CS758: Multicore Programming

Introduction

Prof. David Wood
Fall 2009

Credits

- Material for these slides has been contributed by
  - Prof. Saman Amarasinghe, MIT
  - Prof. Mark Hill, Wisconsin
  - Prof. David Patterson, Berkeley
  - Prof. Marc Snir, Illinois

The "Software Crisis"

> "To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

-- E. Dijkstra, 1972 Turing Award Lecture

The First Software Crisis

- Time Frame: '60s and '70s
- Problem: Assembly Language Programming
  - Computers could handle larger more complex programs
- Needed to get Abstraction and Portability without losing Performance

How Did We Solve the First Software Crisis?

- High-level languages for von-Neumann machines
  - FORTRAN and C
- Provided "common machine language" for uniprocessors

Common Properties
Single flow of control
Single memory image

Differences:
Register File
ISA
Functional Units

The Second Software Crisis

- Time Frame: '80s and '90s
- Problem: Inability to build and maintain complex and robust applications requiring multi-million lines of code developed by hundreds of programmers
  - Computers could handle larger more complex programs
- Needed to get Composability, Malleability and Maintainability
  - High-performance was not an issue → left for Moore's Law

How Did We Solve the Second Software Crisis?

- Object Oriented Programming
  - C++, C# and Java
- Also…
  - Better tools
    - Component libraries, Purify
  - Better software engineering methodology
    - Design patterns, specification, testing, code reviews

Today: Programmers are Oblivious to Processors

- Solid boundary between Hardware and Software
- Programmers don't have to know anything about the processor
  - High level languages abstract away the processors
    - Ex: Java bytecode is machine independent
  - Moore's law does not require the programmers to know anything about the processors to get good speedups
- Programs are oblivious of the processor → work on all processors
  - A program written in '70 using C still works and is much faster today
- This abstraction provides a lot of freedom for the programmers

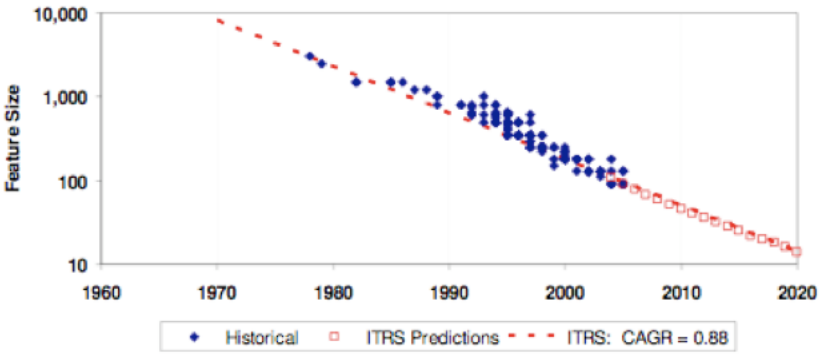
The Origins of a Third Crisis

- Time Frame: 2005 to 20??
- Problem: Sequential performance is left behind by Moore's law
- Needed continuous and reasonable performance improvements
  - to support new features
  - to support larger datasets
- While sustaining portability, malleability and maintainability without unduly increasing complexity faced by the programmer

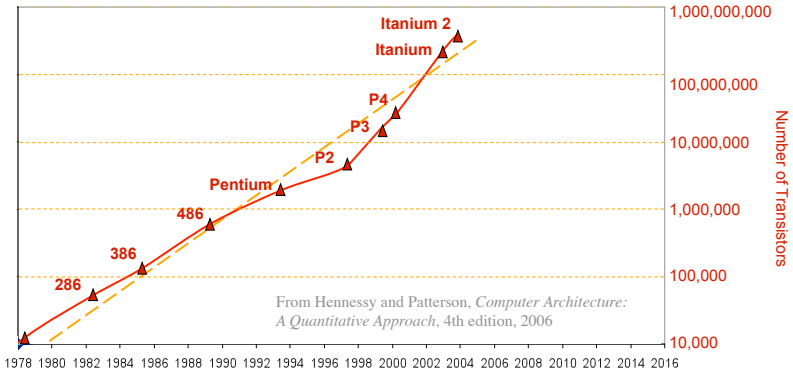
→ critical to keep-up with the current rate of evolution in software

March to multicore

- Silicon feature size



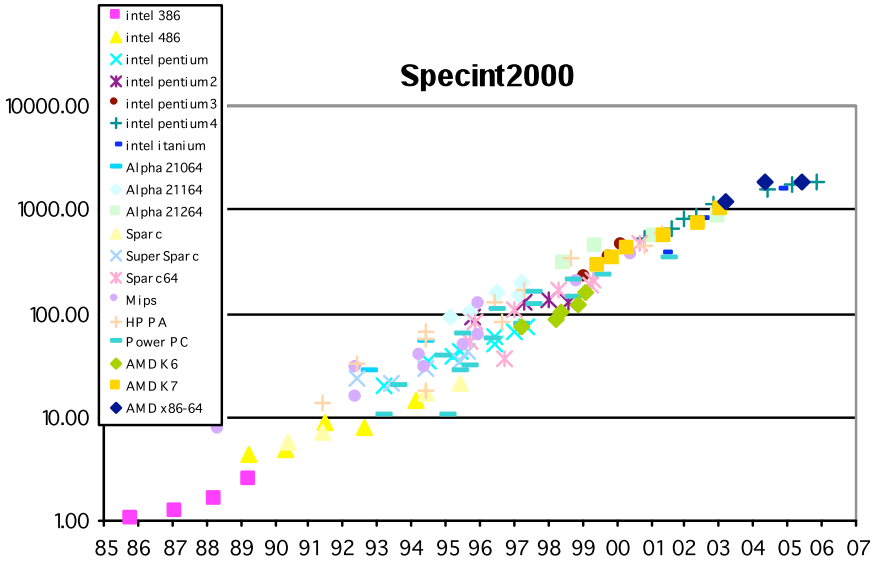
The March to Multicore: Moore's Law



Moore: # of transistors per chip will double every N months
N= 12, then 18, and now around 24

From David Patterson

The March to Multicore: Uniprocessor Performance (SPECint)

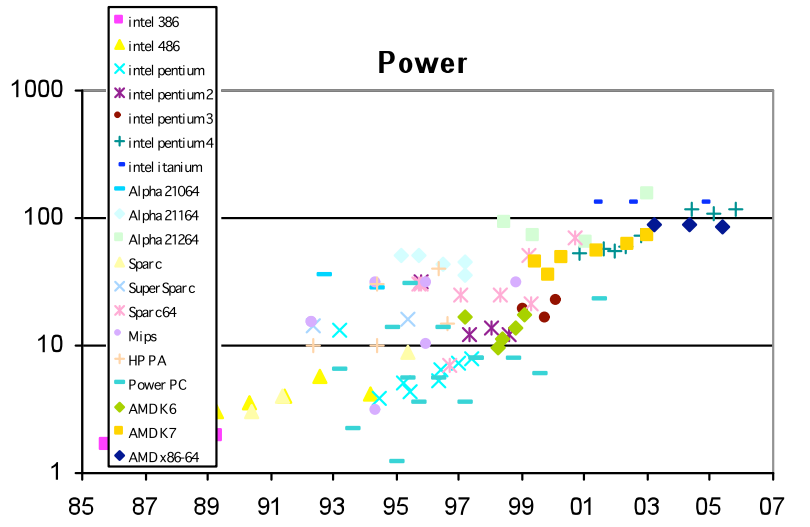


The March to Multicore: Uniprocessor Performance (SPECint)

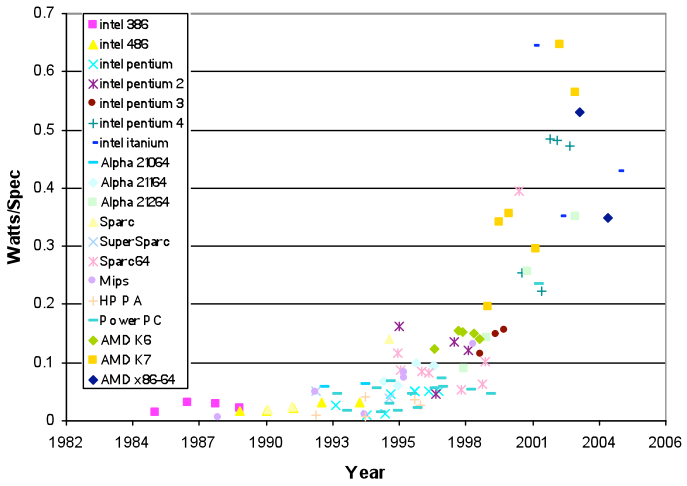
- General-purpose uniprocessors have stopped historic performance scaling
  - Power consumption ➔ Thermal limits
  - Wire delays
  - DRAM access latency
  - Diminishing returns of more instruction-level parallelism

From David Patterson

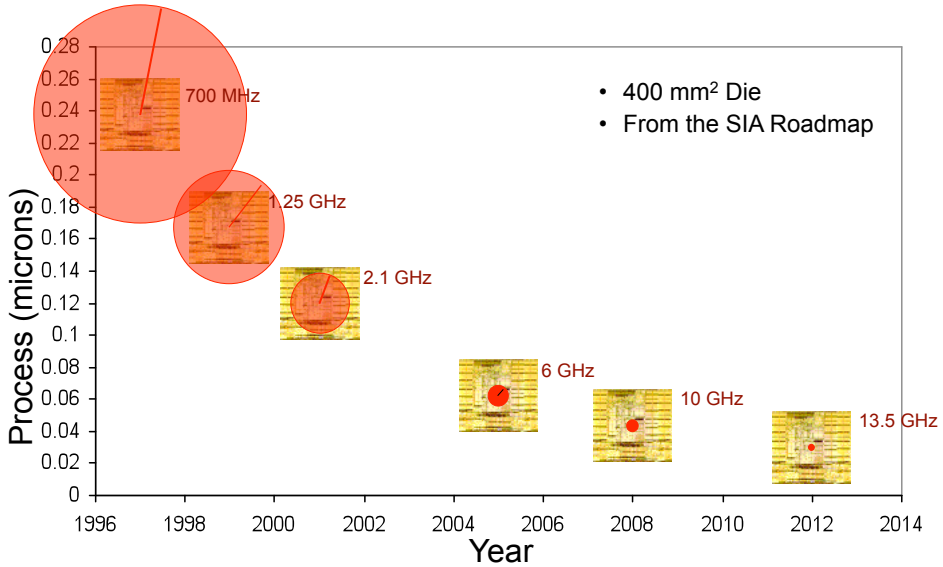
Power Consumption (watts)



Power Efficiency (watts/spec)



Range of a Wire in One Clock Cycle

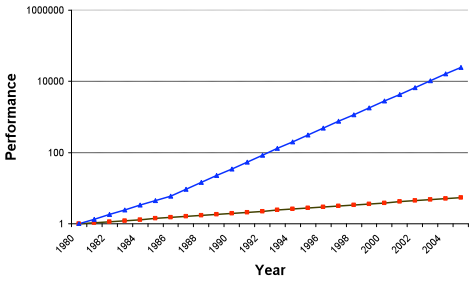


DRAM Access Latency

- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well

µProc
60%/yr.
(2X/1.5yr)

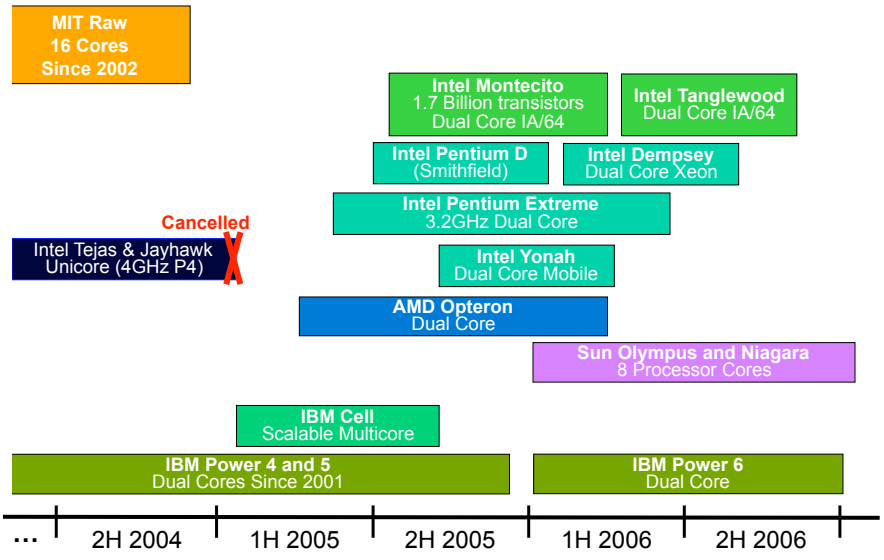
DRAM
9%/yr.
(2X/10 yrs)



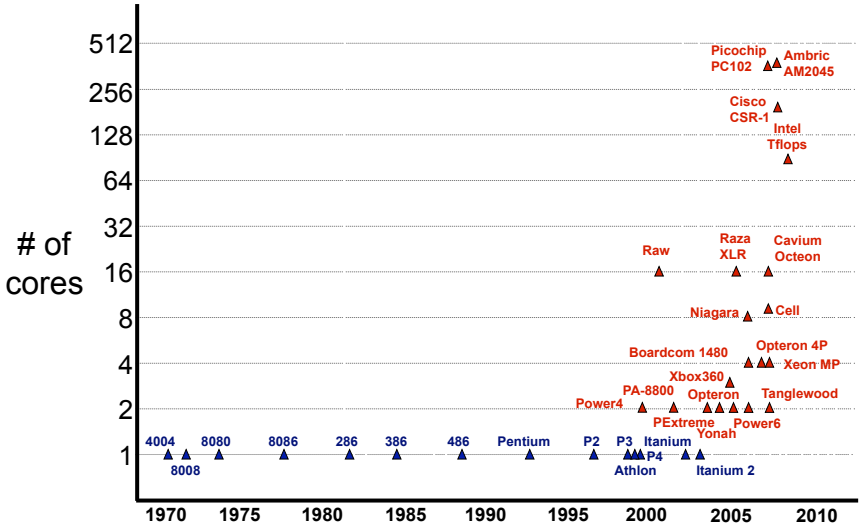
Diminishing Returns

- The ’80s: Superscalar expansion
  - 50% per year improvement in performance
  - Transistors applied to *implicit* parallelism
    - pipeline processor (10 CPI --> 1 CPI)
- The ’90s: The Era of Diminishing Returns
  - Squeaking out the last implicit parallelism
    - 2-way to 6-way issue, out-of-order issue, branch prediction
    - 1 CPI --> 0.5 CPI
  - performance below expectations
  - projects delayed & canceled
- The ’00s: The Beginning of the Multicore Era
  - The shift to Explicit Parallelism

Uniprocessors essentially extinct
Multicores are here



Multicores are Here



Programming Multicores

The Dilbert Approach

© Scott Adams, Inc./Dist. by UFS, Inc.

CS758: Multicore Programming

- Pthreads
- OpenMP
- Cilk++
- Threaded Building Blocks (TBB)
- Serialization Sets
- MapReduce
- Transactional Memory

CS758: Requirements and Outcomes

- Requirements
  - Substantial programming experience (C/C++)
  - At least one 700-level course in either architecture, programming languages, or operating systems
    - Some architecture background (at least CS552)
  - Instructor's consent
- Outcomes
  - Know fundamental concepts of parallel programming (both hardware and software)
  - Understand issues of parallel performance
  - Hands-on experience with several multicore platforms and programming models
  - Significant parallel programming project

Course operation

- See web site