1.1 Introduction

Instruction-level parallelism growth stunted.

Future:

- Data-level parallelism
- Thread-level parallelism
- Request-level parallelism (Warehouse-scale computers)

1.2 Classes of Computers

Two kinds of parallelism in applications:

1. Data-level parallelism (DLP): arises because there are many data items that can be operated on at the same time.
2. Task-level parallelism (TLP): arises because tasks of work are created that can operate independently and largely in parallel.

Computer hardware can exploit these two application parallelism in four major ways:

1. *Instruction-level parallelism* (ILP): exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution.
2. *Vector architecture and graphic processors units* (GPUs): exploit data-level parallelism by applying a single instruction to a collection of data in parallel.
3. *Thread-level parallelism*: exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction among parallel threads.
4. *Request-level parallelism*: exploits parallelism among largely decoupled tasks specified by the programmer or the operating system. (Super computer systems)

All computers can be placed into four categories:

1. *Single instruction stream, single data stream* (SISD): This category is the *uniprocessor.* This is like a standard sequential computer, but it can exploit instruction-level parallelism. (CH 3 goes over SISD architectures using ILP techniques like superscalar)
2. *Single instruction stream, multiple data streams* (SIMD): Same instruction is executed by multiple processors using different data streams. SIMD architecture exploits data-level parallelism by applying the same operations to multiple items of data in parallel.
    - Each processor has its own data memory
    - There is a single instruction memory and control processor: it fetches and dispatches instructions.
    - Popular in vector architectures
3. *Multiple instruction streams, single data streams* (MISD): No commercial multiprocessor of this type has been built to date, but it rounds out this simple classification.
4. *Multiple instruction streams, multiple data streams* (MIMD): Each processor fetches its own instructions and operates on its own data, and it targets task-level parallelism.
    - MIMD is more flexible than SIMD thus generally more applicable
    - Inherently more expensive than SIMD
    - MIMD can exploit data-level parallelism but overhead is likely to be higher than SIMD
    - Large overhead means grain-size must be large to be used efficiently

Overall, many parallel processors are hybrids of the four model classes.

1.3 Defining Computer Architecture

Instruction set architecture (ISA): the actual programmer-visible instruction set.

- ISA is the boundary between the software and hardware.

7 Dimensions of ISA

1. *Class of ISA*: nearly all ISA are classified as general-purpose register architectures, where the operands are either registers or memory locations.
    - 80x86 has 16 general purpose registers and 16 that can hold floating point data
    - MIPS has 32 general-purpose and 32 floating point registers
    - Two popular classes are
        1. Register-memory (such as 80x86): access memory as part of many instructions
        2. Load-stores (such as MIPS or ARM): acces memory only with load or store instructions.
    - All recent ISAs are load-store
2. *Memory addressing*: Virtually all desktop and server computers use byte addressing to access memory operands.
    - Some architecture such as MIPS require objects be aligned.
    - 80x86 doesn't require alignment, but would access faster if it were.
3. *Addressing modes*: specify the address of a memory object
    - MIPS addressing modes: Register, immediate (for constants), and displacement
        - A constant offset is added to a register to form the memory address
    - 80x86 addressing modes: supports MIPS mode plus three variations of displacement
        1. No register (absolute)
        2. Two register (based indexed with displacement)
        3. two registers where on register is multiplied by the size of the operand in bytes (based with scaled index and displacement)
        - It has more like the last three, minus the displacement field, plus register indirect, indexed, and based with scaled index.
4. *Types and sizes of operands*: Supports operand size of
    - 8-bit (ASCII characters)
    - 16-bit (Unicode character or half word)
    - 32-bit (Integer or word)
    - 64-bit (Double word or long integer)
    - IEEE 754 floating in 32-bit (Single precision)
    - 64-bit (Double precision)
    - 80x86 supports 80-bit floating point (Extended double precision)
5. *Operations*: General categories of operations are data transfer, arithmetic logical, control, and floating points.
6. *Control flow instructions*: Virtually all ISAs, including these three, support conditional branches, unconditional jumps, procedure calls, and returns.
    - All three uses PC-relative addressing: branch address is specified by an address field that is added to the PC.
7. *Encoding an ISA*: Two basic choices on encoding: fixed length and variable length.

Designing the Organization and Hardware

Implementation of a computer has two components:

1. Organization: high-level aspects of a computer's design such as:
    - Memory system
    - Memory interconnect
    - Design of the internal processor CPU
2. Hardware: specifics of a computer, including:
    - Detailed logic design
    - Packaging technology of computer

Microarchitecture: used instead of organization sometimes.

Multicore: Multiprocessors microprocessors

Architecture: covers three aspects of computer design

1. Instruction Set Architecture
2. Organization or Microarchitecture
3. Hardware

1.8 Measuring, Reporting, and Summarizing Performance

Execution time (response time): time between start and completion of an event.

Throughput: total amount of work done in a given time.

"X is n times faster than Y"

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = \text{n}$$

Since execution time is the reciprocal of performance the following relationship holds:

$$\text{n} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

Latency: time to complete a task (response time, elapsed time)

Benchmarks

Benchmark: tool use to measure performance.

There are issues with benchmarks (people can cheat) There are three different approaches to control conditions to make sure benchmarks truly measure performance:

1. No source code modifications allowed
2. Source code modifications allowed but are essentially impossible
3. Source code modifications are allowed, as long as the modified version produces the same output

Desktop Benchmarks

Desktop Benchmarks divide into two broad classes:

1. Processor-intensive benchmarks
2. Graphics-intensive benchmarks

Server Benchmarks

There are multiple types of server benchmarks:

- Processor throughput-oriented benchmark: tests SPEC CPU
- Tests for many I/O activity
- Transaction-processing benchmarks: measure ability of a system to handle transactions that consist of database accesses and updates

Reporting Performance Results

Reproducibility: list everything another experimenter would need to duplicate the results

Normalize execution times to a reference computer by dividing the time on the reference computer by the time on the computer being rated. This is the best way to see the performance increase of two systems.

$$\text{n} = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\frac{\text{Execution time}_{reference}}{\text{Execution time}_A}}{\frac{\text{Execution time}_{reference}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

SPECRatio is a ratio rather than an absolute execution time, the mean must be computed using the geometric mean.

$$\text{Geometric mean} = \sqrt[n]{\prod_{i=1}^{n} \text{sample}_i}$$

1.9 Quantitative Principles of Computer Design

Section is about guidelines/principles that are useful to design

- Using parallel computing to boost scalability and over performance
- Principle of locality: programs tend to reuse data and instructions they have used recently.
  - Temporal locality: recently accessed items are likely to be accessed again in the future
  - Spatial locality: items whose addresses are near one another tend to be referenced close together in time.
- Focus on Common Case
  - When making a design trade off, favor the frequent case over the infrequent case.

Amdahl's Law

Amdahl's Law: the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

Amdahl's law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

1. The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement
2. The improvement gained by the enhanced execution mode, that is, how much faster the task would run if the enhanced mode were used for the entire program