combinations.



Multiprocessor
(Multicore)



Distributed
Architecture

Hardware

- Main performance and scalability issue:
  - communication via bus or network.
  - memory access
- Main programming implication: communication and synchronization via message passing, or shared memory.

Basics: Software

- A process: a snapshot description of a program execution (excluding file contents, etc.).
- Typically: program counter, memory content (including local registers and global memory), and a process id.
- Unix: create via execution of a fork; synchronization via join, information transfer via sharing of parts of the memory, or sockets.
- Windows: createProcess (…).

Processes are Costly

- A new process (Unix) receives a copy of parents properties.
- Opening: naively, all memory copied.
  - Practically, copy-on-write.
- Switching: TLB invalidated.

- Making it lighter and faster: share memory… Threads!

Threads

- A process may contain several threads, sharing memory space.
- A thread consists of a thread id, program counter, registers, and a stack.
- Java: create a sub-class of Thread and override the “run()” method with the code.
- Communication using the shared memory.
- Synchronization via locks, monitors, and special instructions (compare-and-swap, etc.).

Comparison

- Processes are easily migrated (to execute remotely).
- Processes are isolated and thus less bug-prone.
- Threads are lighter to create and switch.

Summary: Basic Terms

- Computer platforms typically multicores or multiprocessors.
- They may connect to create a distributed system.
- Software for parallel computation typically spawns processes or creates threads.

List of Topics

- Introduction
- Concurrent Algorithms
- Debugging
- Concurrent execution on GPUs
- Concurrent execution on non-volatile memory
- Server architecture
- Languages and Runtimes for Concurrent and Distributed Programming
- Systems Scalability

Administration

- You picked 19 papers.
  - This means you influenced the material.
  - This means everybody had a choice.

- Next week Topics 2 and 3 (introduction)
  - Adi Simhi & Nadav Elias
- Passover (no class) on the second week.
- I will talk on the third week
- Topics 4 and 6 will be taught on the fourth week
- I will update the page with the subsequent schedule
- Web page for announcements.

Presentation

- Each student will pass a lecture about one of the topics.
- A lecture should take 50 minutes, two lectures each week.
- PowerPoint or Keynote or PDF in English.
  - Note that the interface is HDMI. (Check your connection ahead of time.)
  - This year: Zoom, practice sharing screen ahead of time…
- Lectures should be sent to me (erez@cs) in the evening before the seminar (by 19:00 Wednesday) or earlier.
- After the talk, the lecturer will modify the slides to create a final version and send it to me the next day (Thursday).
- The lectures will be posted on the course web-site.
- Any use of available material should be clearly stated (on each slide).

Grading

- Lecture (at least 85%),
- participation (at most 15%),
- Possible alternative mode.
- You must attend…
  - Reduction of 5 points for each (unjustified) absence.
  - A justified absence implies a summary.
- The talk will be graded by:
  - Knowledge of the material.
  - Slides effectiveness.
  - Communication of the ideas.
  - Also, keeping the lecture time limits.
  - Also, following the procedure.

Corona Participation

- Possible alternative mode.
- By the end of each talk there will be 3 basic questions that everybody will have to answer to show that he understood the basics.

Corona Participation

- Possible alternative mode.
- By the end of each talk there will be 3 basic questions that everybody will have to answer to show that he understood the basics.

- **Example questions for this talk**:
  1. Moore Law:
     A. predicts exponential growth in transistors/circuit
     B. predicts that power will become a problem
     C. Became relevant recently
     D. Predicts concurrency

Corona Participation

- Possible alternative mode.
- By the end of each talk there will be 3 basic questions that everybody will have to answer to show that he understood the basics.

- **Example questions for this talk**:
  2. What is the current status?
     A. Concurrency is well understood
     B. Compiler automatic parallelization is good at parallelizing general programs
     C. Almost all machines are multicores
     D. There is no way experts can help writing concurrent code

Corona Participation

- Possible alternative mode.
- By the end of each talk there will be 3 basic questions that everybody will have to answer to show that he understood the basics.

- **Example questions for this talk**:

3. Why do we need threads in addition to processes?
    A. Threads are easier to program
    B. Threads are lighter to create and switch
    C. Threads can extend virtual memory
    D. Threads can access kernel code

Talks

- Main goal: make people understand!
- Original presentations of most topics can be found on the web. You can use these slides as the basis for your presentations.
  - But the original presentations are for a different timing length and a different audience.
  - Provide some background and expand on interesting aspects.
- Practice giving the talk (on your friends / fellow students).

Registration

- Registration is manual
  - I will register undergraduates.
  - Graduate students register on their own.
- Assignments appear on the web. Let me know if there is a problem.
- Check for info and updates on: http://www.cs.technion.ac.il/~erez/courses/seminar

To Conclude…

- Lots of interesting talks on parallel and distributed algorithms awaiting!
- Send me your talks the evening before!
- See you after Passover.