Seminar 236832

Advanced Topics in Concurrent Programming

Erez Petrank

Agenda

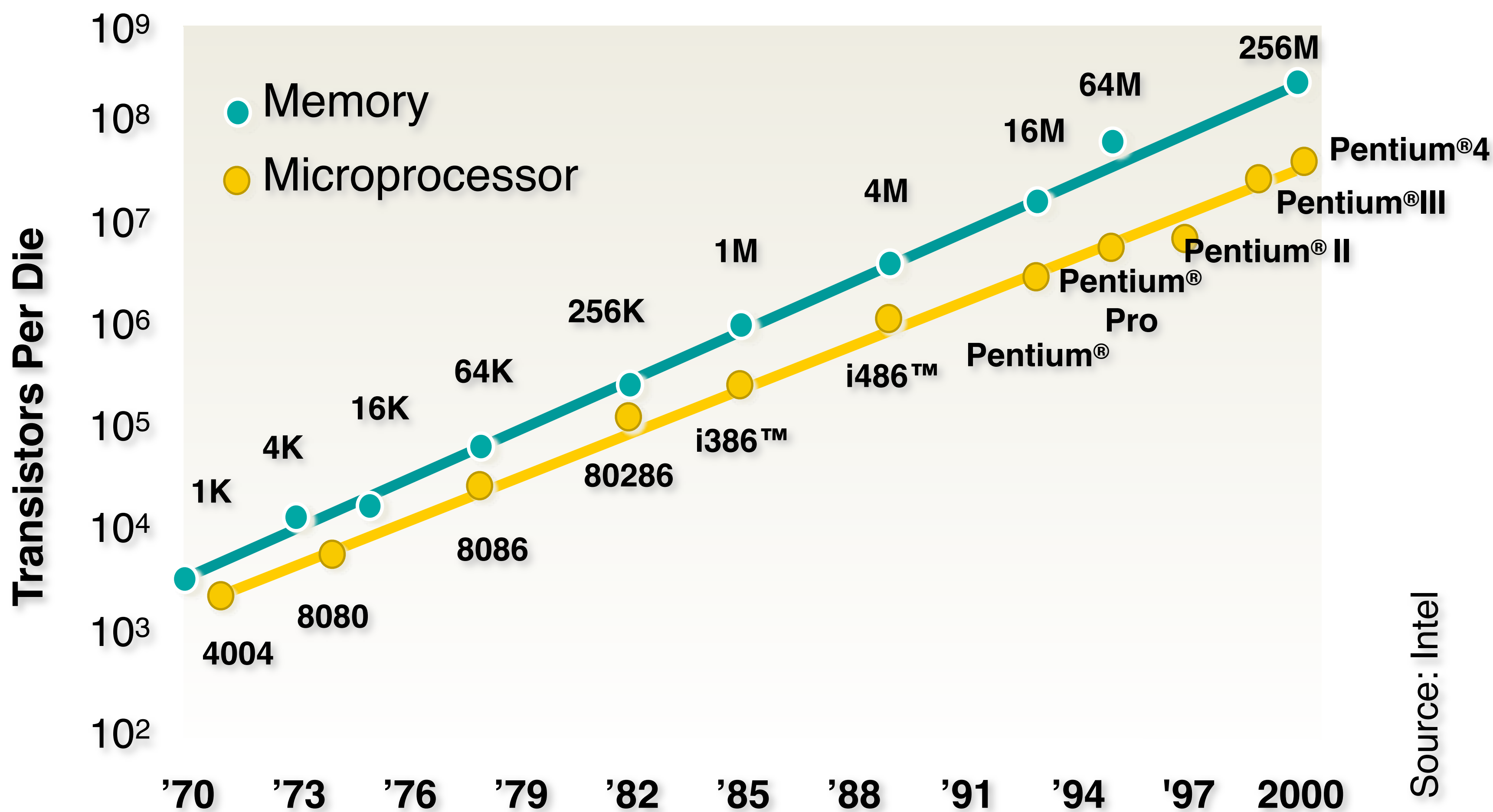
- The (practical) problem of parallelism today.
- Basic terms.
- Administration.

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

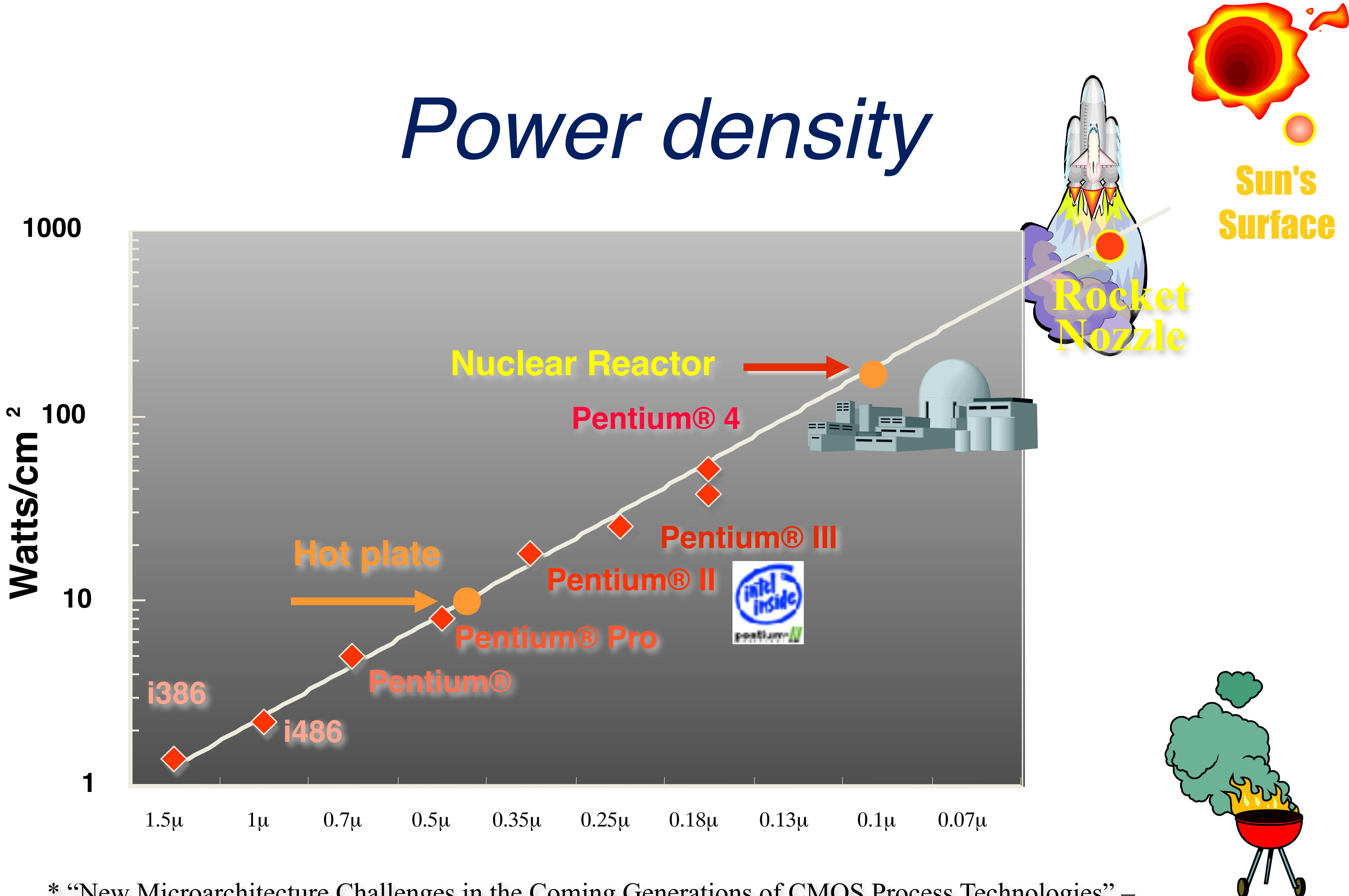

Memory
Microprocessor
Transistors Per Die
10⁹
10⁸
10⁷
10⁶
10⁵
10⁴
10³
10²
'70
'73
'76
'79
'82
'85
'88
'91
'94
'97
2000
1K
4K
16K
64K
256K
1M
4M
16M
64M
256M
4004
8080
8086
80286
i386™
i486™
Pentium®
Pentium® Pro
Pentium® II
Pentium®III
Pentium®4
Source: Intel

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!

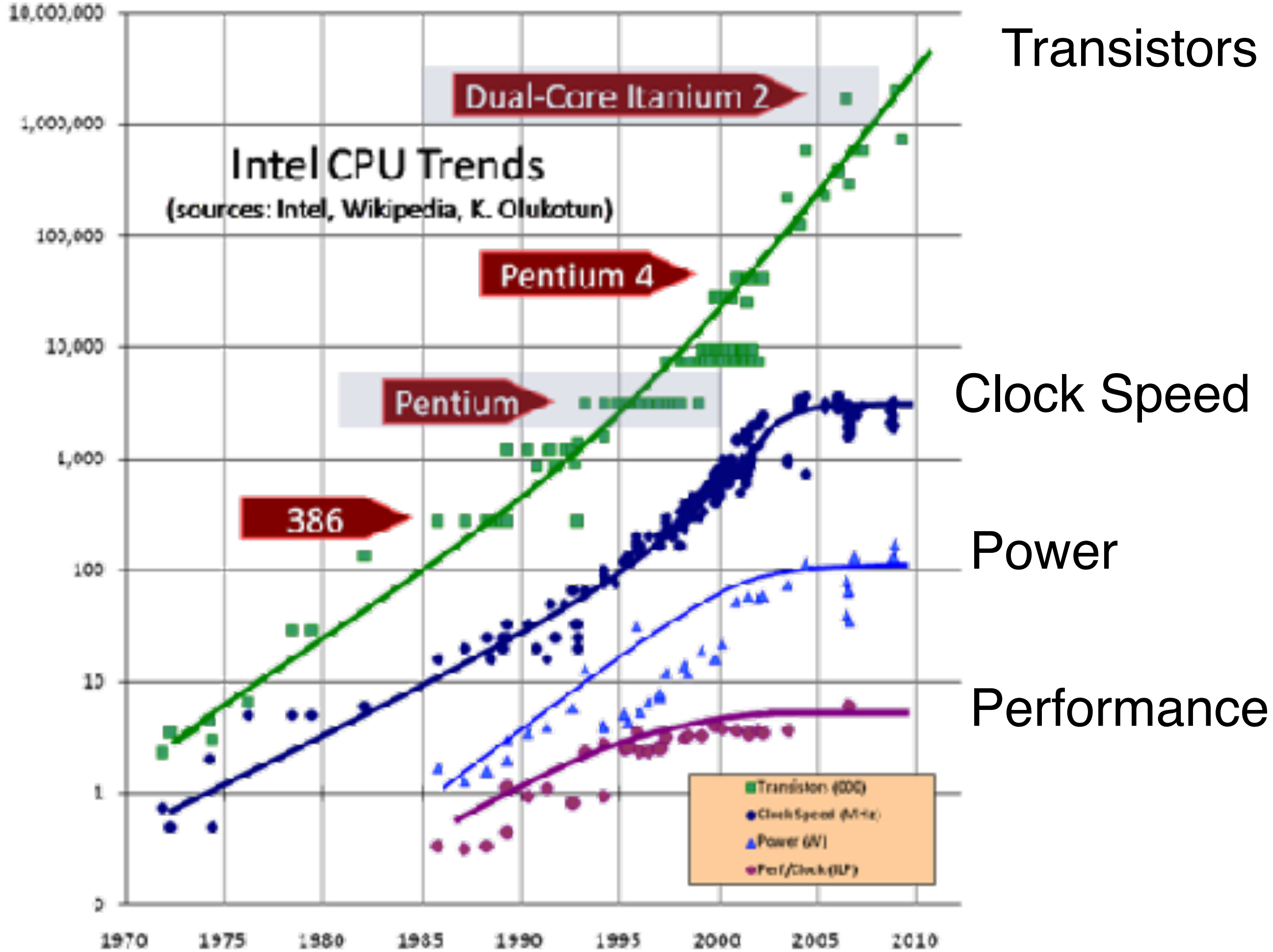
But then, power increases!

Power density



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

Processor Development

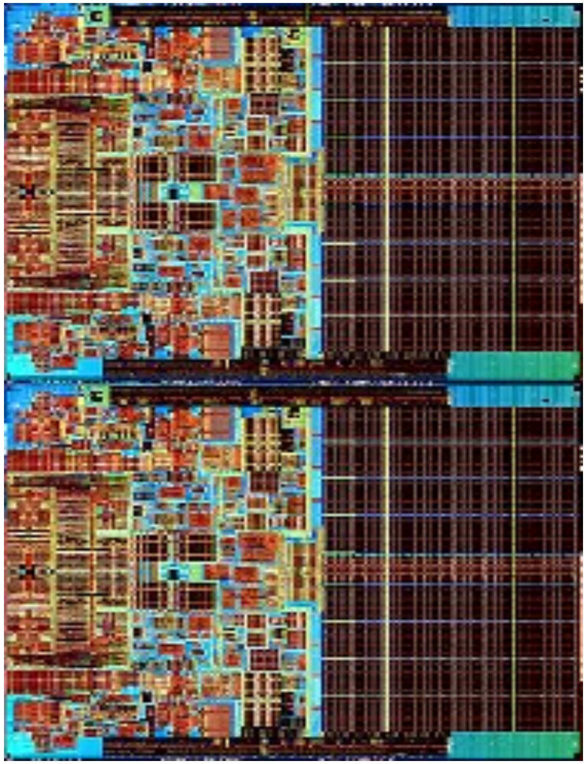


For Software: Free Lunch is Over

- 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, 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	CORRELATE	GLU	Mentat	Parafrase2	pC++
ACE	CPS	GUARD	Legion	Paralation	SCHEDULE
ACT++	CRL	HAsL.	Meta Chaos	Parallel-C++	SciTL
Active messages	CSP	Haskell	Midway	Parallaxis	POET
Adl	Cthreads	HPC++	Millipede	ParC	SDDA.
Adsmith	CUMULVS	JAVAR.	CparPar	ParLib++	SHMEM
ADDAP	DAGGER	HORUS	Mirage	ParLin	SIMPLE
AFAPI	DAPPLE	HPC	MpC	Parmacs	Sina
ALWAN	Data Parallel C	IMPACT	MOSIX	Parti	SISAL.
AM	DC++	ISIS.	Modula-P	pC	distributed smalltalk
AMDC	DCE++	JAVAR	Modula-2*	pC++	SMI.
AppLeS	DDD	JADE	Multipol	PCN	SONiC
Amoeba	DICE.	Java RMI	MPI	PCP:	Split-C.
ARTS	DIPC	javaPG	MPC++	PH	SR
Athapascan-0b	DOLIB	JavaSpace	Munin	PEACE	Sthreads
Aurora	DOME	JIDL	Nano-Threads	PCU	Strand.
Automap	DOSMOS.	Joyce	NESL	PET	SUIF.
bb_threads	DRL	Khoros	NetClasses++	PETSc	Synergy
Blaze	DSM-Threads	Karma	Nexus	PENNY	Telegrphos
BSP	Ease .	KOAN/Fortran-S	Nimrod	Phosphorus	SuperPascal
BlockComm	ECO	LAM	NOW	POET.	TCGMSG.
C*.	Eiffel	Lilac	Objective Linda	Polaris	Threads.h++.
"C* in C	Eilean	Linda	Occam	POOMA	TreadMarks
C**	Emerald	JADA	Omega	POOL-T	TRAPPER
CarlOS	EPL	WWWinda	OpenMP	PRESTO	uC++
Cashmere	Excalibur	ISETL-Linda	Orca	P-RIO	UNITY
C4	Express	ParLin	OOF90	Prospero	UC
CC++	Falcon	Eilean	P++	Proteus	V
Chu	Filaments	P4-Linda	P3L	QPC++	ViC*
Charlotte	FM	Glenda	p4-Linda	PVM	Visifold V-NUS
Charm	FLASH	POSYBL	Pablo	PSI	VPE
Charm++	The FORCE	Objective-Linda	PADE	PSDM	Win32 threads
Cid	Fork	LiPS	PADRE	Quake	WinPar
Cilk	Fortran-M	Locust	Panda	Quark	WWWinda
CM-Fortran	FX	Lparx	Papers	Quick Threads	XENOOPS
Converse	GA	Lucid	AFAPI.	Sage++	XPC
Code	GAMMA	Maisie	Para++	SCANDAL	Zounds
COOL	Glenda	Manifold	Paradigm	SAM	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"?
- Can we avoid using locks and just use low-level synchronization such as compare-and-swap?
- 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) has 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

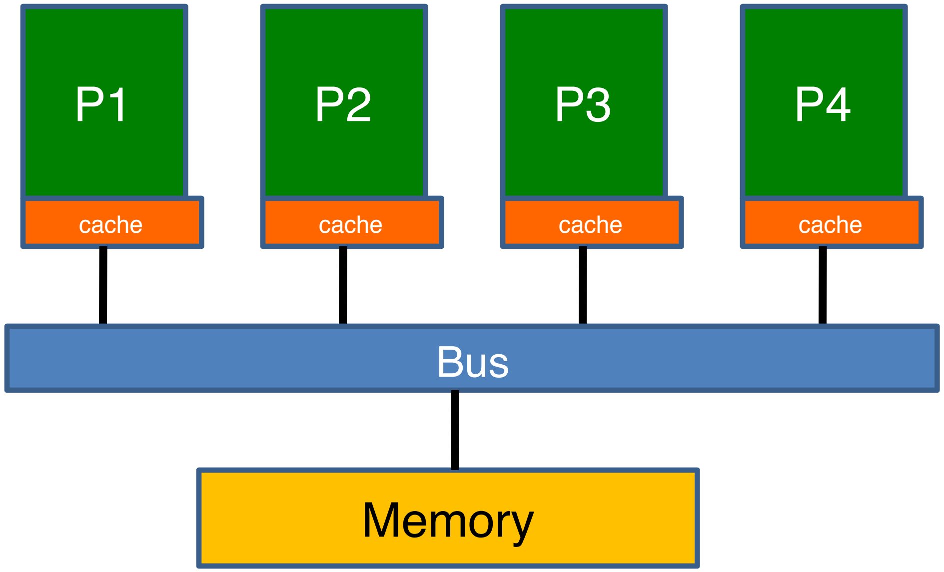
- Parallel platfroms are widely available.
- We do not know how to efficiently develop, debug and verify parallel software.
- A major challenge for the computing world today.
- We will study solution directions,
  - and technical issues in the design of scalable parallel algorithms.

Some Basics

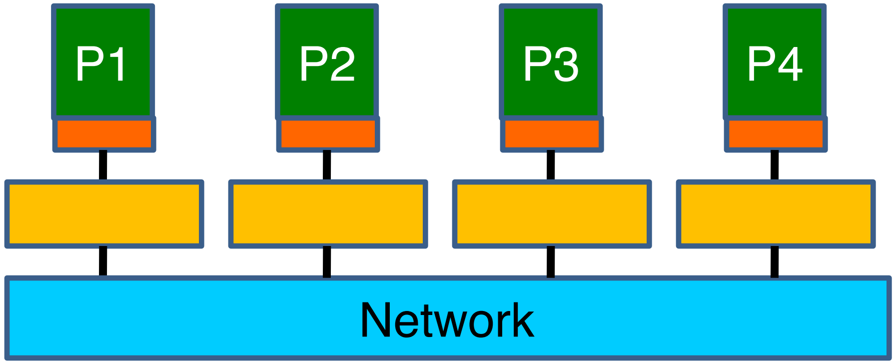
Platforms, Processes, Threads

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, which share its 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 today typically consist of multicores or multiprocessors.
- They may connect to create a distributed system.
- Software for parallel computation is usually written by spawning processes or creating 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 18 out of 36 papers.
  - This means you influenced the material.
  - This means everybody had a choice.

- Next week Topics 1 and 2 (introduction)
  - אדר אמיר ויעל דואק
- There will be no class on the third week Nov. 9th.
- Topics 4 will happen in the first hour of the fourth week and I will give the second hour.
- 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 VGA. (Check your connection 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%),
- 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.

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/seminar18

To Conclude…

- 36 papers offered for choice.
- 18 papers will be presented (according to your choice).

- Send me your talks the evening before!

- See you next week.