Historically, Huge Performance Gains came from Huge Clock Frequency Increases

Unfortunately .........
Hardware Solution
Evolution of Computer Architectures
Micro-Scopnic View
Clock Rate Limits Have Been Reached

Source: Patterson, *Computer Organization and Design*
Hardware Solution
Evolution of Computer Architectures
Micro-Scopic View

Blocked by Heat and Power “Wall”
Single Core vs. Dual Core

Single Core clocked at $2f$

Throughput:
$2f$

Heat:
$8f^3$

Dual Core clocked at $f$

Throughput:
$f + f$

Heat:
$f^3 + f^3$
Hardware Solution
Evolution of Computer Architectures
Micro-Scopic View

- Instruction-Level Parallelism (ILP) was also Heavily Used
  - Implemented On-Chip via Hardware
  - Transparent to Software (No impact on Programmers)

- We will Study Two Types:
  - Pipelining (Intra-Instruction Parallelism)
  - Multi-Function Units (Inter-Instruction Parallelism)

- ILP has provided reasonable speedups in the past, Unfortunately…….
Effort | Performance | Hardware Solution
---|---|---
Scalar In-Order | Moderate Pipelining, Look-Ahead | Very Deep Pipe, Aggressive Look-Ahead
Made sense to go Superscalar and Multi-Function: Good ROI | Very little gain for substantial effort

Evolution of Computer Architectures

- Gain - to - Effort Ratio of ILP beyond “Knee” of Curve
- Diminishing Returns due to Increased Cost and Complexity of Extracting ILP
Evolution of Computer Architectures

Micro-Scopic View Summary

- Clock Frequency Scaling Limits have been Reached
- Instruction Level Parallelism Limits have been Reached
- Era of Single Core Performance Increases has Ended
- No More “Free Lunch” for Software Programmers
  - Multiple Cores Will Directly Expose Parallelism to SW
- All Future Micro-Processor Designs will be Multi-Core
  - Evident in Chip Manufacturer’s RoadMaps
Hardware Solution
Evolution of Computer Architectures

Micro

Macro

Sequential Processing → Parallel Processing
Hardware Solution

Technology Curve

Cost / Performance vs. Performance

Under-Utilization

Optimum Utilization

Over-Utilization
Hardware Solution
Technology Curve

Cost / Performance
vs.
Performance

Sequential Processing

Parallel Processing

$\text{Cost}$

$\text{Performance}$

$\text{\$\$\$}$

$\text{\$}$

$\text{X}$

$\text{2X}$

Performance
Parallel Processing is really an Evolution in Micro- and Macro-Architecture Hardware that provides a Solution to:
• The Heat and Power Wall
• The Limitations of ILP
• Cost-Effective Higher Performance

Parallel Processing is also a Software Challenge
Software Challenge

Change in Hardware Requires Change in Software

- The Car (Hardware) has Changed
  - From a Sequential Engine To a Parallel Engine
- The Driver (Software) Must Change Driving Techniques
  - Otherwise, Sub-Optimal Performance will Result
Software Challenge
Lack of Good PP Tools

- Lack of Tools Compounds Problem
  - Existing Tool Chain only for Sequential Programming

- Need New Parallel Programming Tools & Infrastructure
  - Effective Models for Parallel Systems
  - Constructs to make Parallel Architecture more Visible
  - Languages to More Clearly Express Parallelism
  - Reverse Engineering Analysis Tools
    - To Assist with Conversion of Sequential to Parallel
      - Especially for Optimized Sequential Code
Software Challenge
Race Conditions

- Parallelism can Give Rise to a New Class of Problems
  - Caused by the Interactions Between Parallel Threads

- Race Condition:
  Multiple Threads Perform Concurrent Access to the Same Shared Memory Location

- Threads “Race” Against Each Other
  - Execution order is assumed but cannot be guaranteed
  - Outcome depends on which one wins (by chance)
  - Results in Non-Deterministic Behavior
Race Conditions are Especially Hard to Detect & Debug

- Errors are Very Subtle
  - No Apparent “Failure” Occurs
  - Program Continues to Run “Normally”
  - Program Completes “Normally”
- Errors are Intermittent
  - Hard to Reproduce and Diagnose
- Errors Can Slip Through SQA Testing Process
  - Potential Lurking Bug

- Most Common Error in Parallel Programs
Semaphores Offer a Solution to Race Conditions
- However Semaphores themselves can cause Problems:
  - Introduce Overhead
  - Can Create Bottlenecks
  - Mutually Exclusive (one-at-a-time) Access
Another Potential Problem Arising From Parallelism

Deadlock:

Two or More Threads are Blocked because Each is Waiting for a Resource Held by the Other
Software Challenge
Deadlock

- Not as Hard as Race Conditions
  - Errors are More Obvious
    - System Usually Freezes

- But Similar to Race Conditions
  - Errors are Intermittent
    - Hard to Detect, Reproduce, Diagnose, Debug
  - Errors Can Slip Through SQA Testing Process
    - Potential Lurking Bug

- Another Common Error in Parallel Programs
Software Challenge
Concurrent vs. Parallel

- Time-Sharing = Multi-Tasking = Multiplexing = Concurrent
  - One Processor is being shared (switched quickly) between tasks making them appear to be “Concurrent”
  - But it’s essentially just an illusion, because at any instant in time, only one task is really executing

- Concurrency is not the same as true Parallelism

**Concurrent:** Two Threads are *In Progress* at Same Time

**Parallelism:** Two Threads are *Executing* at Same Time
Software Challenge
Concurrent vs. Parallel

- SW Problem is Harder than that from “Time-Sharing” Era
  - Multi-Cores (Micro) & Multi-Nodes (Macro) HW enable:
    - Not Just “Multi-Tasking” or Concurrency, but
    - True Parallelism

- Potential Problem when migrating “Multi-Tasking” Code

- Consider a SW Application Programmed with Two Tasks:
  - One task is assigned a low priority; other a high priority
  - In Multi-Tasking: LP task cannot run until HP is done
    - Programmer could have assumed Mutual Exclusion
  - In Parallel System: LP and HP can run at Same Time
Software Challenge

Debugging

- Harder Because of Intermittent, Non-Deterministic Bugs
- Time Sensitive (Temporally Aware) SW Tools Needed
- New Parallel Debugging Tools Required
  - Need to Exert Temporal Control over Multiple Threads
  - Ideal Debugger would have:
    - Reverse Execution Capability (cycle-accurate undo)
    - Instant Replay Capability (w/ accurate time counter)
- Cannot Use Ad-hoc Debugging via PRINT Statements
  - Adds Extra Instructions which could Change Timing
Software Challenge - Technical

Amdahl’s Law

- Parallel Speedup is Limited by the Amount of Serial Code

![Graph showing Maximum Theoretical Speedup from Amdahl's Law](image)
Software Challenge - Business
Changing Technology Curves is Hard

- Never Ride Technology Curve “Up” into Over-Utilization

Cost / Performance vs. Performance

Sequential

Parallel

Cost

Performance

Performance
Parallel Programming is Hard
  • More Complex
  • Lack of Tools
  • New Type of Bugs
    • Race Conditions
    • Deadlocks
  • Harder to Debug, Test, Profile, Tune, Scale

Parallel Programming is a Software Challenge
Opportunities

- The Universe is Inherently Parallel
  - Natural Physical and Social / Work Processes
    - Weather, Galaxy Formation, Epidemics, Traffic Jam
- Can Leverage Unique Capabilities offered by Parallelism
- Add New Features via Separate Parallel Modules
  - Avoids Re-engineering of Old Module
  - More Functionality
  - No Increase in Wall Processing Times
- Speculative Computation
  - Precompute alternatives to Minimize Response Time
    - e.g.) Video Game Up / Down / Left / Right
    - More Responsive User Interfaces
Opportunities

- (Yet) Undiscovered Technical Opportunities
- New Parallel Algorithms
- Super-Linear Speedups
  - Parallel Computer has $N$ times more Memory
  - Larger % of SW can fit in upper Levels of Memory Hierarchy
  - “Divide and Conquer” leverages faster Mem

• An Important Reason for using Parallel Computers
Will Feedback Cycle Prevail?

- If Hardware goes Parallel, Will Software go Parallel Too?