ECE 486/586
Computer Architecture

Prof. Mark G. Faust

Maseeh College of Engineering
and Computer Science

PORTLAND STATE
UNIVERSITY

Outline

- Measuring, reporting, and summarizing performance
- Benchmarks
- SPECRatio and geometric means
- Quantitative principals of computer design
  - Execution time equation
  - Amdahl's law

Which is Faster?

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Time to run the task
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Performance)
  - Throughput, bandwidth
- Latency
  - Concorde 1x/day, 747 12x/day)

Measuring and Reporting Performance

- User's Perspective
  - A computer is faster when a program/task runs in less time
  - Response time, latency, execution time
- IT Center Manager's Perspective
  - A computer is faster when it completes more programs/tasks in a unit of time
  - Throughput

Measuring and Reporting Performance

- "Computer X is n times faster than computer Y" means:

$$\text{Speedup (n)} = \frac{\text{Execution time of Y}}{\text{Execution time of X}}$$

$$\begin{aligned}\%\ \text{Improvement} &= \frac{\text{Execution time of Y - Execution time of X}}{\text{Execution time of Y}} \\ &= \frac{\text{Execution time of Y}}{\text{Execution time of Y}} - \frac{\text{Execution time of X}}{\text{Execution time of Y}} \\ &= 1 - \frac{1}{\text{Speedup}}\end{aligned}$$

Measuring and Reporting Performance

- We might say “The throughput of server X is n times higher than server Y”:

$$n = \frac{\text{Execution time of Y}}{\text{Execution time of X}} = \frac{\text{Tasks Executed per Second by X}}{\text{Tasks Executed per Second by Y}}$$

Measuring and Reporting Performance

- The most meaningful way to measure response time is by *elapsed time* or *wall clock time* which accounts for
  - Disk accesses
  - Memory accesses
  - I/O activities
  - Other users (unless system is unloaded)
  - CPU time
    - User time
    - OS time (on behalf of user)

Measuring and Reporting Performance

- Unix **time** command
  - 90.7u 12.9s 2:39 65%
    - User CPU time in seconds
    - System CPU time
    - Elapsed Time
    - Percentage of Elapsed Time which is CPU Time
    - (90.7 + 12.9)/159 = 65%
    - 35% on I/O or running other programs
- Issues
  - Accuracy of OS self-reporting CPU time
  - User CPU or (System + User CPU)
    - Partitioning of processing may vary with OS → use total
  - System Performance – Elapsed time on unloaded system
  - CPU Performance – User CPU time on unloaded system

Evaluating Performance

- Simulate a workload which we hope to be representative and therefore predictive of actual behavior for our application
  - Real applications
  - Modified (or scripted) applications
  - Kernels
  - Toy benchmarks
  - Synthetic benchmarks

Benchmark Suites

- SPEC
  - Standard Performance Evaluation Corporation
  - www.spec.org
  - Benchmarks for wide variety of applications classes
  - Tend to be real programs
- Desktop benchmarks
  - CPU intensive
    - SPEC CPU2000
      - 12 integer benchmarks: CINT2000
      - 14 floating point benchmarks: CFP2000
  - Graphics intensive
    - SPECviewperf
      - 3D rendering
    - SPECapc
      - Large applications

Benchmark Suites

- Server benchmarks
  - Variety in application and therefore benchmark
  - I/O activity (network, file I/O, etc)
  - SPEC
    - File Server benchmark: SPECSFS
      - Throughput-oriented benchmark for NFS (Network File System) performance
      - Response time requirements
      - Script of file server requests
      - I/O subsystem (network, disk) as well as processor
    - Web Server benchmark: SPECWeb
      - Web server benchmark simulating multiple clients
      - Static and dynamic page requests, posting data
    - Java-based web applications: SPECjbb
    - Virtualized data centers: SPECvirt_Sc2010

Benchmark Suites

- Server benchmarks
  - TPC (Transaction Processing Council www.tpc.org)
  - Measure ability of system to handle transactions
    - Database accesses, updates
    - Airline reservation, ATM systems
    - Complex query environment: TPC-C
    - Ad hoc decision support TCP-H
    - On-line transaction processing TPC-E
      - Simulates brokerage customer accounts
    - Measure is “transactions/second”

Benchmark Suites

- Embedded benchmarks
  - Difficulties
    - In earlier stage than PC and server industries in employing benchmarks
    - Extremely diverse application area
      - Simple appliance control to high performance networking switch
    - May execute only one program for entire product life
  - EEMBC
    - EDN Embedded Microprocessor Benchmark Consortium
    - Five Classes
      - Automotive/industrial
      - Consumer
      - Networking
      - Office automation
      - Telecommunications

Comparing and Summarizing Performance

Execution times of two programs on three systems

(Time in secs)	Computer A	Computer B	Computer C
Program P1	1	10	20
Program P2	1000	100	20
Total time	1001	110	40

Conclude
A is 10x faster than B for program P1
B is 10x faster than A for program P2
A is 20x faster than C for program P1
C is 50x faster than A for program P2
B is 2x faster than C for program P1
C is 5x faster than B for program P2

Comparing and Summarizing Performance

- Total Execution Time:
  A Consistent Summary Measure

(Time in secs)	Computer A	Computer B	Computer C
Program P1	1	10	20
Program P2	1000	100	20
Total time	1001	110	40

Conclude
B is 9.1x faster than A for programs P1 and P2
C is 25x faster than A for programs P1 and P2
C is 2.75x faster than B for programs P1 and P2

Comparing and Summarizing Performance

- Arithmetic mean (Assumes each program executed once)

$$\frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

- Weighted execution time

$$\sum_{i=1}^{n}\text{Weight}_i \times \text{Time}_i$$

Comparing and Summarizing Performance

- Weighting and Arithmetic Means

	Computers			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1 (secs)	1.00	10.00	20.00	0.50	0.909	0.999
Program P2 (secs)	1000.00	100.00	20.00	0.50	0.091	0.001
Arithmetic mean: W(1)	500.50	55.00	20.00			
Arithmetic mean: W(2)	91.91	18.19	20.00			
Arithmetic mean: W(3)	2.00	10.09	20.00			

Normalizing Execution Times

- Normalize execution times to a reference system and then average the normalized times
- Arithmetic Mean
  - Skewed by size of input
    - Gaming the benchmark by making input large on benchmark with best differential performance
  - Dependent upon which system is chosen for reference

Normalized Execution Times

- Arithmetic means useless for normalized times
- Relative geometric means consistent regardless of choice of reference system for normalization

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arithmetic mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

$$\frac{1.0}{0.63} = \frac{1.58}{1.0}$$

Geometric Mean: An Example

$$\sqrt[n]{\prod_{i=1}^{n} \text{Normalized Execution Time}_i}$$

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arithmetic mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

B: $\sqrt[2]{10.0 \times 0.1} = 1$

C: $\sqrt[2]{20.0 \times 0.02} = \sqrt[2]{0.4} \approx 0.63$

A: $\sqrt[2]{0.05 \times 50.0} = \sqrt[2]{2.5} \approx 1.58$

B: $\sqrt[2]{0.5 \times 5.0} = \sqrt[2]{2.5} \approx 1.58$

A: $\sqrt[2]{0.1 \times 10.0} = 1.0$

C: $\sqrt[2]{2.0 \times 0.2} = \sqrt[2]{0.4} \approx 0.63$

Summarizing Performance

- The best performance comparison method is the ratio of total execution times

$$\text{speedup} = \frac{\sum_{i=1}^{n} \text{Time}_{Xi}}{\sum_{i=1}^{n} \text{Time}_{Yi}}$$

SPECRatio

Benchmarks	Ultra 5 time (sec)	Opteron time (sec)	SPECRatio	Itanium 2 time (sec)	SPECRatio	Opteron/Itanium times (sec)	Itanium/Opteron SPECRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			20.86		27.12	1.30	1.30

- SPECRatio measure of relative performance on SPEC benchmarks
- Choice of reference machine irrelevant
- Geometric mean of SPECRatios (1.30) is ratio of geometric means (27.12/20.86)

Summarizing Performance

- Harmonic mean

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}}$$

  - Useful when averaging rates (e.g. MIPS, MFLOPs)

Harmonic Mean: An Example

- During a car trip you travel
  - 30 miles at 60 MPH
  - 10 miles at 40 MPH
  - 25 miles at 30 MPH

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}}$$

- What was your "average" speed?

$$\frac{30+10+25 \text{ miles}}{\frac{30 \text{ miles}}{60 \text{ miles/hour}} + \frac{10 \text{ miles}}{40 \text{ miles/hour}} + \frac{25 \text{ miles}}{30 \text{ miles/hour}}} \approx \frac{65}{1.58} = 41 \text{ MPH}$$

Benchmarks

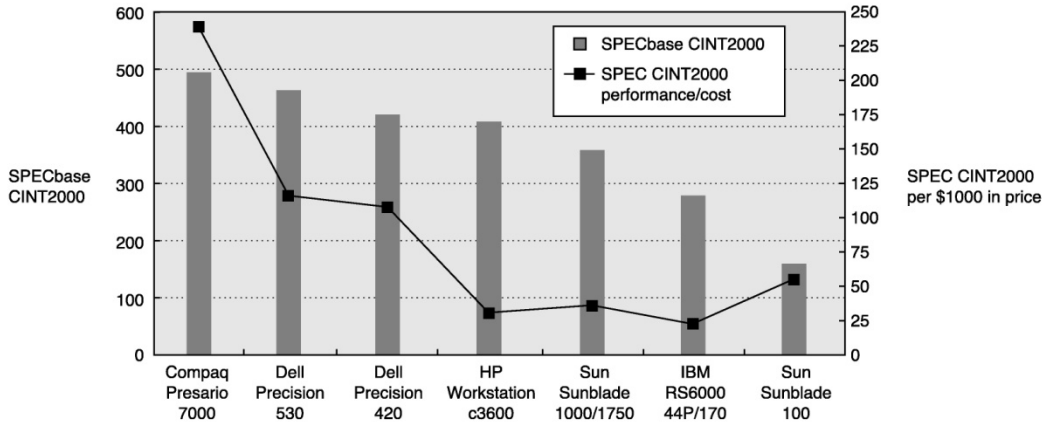
- Who uses benchmarks?
  - Consumers
  - IT Systems Designers and Buyers
    - PCs and Servers
  - Embedded Systems Designers
  - Computer Architects
- May want to quantify
  - "Raw" performance
    - Execution time, throughput, response time
  - Performance/$
  - Performance/watt
  - Performance/watt/$

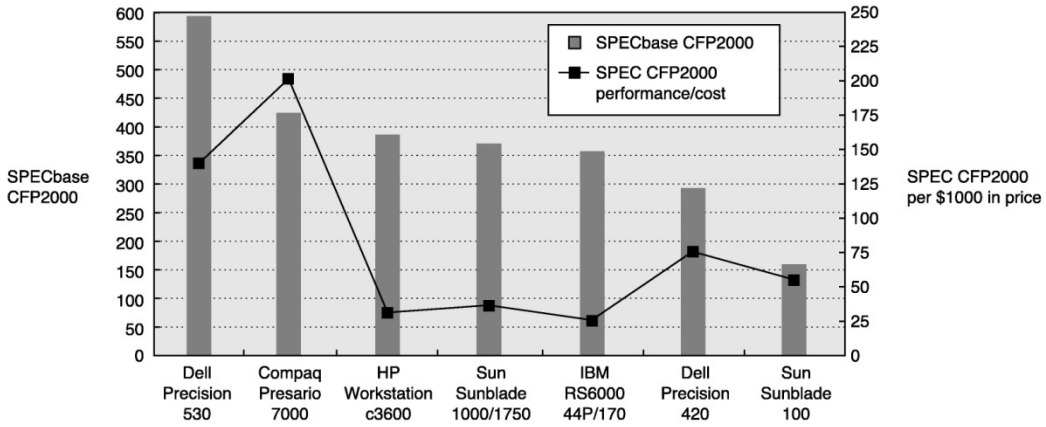
Purchasing decision

System design and purchasing decisions

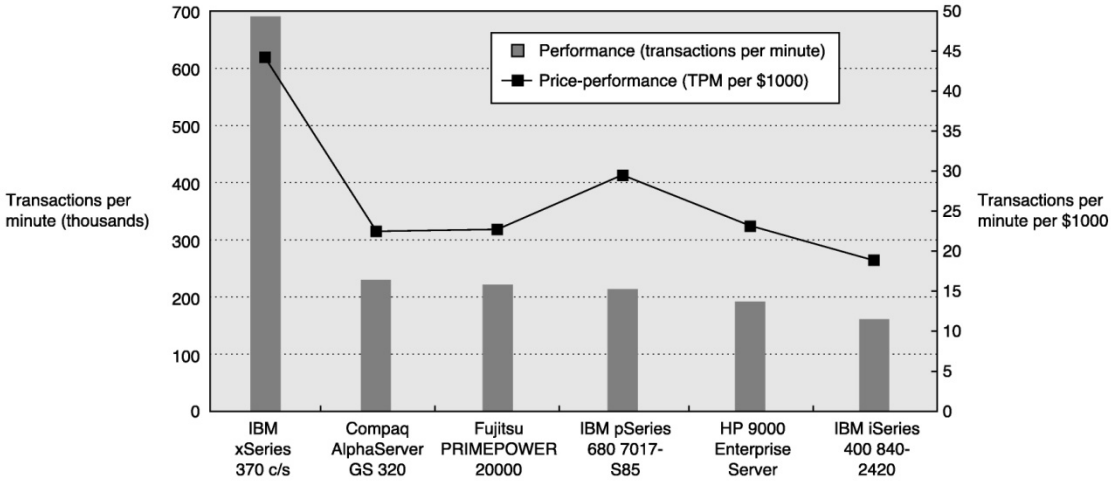
Design evaluation and decision making

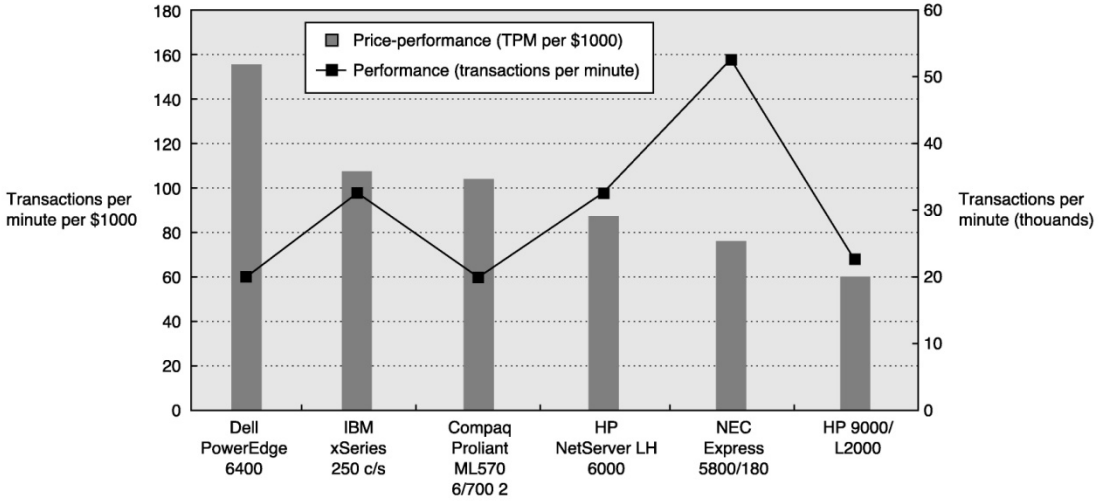
Desktop Benchmark Results



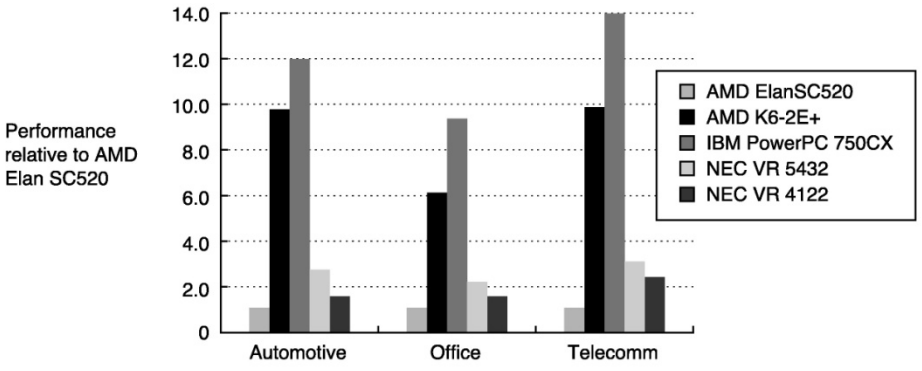


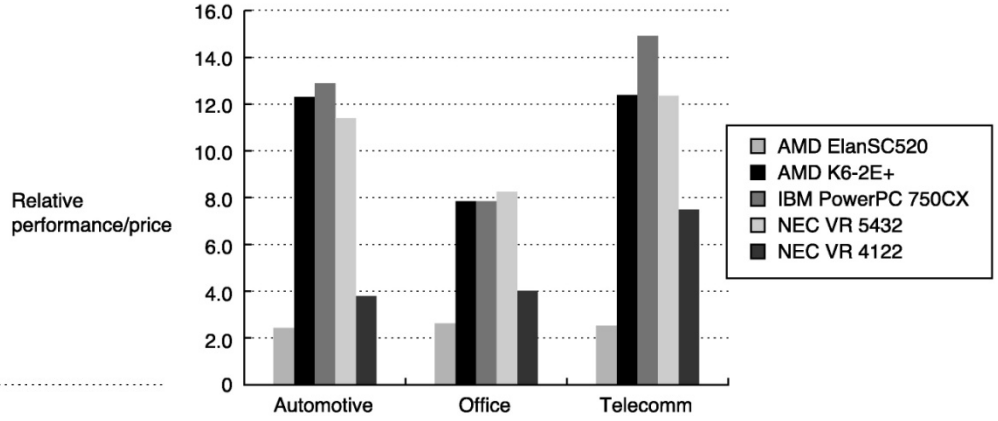
Transaction Server Benchmark Results

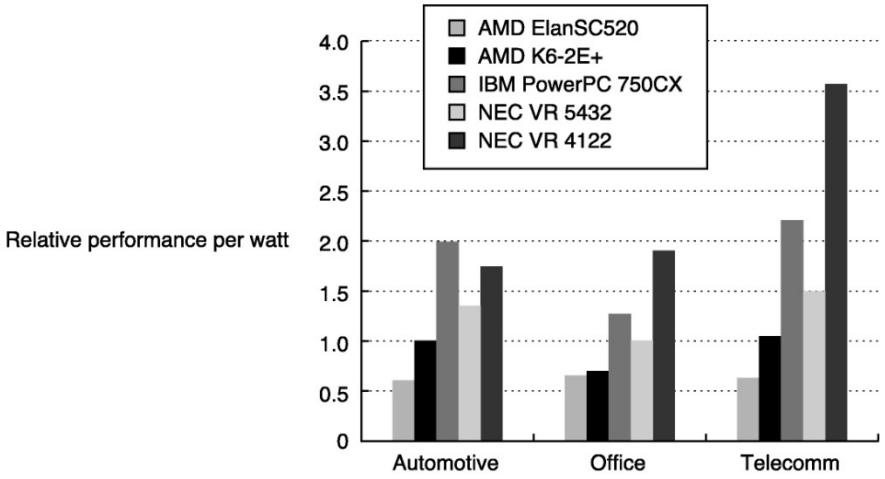




Embedded System Benchmark Results



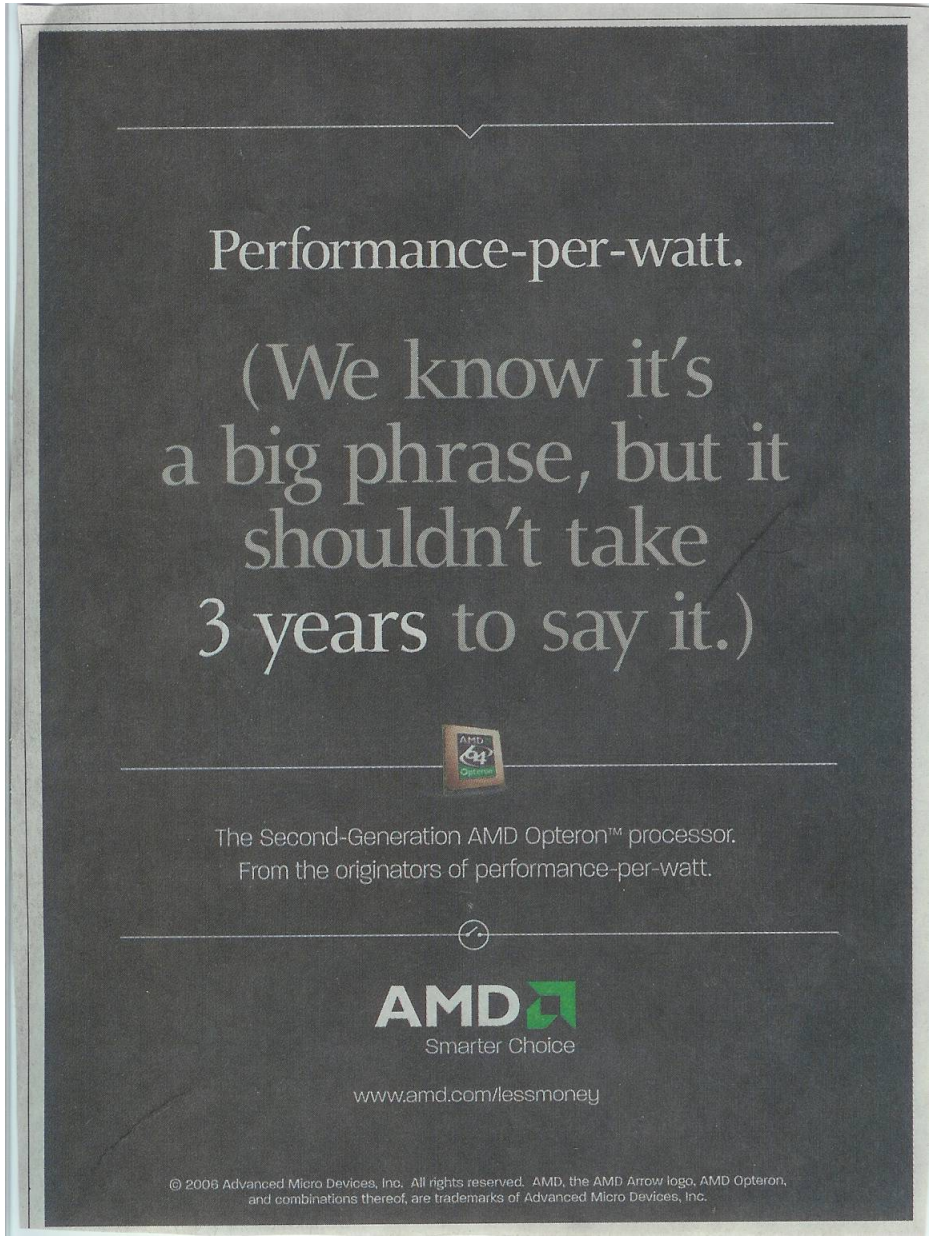


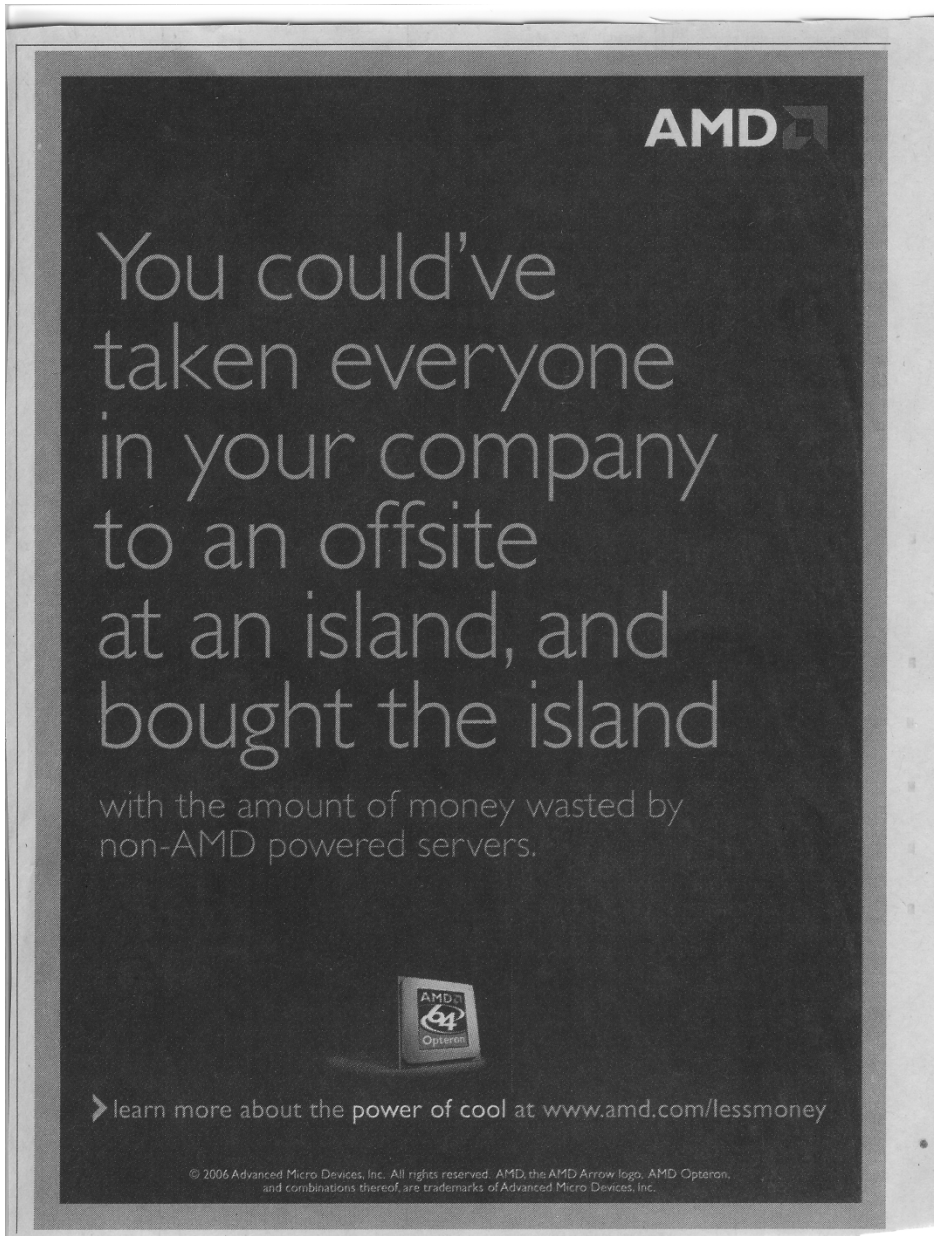


“Gaming” Benchmarks

- Source code modifications permitted?
- Benchmark switches permitted?
  - Optimize benchmark performance but might not ever be used in normal practice
- Input data set size/content fixed?
  - Use very large test data on those benchmark components where platform is known to perform best – skews averages

Performance vs. Performance-per-watt





Performance vs. Performance-per-watt

Power-Hungry Computers Put Data Centers in Bind

Newer Hardware Guzzles Electricity and Spews Heat, Requiring Costly Alterations

By Don Clark

The University at Buffalo installed a $2.3 million Dell Inc. supercomputer last summer, hoping to bolster its image as a research institution. Instead, the big machine came to symbolize an increasingly vexing problem for data centers world-wide.

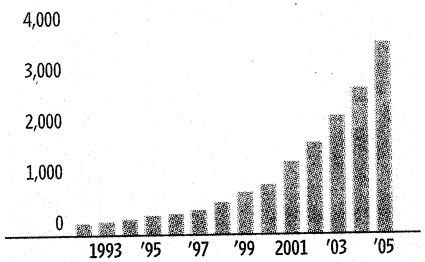
Once the machine was delivered, university officials discovered they had only enough electrical power to switch on two-thirds of the system. They have temporarily responded by throttling back use of an older supercomputer, but a $20,000 electrical-system upgrade will be needed to run both systems at full capacity.

"The calculations that were done fell slightly short," says Bruce Holm, a senior vice provost at the school, which is part of the State University of New York. "The bottom line was that they missed."

More such misses are likely. That's because, in its long-running race to boost performance, the computer industry has hit a major hurdle: The newest hardware—particularly the servers that run most business programs and Web sites—draws too much electricity and generates too much heat.

Energy Surge

Electricity required for low-end servers, in watts per square foot



Source: American Society of Heating Refrigerating and Air-Conditioning Engineers

The power-hungry machines, along with rising energy prices, are generating enormous utility bills and forcing changes on Silicon Valley technology suppliers that are akin to Detroit's struggle to improve gas mileage. (See related article on page B4.)

Though more-energy-efficient computers are on the way, it could be years before companies replace the systems they have already purchased.

In the meantime, bringing in more electricity and cooling is expensive and difficult in some data-center buildings. Organizations face unpleasant choices that include building new facilities, putting off server purchases or leaving costly space in computer rooms unoccupied to avoid overwhelming their air-conditioning systems.

Facilities planners at the University at Buffalo, for example, originally erred because they thought an older supercomputer would no longer be needed by the time their new machine arrived, Mr. Holm says. The need for both systems caused the university to consider spending as much as $150,000 to upgrade the current data center's air conditioning, just as the university was on the verge of moving the systems to a more modern computer room. "It's that kind of juggling act," Mr. Holm says.

If planners miscalculate, servers overheat, damaging circuitry or causing shutdowns that disrupt operations. The Uptime Institute, an organization that represents data-center managers, predicts

Please Turn to Page A16, Column 1

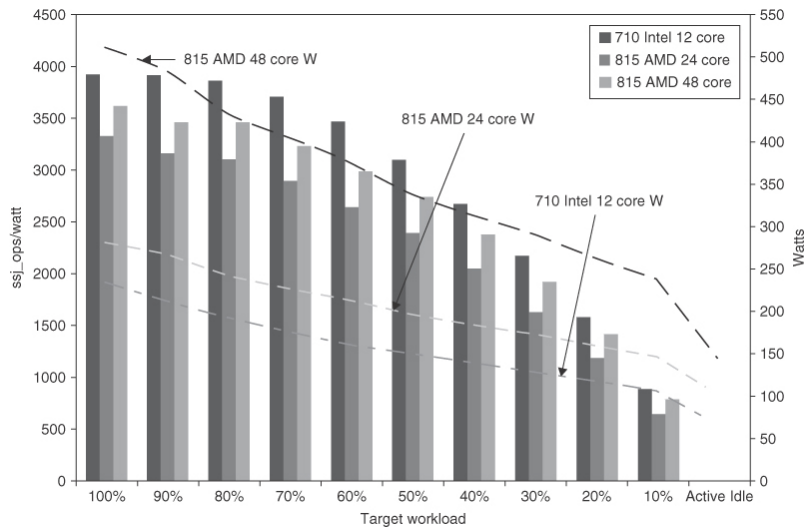
In Brief–

Power Consumption Rises For Servers, Study Shows

Power consumption of server systems doubled between 2000 and 2005, requiring the generating capacity of about 14 power plants world-wide, according to a new study. Jonathan Koomey, a staff scientist at Lawrence Berkeley National Laboratory and consulting professor at Stanford University, estimated that servers and associated equipment in computer rooms accounted for 1.2% of U.S. electricity consumption in 2005—an amount comparable to that used by color televisions—and 0.8% of world-wide consumption. The annual electricity bill was about $2.7 billion in the U.S. and $7.3 billion for the world, he wrote. The research was funded by **Advanced Micro Devices** Inc., which has used energy consumption as an issue in marketing its chips against those of rival **Intel**

Comparing performance

Base Server	PowerEdge R710	PowerEdge R815	PowerEdge R815
Processor	Xeon X5670	Opteron 6174	Opteron 6174
Clock rate	2.93 GHz	2.20 GHz	2.20 GHz
Total cores	12	24	48
Sockets	2	2	4
Price	$9352	$9286	$12,658
Max ssj_ops	910,978	926,676	1,840,450
Max ssj_ops/$	97	100	145



$$\frac{ssj\_operations/sec}{Watt} = \frac{ssj\_operations/sec}{Joules/sec} = \frac{ssj\_operations}{Joule}$$

$$Overall\ \frac{ssj\_operations}{watt} = \frac{\sum ssj\_operations}{\sum power}$$

Comparing performance

Base Server	PowerEdge R710	PowerEdge R815	PowerEdge R815
Processor	Xeon X5670	Opteron 6174	Opteron 6174
Clock rate	2.93 GHz	2.20 GHz	2.20 GHz
Total cores	12	24	48
Sockets	2	2	4
Price	$9352	$9286	$12,658
Max ssj_ops	910,978	926,676	1,840,450
Max ssj_ops/$	97	100	145
Ssj_ops/watt/$	324	254	213

CPU Execution Time

$$\text{CPU Time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

or

Program	Implementation	Process
Compiler	- Caches	technology
ISA	- Execution units	
	- Pipeline	

$$\text{CPU Time} = I \times CPI \times t_c$$

where

I = total number of instructions in a program

CPI = average number of clock cycles per instruction

t_c = processor's clock cycle time

CPI: Average Number of Clocks per Instruction

$$\text{CPU Time} = I \times \text{CPI} \times t_c$$

$$\text{CPU Time} = t_c \times \sum_{j}^{n} (\text{CPI}_j \times I_j)$$

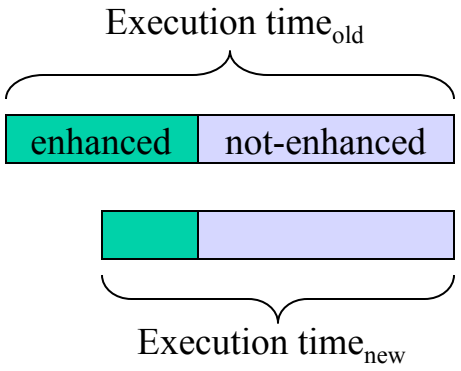
$$\text{CPI} = \sum_{j}^{n} \text{CPI}_j \times F_j \quad \text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

Instruction	Frequency	CPI_i	Cycles	% Time	% Time
ALU	50%	1	0.5	0.5/1.5	33%
Load	20%	2	0.4	0.4/1.5	27%
Store	10%	2	0.2	0.2/1.5	13%
Branch	20%	2	0.4	0.4/1.5	27%
Total			1.5		

Amdahl's Law

$$\text{Execution Time}_{new} = \text{Execution Time}_{old} \times \frac{\text{fraction}_{\text{not-enhanced}}}{\text{speedup}_{\text{not-enhanced}}}$$

$$+ \text{Execution Time}_{old} \times \frac{\text{fraction}_{enhanced}}{\text{speedup}_{enhanced}}$$



$$\text{Execution Time}_{new} = \text{Execution Time}_{old} \times \left[\frac{\text{fraction}_{\text{not-enhanced}}}{\text{speedup}_{\text{not-enhanced}}} + \frac{\text{fraction}_{enhanced}}{\text{speedup}_{enhanced}} \right]$$

$$\text{Execution Time}_{new} = \text{Execution Time}_{old} \times \left[(1 - \text{fraction}_{enhanced}) + \frac{\text{fraction}_{enhanced}}{\text{speedup}_{enhanced}} \right]$$

$$\text{Speedup}_{overall} = \frac{\text{Execution Time}_{old}}{\text{Execution Time}_{new}} = \frac{1}{\left[(1 - \text{fraction}_{enhanced}) + \frac{\text{fraction}_{enhanced}}{\text{speedup}_{enhanced}} \right]}$$

Amdahl's Law: An Example

- We are considering an enhancement to the processor of a web server. The new CPU is 10x faster on computation in the Web serving application than the original CPU. Assuming that the CPU is busy with computation 40% of the time and waiting for I/O 60% of the time, what is the overall speedup we can anticipate by incorporating the enhancement?

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}\right]}$$

$$\text{fraction}_{\text{enhanced}} = 0.4$$

$$\text{speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(0.6) + \frac{0.4}{10}\right]} = \frac{1}{0.64} \approx 1.56$$

Amdahl's Law

- That can't be right!?
- What if speedup is system-wide?

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}\right]}$$

$$\text{fraction}_{\text{enhanced}} = 1.0$$

$$\text{speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - 1) + \frac{1}{10}\right]} = \frac{1}{\left[\frac{1}{10}\right]} = 10$$

Amdahl's Law

- What if speedup is so great it eliminates completely the execution time formerly consumed by the fraction enhanced?

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}\right]}$$

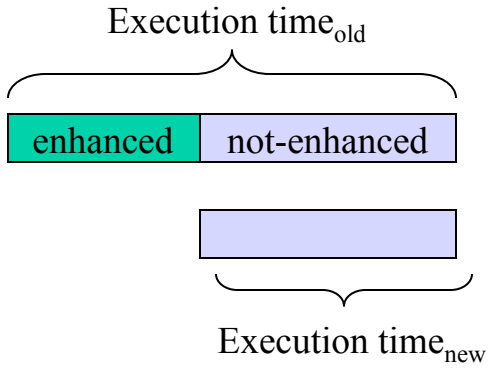
$\text{fraction}_{\text{enhanced}} = 0.5$

$\text{speedup}_{\text{enhanced}} = \infty$ (i.e. execution time of $\text{fraction}_{\text{enhanced}}$ is 0)

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - 0.5) + \frac{0.5}{\infty}\right]} = \frac{1}{\left[0.5 + \frac{0.5}{\infty}\right]} \approx \frac{1}{0.5} \approx 2$$

Amdahl's Law: Diminishing Returns

- Even an infinite speed up in a portion which eliminates its execution time contribution completely improves the overall execution time only in inverse proportion to its fractional share of the original execution time.



Amdahl's Law: An Example

- A common transformation required in graphics engines is square root. Implementations of floating point (FP) square root differ considerably in performance. Assume FPSQR is responsible for 20% of the execution time of a critical graphics application. One proposal is to enhance the FPSQR hardware to speed FPSQR by 10x. Another proposal is to make all floating point operations run faster by a factor of 1.6. Floating point instructions represent 50% of the overall execution time. Assuming the cost of each enhancement is the same which would you prefer?

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}\right]}$$

$$\text{Speedup}_{\text{FPSQR}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - 0.2) + \frac{0.2}{10}\right]} = \frac{1}{0.82} \approx 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{1}{\left[(1 - 0.5) + \frac{0.5}{1.6}\right]} = \frac{1}{0.8125} \approx 1.23$$

Using CPI: An Example

- Consider the same problem. Assume the following:
  - Frequency of FP operations (including FPSQR) = 25%
  - Average CPI of FP operations = 4.0
  - Average CPI of other instructions = 1.33
  - Frequency of FPSQR = 2%
  - CPI of FPSQR = 20
- Two design alternatives: decrease CPI of FPSQR to 2 or decrease average CPI of all FP operations to 2.5.

$$\text{CPI}_{\text{original}} = \sum_{j}^{n} (\text{CPI}_j \times \frac{\text{I}_j}{\text{Instruction Count}})$$
$$= (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

$$\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPQSR}} - \text{CPI}_{\text{new FPSQR}})$$
$$= 2.0 - 2\% \times (20 - 2.0) = 1.64$$

$$\text{CPI}_{\text{with new FP}} = \text{CPI}_{\text{original}} - 25\% \times (\text{CPI}_{\text{old FP}} - \text{CPI}_{\text{new FP}})$$
$$= 2.0 - 25\% \times (4.0 - 2.5) = 1.625$$

Some Principles

- Make the common case fast
  - Remember Pareto?
  - 90% of time spent in 10% of code
  - 90% of instructions are loads/stores (in reality ~ 40%)
    - Implications?
    - Optimize these instructions, employ caches, etc
  - 90% of jumps are to within 512 bytes of current location
    - Implications?
    - Use immediate mode address for jump?
  - Most loop branches are taken
    - Implications?
    - Branch prediction (fetch the branch target instead of next sequential instruction)

Some Principles

- Exploit Parallelism
  - System Level
    - Multiple processors
    - Multiple disks
    - Buses: pipelining, hidden arbitration
  - Processor Level
    - Overlapping instruction execution with pipelining
    - Multiple execution units
  - Logic/Gate Level
    - N-way set associative caches
    - Multiple memory banks
      - Hide pre-charge time
    - Carry lookahead adders

Some Principles

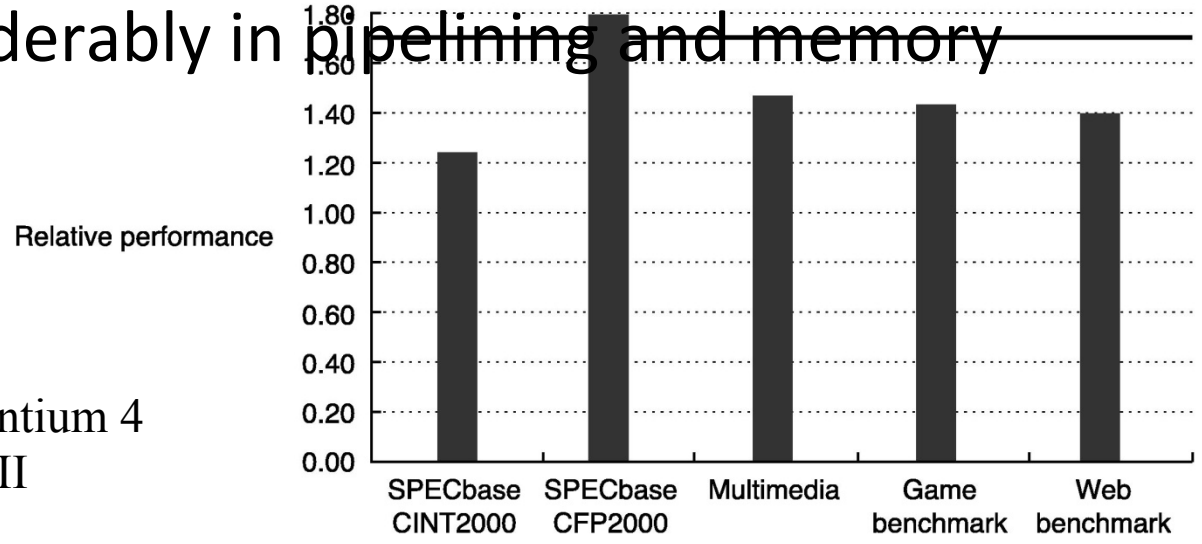
- Exploit Locality
  - Spatial
    - Memory near current reference is likely to be accessed also
    - Code and data
      - Sequential instruction execution
      - Sequential array elements, fields in a record/struct
  - Temporal
    - Recently accessed memory is likely to be referenced again
    - Code and data
      - Loops and functional calls
      - Repeated access to variable(s)

Fallacies and Pitfalls

- Fallacy
  - A falsehood – often widely believed to be true
- Pitfall
  - Potential error

Fallacy

- The relative performance of two processors with the same ISA can be judged by clock rate or by the performance of a single benchmark suite
  - Despite sharing the same ISA, two processors may differ considerably in pipelining and memory subsystems



Performance of 1.7GHz Pentium 4 relative to 1GHz Pentium III

Fallacy (not in text)

- The relative performance of two processors with the same ISA can be judged by CPI

gcc optimization	Relative Performance	Clock cycles (millions)	Instruction count (millions)	CPI
None	1.00	158,615	114,938	1.38
O1	2.37	66,990	37,470	1.79
O2	2.38	66,521	39,993	1.66
O3	2.41	65,747	44,993	1.46

Sort program on 100K words using different compiler optimization options on Pentium 4 with clock rate of 3.06 GHz, 533 MHz system bus, 2 GB PC2100 DDR SDRAM running Linux 2.4.20 [from CODI]

Fallacy

- Benchmarks remain valid indefinitely
- Why might validity/applicability of a benchmark change over time?

Benchmarks

- Why might validity/applicability of a benchmark change over time?
  - User requirements change
    - Benchmark may no longer reflect typical application mix, losing predictive value
    - Larger file input sizes may affect results (I/O vs. CPU contribution)
  - Technology changes
    - SPEC92 "gcc" – CPU intensive
    - Over time, CPU performance improves faster than I/O
    - I/O becomes dominant contributor
  - Implementations change
    - New algorithms → different instruction mix, behavior
    - Use of 3rd party packages (e.g. databases, libraries)
  - Operating systems
    - Multi-tasking overhead
      - Concrete example: lots of registers → faster CPU; context swap?

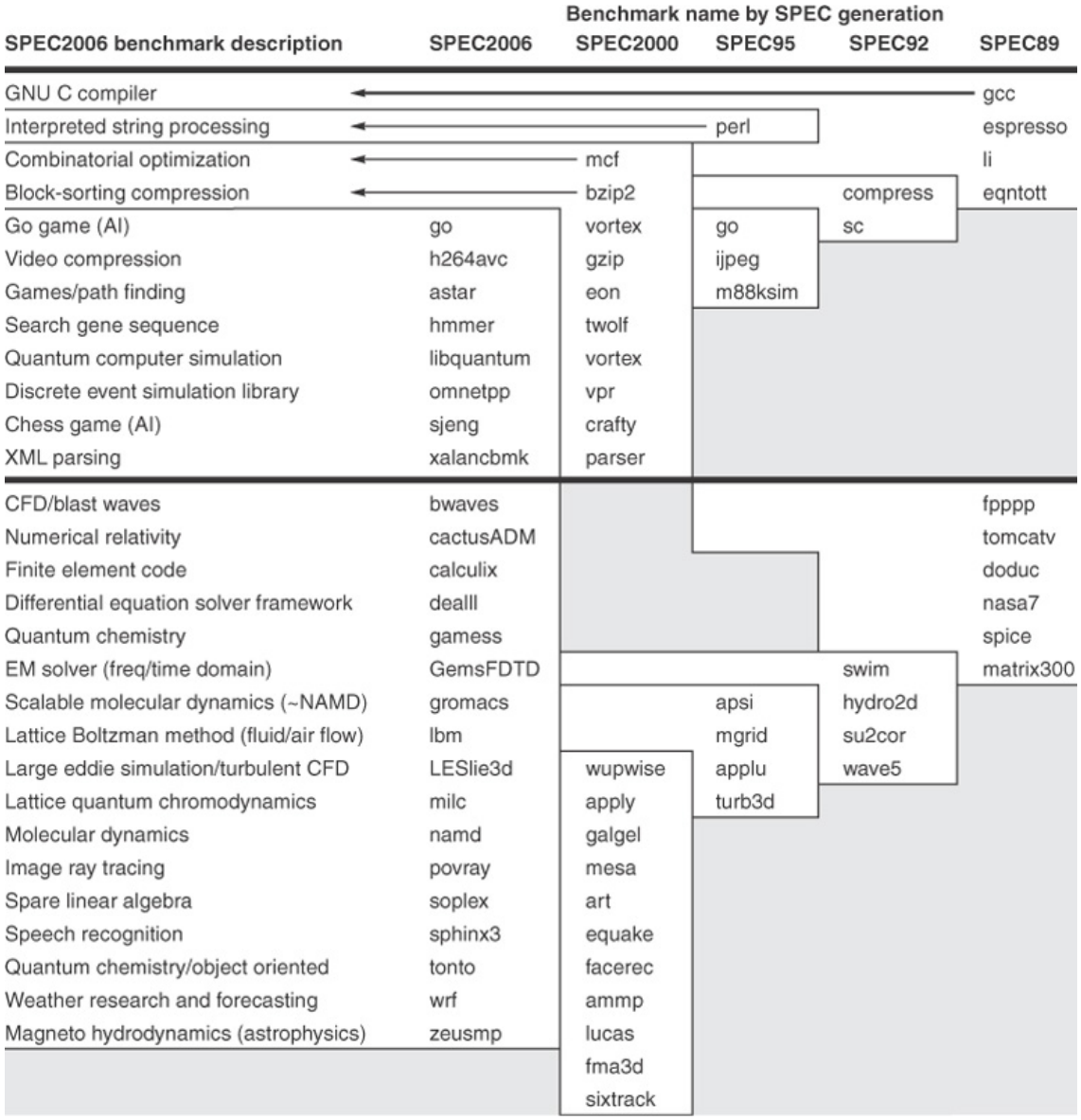
Benchmarks

- Why might validity/applicability of a benchmark change over time?
  - Compiler technology
    - Optimizations eliminate what were once relevant measures
  - Servers
    - E-commerce
      - More database transactions
      - Credit card processing
      - Networking to other web-based services, advertisers
  - PCs
    - More graphics intensive
    - Simultaneous loading (e.g. streaming audio while working)

SPEC benchmarks over time

12 programs
9 C
3 C++

17 programs
6 Fortran
4 C++
3 C
4 C/Fortran



SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

Integer

Floating point

Fallacy

- Peak performance tracks observed performance
  - Peak performance is only useful as the level of performance a system is guaranteed not to exceed!
  - Typical performance can vary 10x or more from peak performance
    - Example: Size of benchmark dovetails with key parameters
      - Cache size – no cache misses beyond initial access
    - Example: Cray and Hitachi

Measurement	Cray X-MP	Hitachi S810/20	Performance
A(i) = B(i) * C(i) + D(i) * E(i) (vector length 1000 done 100,000 times)	2.6 secs	1.3 secs	Hitachi two times faster
Vectorized FFT (vector lengths 64, 32, . . . , 2)	3.9 secs	7.7 secs	Cray two times faster

Fallacy

- The best design for a computer is one that optimizes the primary objective without considering implementation
  - Example: Itanium (64-bit not x86 architecture)
    - WSJ Article (AMD & Intel) 64-bit
  - Example: AMD K5 (Pentium Pro response)
    - Yield problems delayed introduction until it was no longer competitive
  - Example: Intel iAPX 432
    - "Best and brightest"
    - OS support, protection, object-orientation
    - Late and incompatible

Fallacy

- Synthetic Benchmarks Predict Performance for Real Programs
  - Optimizing compilers may confer differential advantages not based on the system architecture
    - Concrete example: Fortran Whetstone
      - X = SQRT(EXP(ALOG(X)/T1))
      - X = EXP(ALOG(X)/(2 x T1))
      - Because SQRT(EXP(X)) = EXP(X/2)
  - Synthetic benchmarks may under-reward performance-enhancing optimizations
    - Whetstone loops with few iterations
      - System which optimizes loop branch prediction wouldn't fare as well on benchmark as in practice
  - Synthetic benchmarks may not take into account effects of real world systems (loading, context switching)
    - SCSI drives outperform ATA under loaded conditions

Fallacy

- MIPS is an accurate measure for comparing performance among computers. MIPS is "Millions of instructions per second"

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution time} \times 10^6} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}$$

- Problems
  - What's an instruction? Depends upon ISA
  - MIPS can vary among programs on same computer
  - MIPS can vary inversely with performance
    - HW floating point instructions vs. SW routine
    - HW faster but executes fewer instructions

Pitfall

- Comparing hand-coded assembly and compiler-generated, high-level language performance
  - Hand-coded assembly less likely to be used except in embedded systems
  - Benchmarks may permit hand-coding
  - Unless you also hand-code and your application resembles the benchmark (making the same optimizations possible) actual performance will not match the benchmark

Machine	EEMBC benchmark set	Compiler-generated performance	Hand-coded performance	Ratio hand/ compiler
Trimedia 1300 @166 MHz	Consumer	23.3	110.0	4.7
BOPS Manta @ 136 MHz	Telecomm	2.6	225.8	86.8
TI TMS320C6203 @ 300 MHz	Telecomm	6.8	68.5	10.1

Pitfall

- Neglecting the cost of software in either evaluating a system or examining cost-performance
  - RISC vs. CISC
    - Reduced complexity → in faster, cheaper implementation
    - Software compatibility
      - X86 able to run installed base of "legacy" code
      - People were using that "complexity" in variety of ways
    - Cost of software (and software change) not zero (23% - 38%)
      - In dollars or attitudes
        - » Mac aficionados
    - Intel iAPX 432 (not X86 architecture)

Pitfall

- Falling prey to Amdahl's Law
  - Don't forget to assess the potential impact of an improvement before expending the effort to make it