CS4617 Computer Architecture

Lecture 1

Dr J Vaughan

September 8, 2014

Introduction

"Today less than $500 will purchase a mobile computer that has more performance, more main memory and more disk storage than a computer bought in 1985 for $1 million."
Hennessy & Patterson

Advances in technology

- Innovations in computer design
- Microprocessors took advantage of improvements in IC technology
- Led to increased number of computers being based on microprocessors

Marketplace changes

- Assembly language programming largely unnecessary except for special uses
- Reduced need for object code compatibility
- Operating systems standardised on a few such as Unix/Linux, MicroSoft Windows, MacOS
- Lower cost and risk of producing a new architecture

RISC architectures, early 1980s

- Exploited instruction-level parallelism
- Pipelining, multiple instruction issue
- Exploited caches

RISC raised performance standards

- DEC VAX could not keep up
- Intel adapted by translating 80x86 to RISC internally
- Hardware overhead of translation negligible with large transistor counts
- When transistors and power restricted, as in mobile phones, pure
- RISC dominates
- ARM

Effects of technological growth

1. Increased computing power
2. New classes of computer
   - Microprocessors $\longrightarrow$ PCs, workstations
   - Smartphones, tablets
   - Mobile client services $\longrightarrow$ server warehouses
3. Moore's Law: microprocessor-based computers dominate across entire range of computers
4. Software development can exchange performance for productivity
   - Performance has improved $\times$25000 since 1978
   - C, C++
   - Java, C#
   - Python, Ruby
5. Applications have evolved; speech, sound, video now more important

Limits

- Now, single-processor performance improvement has dropped to less than 22% per year
- Problems: Limit to amount of IC power than can be dissipated by air- cooling
- Limited amount of exploitable instruction-level parallelism in programs
- 2004: Intel cancelled its high-performance one-processor projects
- Future in several processors per chip

Parallelism

- ILP succeeded by DLP, TLP, RLP
- Data-level parallelism (DLP)
- Thread-level parallelism (TLP)
- Request-level parallelism (RLP)
- DLP, TLP, RLP require programmer awareness and intervention
- ILP is automatic; programmer need not be aware

Classes of computers

- Personal Mobile Device (PMD)
- Desktop
- Server
- Clusters/Warehouse-scale computers
- Embedded

Two kinds of parallelism in applications

- Data-level parallelism (DLP): many data items can be operated on at the same time
- Task-level parallelism (TLP): tasks can operate independently and in parallel

Four ways to exploit parallelism in hardware

1. ILP exploits DLP in pipelining and speculative execution
2. Vector processors and Graphics Processing units use DLP by applying one instruction to many data items in parallel
3. Thread-level parallelism uses DLP and task-level parallelism in cooperative processing of data by parallel threads.
4. Request-level parallelism: Parallel operation of tasks that are mainly independent of each other

Flynn's parallel architecture classifications

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
- Multiple instruction streams, single data stream (MISD)
- Multiple instruction streams, multiple data streams (MIMD)
- SISD: One processor, ILP possible
- SIMD: Vector processors, GPU, DLP
- MISD: No computer of this type exists
- MIMD: Many processors:
  - Tightly-coupled - TLP
  - Loosely-coupled - RLP

Instruction Set Architecture (ISA): class determinants

- Memory Addressing
- Addressing Modes
- Types and sizes of operands
- Operations
- Control flow
- ISA encoding

Class of ISA

- General-purpose architectures: operands in registers or memory locations
- Register-memory ISA: 80x86
- Load-store ISA: ARM, MIPS

Memory addressing

- Byte addressing
- Alignment: Byte/Word/doubleword: Required?
- Efficiency: Faster if bytes aligned?

Dependability

- Service Level Agreement (SLA) guarantees a dependable level of service provided
- States of service with respect to an SLA
  1. Service accomplishment: service delivered
  2. Service interruption: delivered service less than SLA
- State transitions
  - Failure (state 1 to state 2)
  - Restoration (state 2 to state 1)
- *Module Reliability* measures time to failure from an initial instant
- *Mean time to failure* (MTTF) is a reliability measure
- Failure rate $= 1/\text{MTTF} =$ *failures in time* (FIT)
- Service Interruption Time = Mean time to repair (MTTR)
- Mean time between failures (MTBF) = MTTF + MTTR

Module availability

- A measure of service accomplishment
- For non-redundant systems with repair,
  Module availability $= \frac{MTTF}{MTTF+MTTR}$

Example: Disk subsystem

- 10 disks, each with MTTF = 1000000 hours
- 1 ATA controller, MTTF = 500000 hours
- 1 power supply, MTTF = 200000 hours
- 1 fan, MTTF = 200000 hours
- 1 ATA cable, MTTF = 1000000 hours
- Assume lifetimes are exponentially distributed and failures are independent
- Calculate system MTTF

Solution

- $$\begin{aligned} \textit{Failure rate}_{system} &= \frac{10}{1000000} + \frac{1}{500000} + \frac{1}{200000} \\ &+ \frac{1}{2000000} + \frac{1}{1000000} \\ &= \frac{10+2+5+5+1}{1000000} = \frac{23}{1000000} \end{aligned}$$
- The rate of failure, FIT (*failures in time*) is reported as the numbers of failures per 10^9 hours, so here the system failure rate is 23000 FIT
- $MTTF_{system} = \frac{1}{\textit{Failure rate}_{system}} = \frac{10^9}{23000} = 43500\ \textit{hours} = \textit{just under 5 years}$

Redundancy

- To cope with failure, use time or resource redundancy
- Time: Repeat the operation
- Resource: Other components take over from failed component
- Assume dependability restored fully after repair/replacement

Example: redundancy

- Add 1 redundant power supply to previous system
- Assume component lifetimes are exponentially distributed
- Assume component failures are independent
- MTTF for redundant power supplies is the mean time until one fails divided by the chance that the second fails before the first is replaced
- If the chance of a second failure is small, MTTF for the pair is large
- Calculate MTTF

Solution to redundant power supply example

- Mean time until one failure = $MTTF_{power\ \ supply}/2$
- MTTR divided by (mean time until the other power supply fails) gives an approximation of Prob(second failure)
- $$MTTF_{power\ \ supply\ \ pair} = \frac{MTTF_{power\ \ supply}/2}{\frac{MTTR_{power\ \ supply}}{MTTF_{power\ \ supply}}} = \frac{MTTF^2_{power\ \ supply}/2}{MTTR_{power\ \ supply}} = \frac{MTTF^2_{power\ \ supply}/2}{2 \times MTTR_{power\ \ supply}}$$
- $MTTF_{power\ \ supply\ \ pair} \approx 850000000 \approx 4150$ times more reliable

Measuring performance

- Response time $= t_{finish} - t_{start}$
- Throughput = Number of tasks completed per unit time
- "X is *n times faster than Y*"
- $\frac{Execution\ \ time_Y}{Execution\ \ time_X} = n$
- $n = \frac{\frac{1}{Performance_Y}}{\frac{1}{Performance_X}}$
- $n = \frac{Performance_X}{Performance_Y}$

Suites of benchmark programs to evaluate performance

- EEMBC: Electronic Design News Embedded Microprocessor Benchmark Consortium
  - 41 kernels to compare performance of embedded applications
- SPEC: Standard Performance Evaluation Corporation
  - www.spec.org
  - SPEC benchmarks cover many application classes
  - SPEC 2006: Desktop benchmark, 12 integer benchmarks, 17 floating point benchmarks
  - SPEC Web: Web server benchmark
  - SPECSFS: Network file system performance, throughput-oriented
- TPC: Transaction Processing Council
  - www.tpc.org
  - Measure ability of a system to handle database transactions
  - TPC-C: Complex query environment
  - TPC-H: Unrelated queries
  - TPC-E: Online transaction processing (OLTP)

Comparing performance

- Normalise execution times to a reference computer
- $SPECRatio = \frac{Execution\ \ time\ \ on\ \ reference\ \ computer}{Execution\ \ time\ \ on\ \ computer\ \ being\ \ measured}$
- If SPECRatio of computer A on a benchmark is 1.25 times higher than computer B, then
-

$$
\begin{aligned}
1.25 &= \frac{SPECRatio_A}{SPECRatio_B} \\
&= \frac{\frac{Executiontime_{reference}}{Executiontime_A}}{\frac{Executiontime_{reference}}{Execution_B}} \\
&= \frac{Executiontime_B}{Executiontime_A} \\
&= \frac{Performance_A}{Performance_B}
\end{aligned}
$$

Combining SPECRatios

- To combine the SPECRatios for different benchmark programs, use the geometric mean
- Geometric mean $= \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$

Design principles for better computer performance

- Take advantage of parallelism
- Principle of locality
- Focus on the common case
  - Amdahl's Law highlights the limited benefits accruing from subsystem performance improvements

Exploit parallelism

- Server benchmark improvement: spread requests among several processors and disks
  Scalability: ability to expand the number of processors and number of disks
- Individual processors
  Pipelining: instruction-level parallelism
- Digital design
  - Set-associative cache
  - Carry-lookahead ALU

Principle of Locality

- Program execution concentrates within a small range of address space and that range changes only intermittently.
  - Temporal locality
  - Spatial locality

Focus on the common case

- In a design trade-off, favour the frequent case
- Example: optimise the Fetch & Decode unit before the multiplication unit
- Example: optimise for no overflow since it is more common than overflow

Amdahl's Law

- $Speedup = \frac{Execution\ time\ for\ entire\ task\ without\ using\ enhancement}{Execution\ time\ for\ entire\ task\ using\ enhancement\ when\ possible}$
- $Speedup_{overall} = \frac{Execution\ time_{old}}{Execution\ time_{new}}$
- $Speedup_{overall} = \frac{1}{(1-Fraction_{enhanced})+\frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$