CS-417
COMPUTER SYSTEMS MODELING

Spring Semester 2020

Batch: 2016-17

(LECTURE # 4)

FAKHRA AFTAB

LECTURER

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

Recap of Lecture # 3

Amdahl’s Law
Means vs. End based Metric
Performance Measures
Review of Computer Systems Performance Evaluation

Chapter # 1 (Cont'd)

COMPUTER SYSTEMS PERFORMANCE MODELING AND EVALUATION

Benchmarking

Benchmarking (in computing) is the act of running a set of computer programs, or other operations, in order to assess the relative performance of an object in the computer system.

Benchmark: elaborately-designed benchmarking programs.

- Usually associated with assessing performance of hardware.
- But the technique is also applicable to software.

Purpose of Benchmarking

- Provides a method of comparing the performance of various systems (or subsystems) across different architectures.
- Benchmarks are designed to mimic a particular type of workload on a component or system.
- It also helps to choose effective system upgrades and to identify potential bottlenecks.

Types of Benchmarks

i) Addition Instruction

- Historically, the performance of computer system was synonymous with that of the processor.
- Addition was the most frequent instruction.
- The machine with the fastest addition operation would produce the best overall performance when executing any application program.

Types of Benchmarks (Cont'd)

ii) Instruction Mixes

- Instruction sets in which each instruction take only the minimum number of cycles required to complete its particular operation were designed.
- Execution time of a single instruction was no longer adequate to summarize performance.
- Gibson proposed instruction mix as benchmarking.
- The basic idea was to categorize all of the instructions into different classes.
- Number of instructions of each class executed by the particular collection of programs was then used to form a weighted average.

Example: Gibson Instruction Mix

1.	Load and Store	13.2
2.	Fixed-Point Add/Sub	6.1
3.	Compares	3.8
4.	Branches	16.6
5.	Float Add/Sub	6.9
6.	Float Multiply	3.8
7.	Float Divide	1.5
8.	Fixed-Point Multiply	0.6
9.	Fixed-Point Divide	0.2
10.	Shifting	4.4
11.	Logical And/Or	1.6
12.	Instructions not using regs	5.3
13.	Indexing	18.0
	Total	100

1959,
IBM 650
IBM 704

Types of Benchmarks (Cont'd)

iii) Kernel

- Small benchmark codes designed to measure basic architectural features of parallel machines
  - Normally abstracted from actual program
  - Easy to port to many different computer systems.
  - Most kernels are processing kernels – the only focus is on processor's performance

Popular kernels

- Livermore loops consists of 24 do loops, some of which can be vectorized, and some of which cannot
- Linpack benchmark (contains basic linear algebra subroutine written in FORTRAN language)
- Tower of Hanoi, 8-Queen Puzzle Problem, Sorting etc.

Types of Benchmarks (Cont'd)

iv) Component Benchmark / Micro-benchmark

- Consists of a relatively small specific piece of code to measure performance of computer's specific portion/component.
- Example: A small program written to test only the processor-memory interface, the input/output subsystem, or the FP execution unit independent of the other components of the system.
- Determine whether the performance capabilities of all the components within a system are balanced.
- Provide values for important parameters for a simulation of the system.
- Requires programmer to have a deep understanding of system being tested.

Types of Benchmarks (Cont’d)

Some more examples:

- The most important component in a particular case depends on the individual's usage patterns.
  - A gamer seeking the best possible frame rates will probably be better served by a faster GPU than by more memory.
  - A casual user seeking more responsive system may benefit by upgrading a slow hard drive to fast solid-state drive.
  - It is necessary to tailor the suite of benchmarks according to user’s specific needs and then weigh the individual test results accordingly.

Types of Benchmarks (Cont'd)

v) Synthetic Benchmarks

- Presents a mix of computations, system calls and I/O requests.

- Artificial programs that mimic a real program execution environment by using statistical data about real high-level language programs.

- Can be developed quickly and do not rely on real data.

- Some examples of the relevant statistical data are:
    - Fraction of statements of each type (assignment, conditionals, for-loops, etc.)
    - Fraction of variables of each type (integer, real, character, etc.) and locality (local or global)
    - Fraction of expressions with certain number and type of operators and operands

Types of Benchmarks (Cont'd)

- Procedure for programming synthetic benchmark:
  - take statistics of all types of operations from many application programs
  - get proportion of each operation
  - write program based on the proportion above

- They are too small to make statistically relevant memory or disk references.

vi) Application Benchmarks (or Real programs)

- A small subset of real-world programs that are representative of the application domain of interest.
- e.g. word processing software, tool software of CAD, user's application software (i.e. MIS)

Types of Benchmarks (Cont'd)

vii) Database benchmarks

- To measure the throughput and response times of database management systems (DBMS)

viii) Parallel benchmarks

- Used on machines with multiple cores, processors or systems consisting of multiple machines.

Application vs Synthetic benchmarks

Main advantage of synthetic benchmark over application one:

- overall application domain characteristics can be closely matched by a single program.
- Application benchmarks *give much better measure of real-world performance* on a given system,
- Synthetic benchmarks are *useful for testing individual components*, like a hard disk or networking device.

Application vs Synthetic benchmarks (Cont'd)

Since a synthetic benchmark not designed to do anything meaningful, may show following *strange characteristics*:

1) Since the expressions controlling conditional statements are chosen randomly,

- a significant amount of unreachable (or *dead*) code may be created, much of which can be eliminated by a *smart* compiler.
- Code-size becomes highly dependent on quality of compiler.

2) Random selection of statements may result in *unusual locality properties*, defeating well-designed paging algorithms.

3) The benchmark may be *small enough* to fit entirely in the cache and result in an unusually good performance.

Some Popular Benchmarks

i) Whetstone synthetic benchmark

- Based upon the characteristics of FORTRAN programs doing extensive floating-point computation.
- A single *Whetstone instruction* is defined as a mix of basic high-level operations, and the results are reported as *mega-Whetstones per second*.
- This benchmark, although still very popular represents outdated and arbitrary instruction mixes and thus
  - not very useful for machines with a high degree of internal parallelism (pipelining, vectored computation etc.),
  - particularly in conjunction with optimizing and parallelizing compilers.

Some Popular Benchmarks (Cont'd)

ii) Dhrystone synthetic benchmark

- written in C,
- designed to represent applications involving primarily integer arithmetic and string manipulation in a block-structured language.
- This benchmark is only 100 statements long, and thus would fit in the instruction cache of most systems.
- Also, since calls only two simple procedures, *strcpy()* and *strcmp()*, a compiler that replaces them by inline code can cause a very significant speed-up.
- The results are quoted as *Dhrystones/sec*, i.e. the rate at which the individual benchmark statements are executed.

Some Popular Benchmarks (Cont'd)

iii) Linpack application benchmark

- Solves a dense 100 x 100 linear system of equations using the Linpack library package.

- Two problems with this benchmark:
  - Over 80% of the time is spent in doing **A(I) =B(I) +C x D(I)** type of calculation, making results highly dependent on how well such a computation is handled,
  - Problem is too small.

Precautions in Benchmarking

A number of *hardware and software factors* need to be considered before running benchmarks.

1) Many benchmarks place *significant stress* on specific components,

- ensure all such components are in *good working order*,
- *properly cooled* (if necessary), and *receiving adequate power*.

2) When comparing machines using benchmarks, some care is needed to eliminate the effect of *extraneous factors*.

- e.g., a highly optimizing compiler can often speedup the code twicely compared with an ordinary compiler.
- Therefore, it's necessary that benchmarks for all machines be compiled using *compilers with similar features and settings*.

3) The details of machine configuration should be spelled out clearly, e.g., one could not claim much shorter times by just increasing the cache-size.

Precautions in Benchmarking (Cont'd)

4) Parameters for OS, applications and drivers must be satisfied to ensure accurate, repeatable benchmark results.

- Windows (and other) OSs proactively prefetch data and store numerous temporary files that could interfere with a benchmark, so it's best to clear out any temporary files and prefetch data before running a test.
- In Windows 7, you can find prefetch data in *C:\Windows\Prefetch*, and temporary files in *C:\Windows\Temp* and *C:\Users\[username]\AppData\Local\Temp*.

5) Using the correct (or latest) drivers for a component to ensure it is operating and performing optimally.

- especially true of graphics boards and motherboards /chipsets, where wrong driver can significantly worsen the system's frame rates or transfer speeds and latency.
- Finally, confirm that the OS is fully updated and patched to ensure optimal compatibility and to reflect the current, real-world OS configuration.

List of Challenges

Interpretation of benchmarking data is extraordinarily difficult.

1) Manufacturers commonly report only those benchmarks (or aspects of benchmarks) that show their products in the best light.

2) Many benchmarks focus entirely on the speed of computational performance, neglecting other important features such as *Qualities of Service*.

- Transaction Processing Performance Council (TPPC) Benchmark specifications partially address these concerns by specifying *ACID property tests*, *database scalability rules* and *service level requirements*.

List of Challenges (Cont'd)

3) Users can have very different perceptions of performance than benchmarks may suggest.

- In particular, users appreciate *predictability* — servers that always meet or exceed service level agreements.
- Benchmarks tend to emphasize mean scores (IT perspective), rather than maximum worst-case response times (real-time computing perspective), or low standard deviations (user perspective).

4) Many server architectures degrade dramatically at high (near 100%) levels of usage.

- Vendors, in particular, tend to publish server benchmarks at continuous at about 80% usage — an unrealistic situation.

Benchmark Strategies

Most of the benchmark programs base the measure of performance on *time required* to execute benchmark. Several other strategies can be employed, however.

Specifically, the three different strategies for using a benchmark program are the following.

1) <u>Fixed Computation Benchmarks:</u>

Measure the time required to perform a *fixed amount of computation*.

- a fixed workload
- Example: The SPEC (Standard Performance Evaluation Corporation) CPU benchmarks
- www.spec.org

Benchmark Strategies (Cont'd)

2) **<u>Fixed Time Benchmarks:</u>**

Measure the amount of computation performed within a *fixed period of time*.

- *Argument*: users with large problems are willing to wait a fixed amount of time to obtain a solution.
- At the end of allotted execution time, total amount of computation completed, used as the measure of relative speeds of different systems.
- Example: The TPC (Transaction Processing perf. Council) benchmarks.
- www.tpc.org

Benchmark Strategies (Cont'd)

3) Variable-Computation & Variable-Time Benchmarks:

Allow both the amount of computation ~~performed~~ and the time to vary.

- ~~These types of benchmarks instead try to~~ measure some ~~other~~ aspect of performance that is a function of the computation and the execution time.
- Example: The HINT (Hierarchical INTegration) benchmark.
- HINT uses a performance measure called QUIPS (QUality Improvements Per Second).