COSC 6385
Computer Architecture

Performance Measurement

Edgar Gabriel
Fall 2008

Measuring performance (1)

- Response time: how long does it take to execute a certain application/a certain amount of work
- Given two platforms \( X \) and \( Y \), \( X \) is \( n \) times faster than \( Y \) for a certain application if
  \[
  n = \frac{\text{Time}_Y}{\text{Time}_X}
  \tag{1}
  \]
- Performance of \( X \) is \( n \) times faster than performance of \( Y \) if
  \[
  n = \frac{\text{Time}_Y}{\text{Time}_X} = \frac{1}{\text{Perf}_Y} = \frac{\text{Perf}_X}{\text{Perf}_Y}
  \tag{2}
  \]
Measuring performance (II)

• Timing how long an application takes
  - **Wall clock time/elapsed time:** time to complete a task as seen by the user. Might include operating system overhead or potentially interfering other applications.
  - **CPU time:** does not include time slices introduced by external sources (e.g. running other applications). CPU time can be further
    • **User CPU time:** CPU time spent in the program
    • **System CPU time:** CPU time spent in the OS performing tasks requested by the program.

Measuring performance

• E.g. using the UNIX `time` command

![Example of using the UNIX time command](image)
Amdahl’s Law

- Describes the performance gains by enhancing one part of the overall system (code, computer)

\[
\text{Speedup} = \frac{\text{Time}_{\text{org}}}{\text{Time}_{\text{enh}}} = \frac{\text{Perf}_{\text{enh}}}{\text{Perf}_{\text{org}}}
\]  

(3)

- Amdahl’s Law depends on two factors:
  - Fraction of the execution time affected by enhancement
  - The improvement gained by the enhancement for this fraction

\[
\text{Time}_{\text{enh}} = \text{Time}_{\text{org}}(1 - \text{Fraction}_{\text{enh}}) + \frac{\text{Fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}
\]

(4)

\[
\text{Speedup}_{\text{overall}} = \frac{\text{Time}_{\text{org}}}{\text{Time}_{\text{enh}}} = \frac{1}{(1 - \text{Fraction}_{\text{enh}}) + \frac{\text{Fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}}
\]

(5)

Amdahl’s Law (III)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enh}}) + \frac{\text{Fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}}
\]

Fraction enhanced: 20%
Fraction enhanced: 40%
Fraction enhanced: 60%
Fraction enhanced: 80%
Amdahl’s Law (IV)

Speedup according to Amdahl’s Law

Fraction \( \text{enh} = 0.4 \)

\( \text{Speedup}_{\text{enh}} = 10 \)

using formula (5)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enh}}) + \frac{\text{Fraction}_{\text{enh}}}{\text{Speedup}_{\text{enh}}}} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56
\]

Amdahl’s Law - example

- Assume a new web-server with a CPU being 10 times faster on computation than the previous web-server. I/O performance is not improved compared to the old machine. The web-server spends 40% of its time in computation and 60% in I/O. How much faster is the new machine overall?

Fraction \( \text{enh} = 0.4 \)

\( \text{Speedup}_{\text{enh}} = 10 \)

using formula (5)
Amdahl’s Law - example (II)

- Example: Consider a graphics card
  - 50% of its total execution time is spent in floating point operations
  - 20% of its total execution time is spent in floating point square root operations (FPSQR).

Option 1: improve the FPSQR operation by a factor of 10.
Option 2: improve all floating point operations by a factor of 1.6

\[
\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1-0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22
\]

\[
\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23 \quad \rightarrow \quad \text{Option 2 slightly faster}
\]

CPU Performance Equation

- Micro-processors are based on a clock running at a constant rate
- Clock cycle time: \( CC_t \)
  - length of the discrete time event in \( \text{ns} \)
- Equivalent measure: Rate \( CPU = \frac{1}{CC_{\text{time}}} \)
  - Expressed in MHz, GHz
- CPU time of a program can then be expressed as

\[
CPU_{\text{time}} = no_{\text{cycles}} \times CC_{\text{time}} \quad (6)
\]

or

\[
CPU_{\text{time}} = \frac{no_{\text{cycles}}}{CPU} \quad (7)
\]
CPU Performance equation (II)

- **CPI**: Average number of clock cycles per instruction
- **IR**: number of instructions
  \[
  CPI = \frac{no_{cycles}}{IR}
  \]  
  (8)

- Since the CPI is often available, the CPU time is
  \[
  CPU_{time} = IC \times CPI \times CC_{time}
  \]  
  (9)

- Expanding formula (6) leads to
  \[
  CPU_{time} = \frac{\text{instructions}}{\text{program}} \times \frac{no_{cycles}}{\text{instruction}} \times \frac{\text{time}}{no_{cycles}}
  \]  
  (10)

CPU performance equation (III)

- According to (7) CPU performance is depending on
  - Clock cycle time → Hardware technology
  - CPI → Organization and instruction set architecture
  - Instruction count → ISA and compiler technology
- Note: on the last slide we used the average CPI over all instructions occurring in an application
- Different instructions can have strongly varying CPI’s
  \[
  no_{cycles} = \sum_{i=1}^{n} IC_i \times CPI_i
  \]  
  (11)

  \[
  CPU_{time} = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times CC_{time}
  \]  
  (12)
CPU performance equation (IV)

• The average CPI for an application can then be calculated as

\[
CPI = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{IC_{total}} = \sum_{i=1}^{n} \frac{IC_i}{IC_{total}} \times CPI_i
\]  

(13)

\frac{IC_i}{IC_{total}} : \text{Fraction of occurrence of that instruction in a program}

• Note: \( CPI_i \) should be measured for every single application separately since it might vary due to pipelining, cache effects etc.

Example (I)

• (Page 43 in the 4th Edition)
  Consider a graphics card, with
  - FP operations (including FPSQR): frequency 25%, average CPI 4.0
  - FPSQR operations only: frequency 2%, average CPI 20
  - all other instructions: average CPI 1.333333
  • Design option 1: decrease CPI of FPSQR to 2
  • Design option 2: decrease CPI of all FP operations to 2.5
  Using formula (13):

\[
CPI_{1\text{arg}} = \sum_{i=1}^{n} \frac{IC_i}{IC_{total}} \times CPI_i = (4 \times 0.25) + (1.333333 \times 0.75) = 2.0
\]

\[
CPI_1 = CPI_{1\text{arg}} - \text{enh} = 2.0 - 0.02(20 - 2) = 1.64
\]

\[
CPI_2 = \sum_{i=1}^{n} \frac{IC_i}{IC_{total}} \times CPI_i = (2.5 \times 0.25) + (1.333333 \times 0.75) = 1.625
\]
Example (II)

- Slightly modified compared to the previous section: consider a graphics card, with
  - FP operations (excluding FPSQR): frequency 25%, average CPI 4.0
  - FPSQR operations: frequency 2%, average CPI 20
  - all other instructions: average CPI 1.33
- Design option 1: decrease CPI of FPSQR to 2
- Design option 2: decrease CPI of all FP operations to 2.5

Using formula (13):

\[
CPI_{avg} = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{IC_{total}} = (4 \times 0.25) + (20 \times 0.02) + (1.33 \times 0.73) = 2.3709
\]

\[
CPI_1 = \sum_{i=1}^{n} \frac{IC_i}{IC_{total}} \times CPI_i = (4 \times 0.25) + (2 \times 0.02) + (1.33 \times 0.73) = 2.0109
\]

\[
CPI_2 = \sum_{i=1}^{n} \frac{IC_i}{IC_{total}} \times CPI_i = (2.5 \times 0.25) + (20 \times 0.02) + (1.33 \times 0.73) = 1.9959
\]

Dependability

- Module reliability measures
  - MTTF: mean time to failure
  - FIT: failures in time

\[
FIT = \frac{1}{MTTF}
\]  \hspace{1cm} (14)

  - Often expressed as failures in 1,000,000,000 hours
- MTTR: mean time to repair
- MTBF: mean time between failures

\[
MTBF = MTTF + MTTR
\]  \hspace{1cm} (15)

- Module availability:

\[
M_A = \frac{MTTF}{MTTF + MTTR}
\]  \hspace{1cm} (16)
Dependability - example

- Assume a disk subsystem with the following components and MTTFs:
  - 10 disks, MTTF=1,000,000h
  - 1 SCSI controller, MTTF=500,000h
  - 1 power supply, MTTF=200,000h
  - 1 fan, MTTF= 200,000h
  - 1 SCSI cable, MTTF=1,000,000h
- What is the MTTF of the entire system?
- What is the probability, that the system fails within a 1 week period?

Dependability - example (II)

- Determine the sum of the failures in time of all components

\[
FIT_{system} = \frac{1}{1,000,000} + \frac{1}{500,000} + \frac{1}{200,000} + \frac{1}{200,000} + \frac{1}{1,000,000} = \frac{10 + 2 + 5 + 5 + 1}{1,000,000} = \frac{23}{1,000,000} = \frac{23,000}{1,000,000,000}
\]

\[
MTTF_{system} = \frac{1}{FIT_{system}} = 43,500h
\]

- Probability that the system fails within a 1 week period:

\[
P_{system} = \frac{24 * 7}{43,500} = 0.00386 = 0.386%
\]
### Dependability - example (III)

- What happens if we add a second power supply and we assume, that the MTTR of a power supply is 24 hours?
- Assumption: failures are not correlated
  - MTTF of the pair of power supplies is the mean time to failure of the overall system divided by the probability, that the redundant unit fails before the primary unit has been replaced
  - MTTF of the overall system is $\frac{MTTF_{\text{power}}}{2}$
  - Probability, that 1 unit fails within MTTR: $\frac{MTTR}{MTTF_{\text{power}}}$

\[
MTTF_{\text{pair}} = \frac{MTTF_{\text{power}}}{2} \times \frac{MTTR}{MTTF_{\text{power}}} = \frac{MTTF_{\text{power}}^2}{2MTTR} = \frac{200,000^2}{2 \times 24} = 830,000,000
\]

### From 1st quiz last year

- In order to minimize the MTTF of their mission critical computers, each program on the space shuttle is executed by two computers simultaneously. Computer A is from manufacturer X and has a MTTF of 40,000 hours, while computer B is from manufacturer Y and has a different MTTF. The overall MTTF of the system is 4,000,000 hours.
  - How large is the probability that the entire system (i.e. both computers) fails during a 400 hour mission?
  - What MTTF does the second/backup computer have if the MTTF of the overall system is 4,000,000 hours, assuming that Computer A is failing right at the beginning of the 400 hour mission and the MTTR is 400h (i.e. the system can only be repaired after landing)?

- Solution will be posted on the web, but try first on your own!
Choosing the right programs to test a system

- Most systems host a wide variety of different applications
- Profiles of certain systems given by their purpose/function
  - Web server:
    - High I/O requirements
    - Hardly any floating point operations
  - A system used for weather forecasting simulations
    - Very high floating point performance required
    - Lots of main memory
    - Number of processors have to match the problem size calculated in order to deliver at least real-time results

Choosing the right programs to test a system (II)

- Real application: use the target application for the machine in order to evaluate its performance
  - Best solution if application available
- Modified applications: real application has been modified in order to measure a certain feature.
  - E.g. remove I/O parts of an application in order to focus on the CPU performance
- Application kernels: focus on the most time-consuming parts of an application
  - E.g. extract the matrix-vector multiply of an application, since this uses 80% of the user CPU time.
Choosing the right programs to test a system (III)

- Toy benchmarks: very small code segments which produce a predictable result
  - E.g. sieve of Eratosthenes, quicksort
- Synthetic benchmarks: try to match the average frequency of operations and operands for a certain program
  - Code does not do any useful work

SPEC Benchmarks

Slides based on a talk and courtesy of Matthias Mueller,
Center for Information Services and High Performance Computing (ZIH)
Technical University Dresden
What is SPEC?

• The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops suites of benchmarks and also reviews and publishes submitted results from our member organizations and other benchmark licensees.

• For more details see http://www.spec.org

SPEC Members


• SPEC Associates: California Institute of Technology * Center for Scientific Computing (CSC) * Defence Science and Technology Organisation - Stirling * Dresden University of Technology * Duke University * JAIST * Kyushu University * Leibniz Rechenzentrum - Germany * National University of Singapore * New South Wales Department of Education and Training * Purdue University * Queen's University * Righthmark * Stanford University * Technical University of Darmstadt * Texas A&M University * Tsinghua University * University of Aizu - Japan * University of California - Berkeley * University of Central Florida * University of Illinois - NCSA * University of Maryland * University of Modena * University of Nebraska, Lincoln * University of New Mexico * University of Pavia * University of Stuttgart * University of Texas at Austin * University of Texas at El Paso * University of Tsukuba * University of Waterloo * VA Austin Automation Center *
SPEC groups

- Open Systems Group (desktop systems, high-end workstations and servers)
  - CPU (CPU benchmarks)
  - JAVA (java client and server side benchmarks)
  - MAIL (mail server benchmarks)
  - SFS (file server benchmarks)
  - WEB (web server benchmarks)
- High Performance Group (HPC systems)
  - OMP (OpenMP benchmark)
  - HPC (HPC application benchmark)
  - MPI (MPI application benchmark)
- Graphics Performance Groups (Graphics)
  - Apc (Graphics application benchmarks)
  - Opc (OpenGL performance benchmarks)

Why do we need benchmarks?

- Identify problems: measure machine properties
- Time evolution: verify that we make progress
- Coverage: help vendors to have representative codes
  - Increase competition by transparency
  - Drive future development (see SPEC CPU2000)
- Relevance: help customers to choose the right computer
SPEC-Benchmarks (CPU)

- 1989: first version SPEC9 with SPECInt89 and SPECfp89
  - contained 10 programs with a total of 150,000 lines of C- and Fortran code
- 1992: second version SPEC92, containing 11 additional programs. One of them was withdrawn later on because it became deprecated by compiler optimizations
  - normalized results obtained by calculating the geometric average of all SPEC-ratios. This leads to the SPECmark
  - Results are split since SPEC92 generally into SPECInt92 and SPECfp92
- Reference architectures:
  - SPEC9 und SPEC92: VAX 11/780
  - SPEC 2000: SUN Microsystems Ultra 5/10 Workstation (300 MHz UltraSPARC III processor, L1-Cache 16KB + 16KB on chip, L2-Cache 2MB off chip, 256 MB main memory)

SPEC-Benchmarks

- All SPEC benchmarks are publicly available and well known/understood
  - Compiler can introduce special optimizations for these benchmarks, which might be irrelevant for other, real-world applications.
    → user has to provide the precise list of compile-flags
    → user has to provide performance of base (non-optimized) run
  - Compiler can use statistical informations collected during the first execution in order to optimize further runs (Cache hit rates, usage of registers)
- Benchmarks designed such that external influences are kept at a minimum (e.g. input/output)
SPEC CPU2000

- 26 independent programs:
  - CINT2000 - integer benchmark: 12 applications (11 in C and 1 in C++)
  - CFP2000 - floating-point benchmark: 14 applications (6 in Fortran-77, 4 in Fortran-90 and 4 in C)
- Additional information is available for each benchmark:
  - Author of the benchmark
  - Detailed description
  - Dokumentation regarding Input and Output
  - Potential problems and references.

CINT2000

CINT2000 (Integer Component of SPEC CPU2000):

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Category</th>
<th>Full Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>C</td>
<td>Compression</td>
<td>HTML Text</td>
</tr>
<tr>
<td>vpr</td>
<td>C</td>
<td>FPGA Circuit Placement and Routing</td>
<td>HTML Text</td>
</tr>
<tr>
<td>gcc</td>
<td>C</td>
<td>C Programming Language Compiler</td>
<td>HTML Text</td>
</tr>
<tr>
<td>mcf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
<td>HTML Text</td>
</tr>
<tr>
<td>crafty</td>
<td>C</td>
<td>Game Playing Chess</td>
<td>HTML Text</td>
</tr>
<tr>
<td>parser</td>
<td>C</td>
<td>Word Processing</td>
<td>HTML Text</td>
</tr>
<tr>
<td>eon</td>
<td>C++</td>
<td>Computer Visualization</td>
<td>HTML Text</td>
</tr>
<tr>
<td>passwd</td>
<td>C</td>
<td>PERL Programming Language</td>
<td>HTML Text</td>
</tr>
<tr>
<td>group</td>
<td>C</td>
<td>Group Theory, Interpreter</td>
<td>HTML Text</td>
</tr>
<tr>
<td>vortex</td>
<td>C</td>
<td>Object-oriented Database</td>
<td>HTML Text</td>
</tr>
<tr>
<td>bzip2</td>
<td>C</td>
<td>Compression</td>
<td>HTML Text</td>
</tr>
<tr>
<td>twolf</td>
<td>C</td>
<td>Place and Route Simulator</td>
<td>HTML Text</td>
</tr>
</tbody>
</table>
CFP2000 (Floating Point Component of SPEC CPU2000):

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language Category</th>
<th>Full Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>168_xwpsi</td>
<td>Fortran 77</td>
<td>Physics / Quantum Chromodynamics</td>
</tr>
<tr>
<td>171_swim</td>
<td>Fortran 77</td>
<td>Shallow Water Modeling</td>
</tr>
<tr>
<td>172_ingrid</td>
<td>Fortran 77</td>
<td>Multi-grid Solver, 3D Potential Field</td>
</tr>
<tr>
<td>173_apflu</td>
<td>Fortran 77</td>
<td>Parabolic / Elliptic Partial Differential Equations</td>
</tr>
<tr>
<td>177_mesa</td>
<td>C</td>
<td>3-D Graphics Library</td>
</tr>
<tr>
<td>178_gadget</td>
<td>Fortran 90</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>179_art</td>
<td>C</td>
<td>Image Recognition / Neural Networks</td>
</tr>
<tr>
<td>183_equake</td>
<td>C</td>
<td>Seismic Wave Propagation Simulation</td>
</tr>
<tr>
<td>187_facenc</td>
<td>Fortran 90</td>
<td>Image Processing Face Recognition</td>
</tr>
<tr>
<td>188_anmp</td>
<td>C</td>
<td>Computational Chemistry</td>
</tr>
<tr>
<td>189_lucas</td>
<td>Fortran 90</td>
<td>Number Theory / Primality Testing</td>
</tr>
<tr>
<td>191_tma3d</td>
<td>Fortran 90</td>
<td>Finite-element Crash Simulation</td>
</tr>
<tr>
<td>200_sixtrack</td>
<td>Fortran 77</td>
<td>High Energy Nuclear Physics Accelerator Design</td>
</tr>
<tr>
<td>301_apsi</td>
<td>Fortran 77</td>
<td>Meteorology: Pollutant Distribution</td>
</tr>
</tbody>
</table>

Example for a CINT benchmark

256.bzip2
SPEC CPU2000 Benchmark Description File

Benchmark Name
256.bzip2

Benchmark Author
Julian Seward <jsward@pmi.org>
Performance metrics

- Two fundamentally different metrics:
  - speed
  - rate (throughput)
- For each metric results for two different optimization level have to be provided
  - moderate optimization
  - aggressive optimization
  \[ \Rightarrow 4 \text{ results for } \text{CINT2000} + 4 \text{ results for } \text{CFP2000} = 8 \text{ metrics} \]
- If taking the measurements of each application individually into account:
  \[ \Rightarrow 2^2 \times (14 + 12) = 104 \text{ metrics} \]
Performance metrics (II)

- All results are relative to a reference system
- The final results is computed by using the geometric mean values

\[
\text{Speed: } \frac{\text{SPEC}_{\text{new,fp}}}{\text{SPEC}_{\text{old,fp}}} = \sqrt[n]{\frac{t_{\text{ref}}}{t_{\text{run}}}} \times 100
\]

\[
\text{Rate: } \text{SPEC}_{\text{new,fp}} = \sqrt[n]{\frac{t_{\text{ref}}}{t_{\text{run}}} \times 1.16 \times N}
\]

with:
- \( n \): number of benchmarks in a suite
- \( t_{\text{ref}} / t_{\text{run}} \): execution time for benchmark \( i \) on the reference/test system
- \( N \): Number of simultaneous tasks

---

Single processor performance according to SPEC CPU2000

![Single processor performance according to SPEC CPU2000](image)

---

COSC 6385 – Computer Architecture
Edgar Gabriel

Slides based on a talk and courtesy of
Matthias Mueller,
Center for Information Services and High Performance Computing
Technical University Dresden

---

COSC 6385 – Computer Architecture
Edgar Gabriel

Slides based on a talk and courtesy of
Matthias Mueller,
Center for Information Services and High Performance Computing
Technical University Dresden
Reporting results

- SPEC produces a minimal set of representative numbers:
  - Reduces complexity to understand correlations
  - Eases comparison of different systems
  - Loss of information

- Results have to be compliant to the SPEC benchmarking rules in order to be approved as an official SPEC report:
  - All components have to available at least 3 months after publication (including a runtime environment for C/C++/Fortran applications)
  - Usage of SPEC tools for compiling and reporting
  - Each individual benchmark has to be executed at least three times
  - Verification of the benchmark output
  - A maximum of four optimization flags are allowed for the base run (including preprocessor and link directives)
  - Disclosure report containing all relevant data has to be available