CMPE 411
Computer Architecture

Lecture 5

Performance Benchmarks

September 14, 2017

www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm

Lecture's Overview

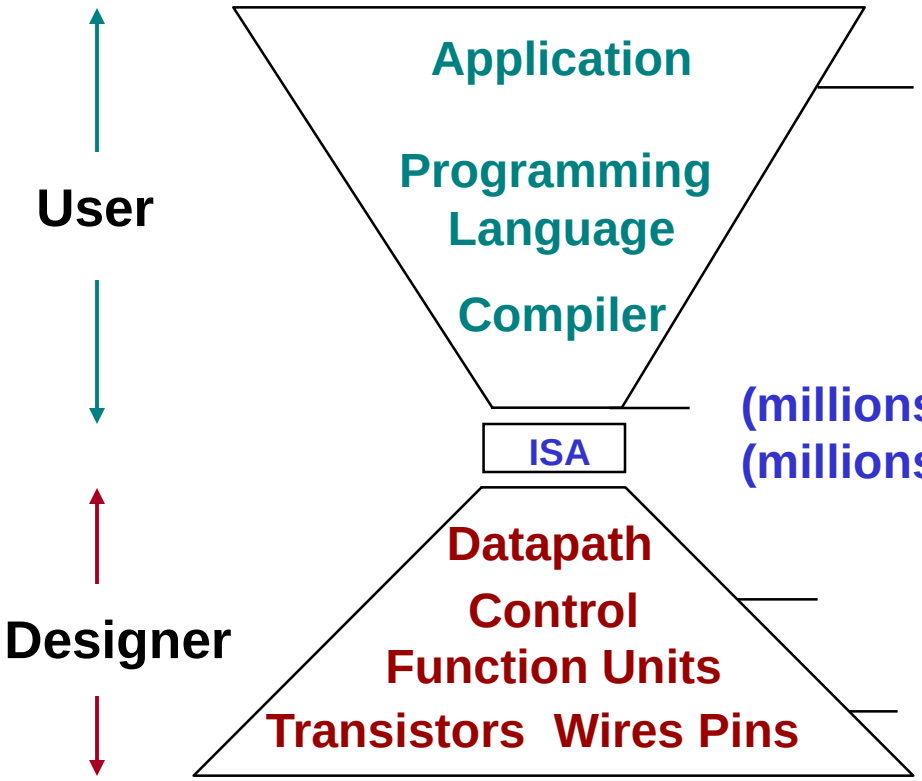
❑ *Previous Lecture:*

- Why measuring computer performance is important & subtle
  (Execution time is the only unimpeachable measure of performance)
- Different performance metrics
  (response time, throughput, CPU time)
- Performance comparison

❑ *This Lecture:*

- Performance reports and summary
- Selection of programs for performance evaluation
- Widely used benchmark programs
- Example industry metrics (e.g. MIPS, MFLOP, etc.)

Metrics of Performance



➔ Maximizing performance means minimizing response (execution) time

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

* Figure is courtesy of Dave Patterson

Calculation of CPU Time

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

	Instr. Count	CPI	Clock Rate
Program	X		
Compiler	X	X	
Instruction Set	X	X	
Organization		X	X
Technology			X

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Where: C_i is the count of number of instructions of class *i* executed

CPI_i is the average number of cycles per instruction for that instruction class

n is the number of different instruction classes

Performance Reports

Hardware	
Model number	Powerstation 550
CPU	41.67-MHz POWER 4164
FPU (floating point)	Integrated
Number of CPU	1
Cache size per CPU	64K data/8k instruction
Memory	64 MB
Disk subsystem	2 400-MB SCSI
Network interface	N/A
Software	
OS type and revision	AIX Ver. 3.1.5
Compiler revision	AIX XL C/6000 Ver. 1.1.5 AIX XL Fortran Ver. 2.2
Other software	None
File system type	AIX
Firmware level	N/A
System	
Tuning parameters	None
Background load	None
System state	Multi-user (single-user login)

Guiding principle is *reproducibility* (report environment & experiments setup)

Comparing & Summarizing Performance

	Computer A	Computer B
Program 1 (seconds)	1	10
Program 2 (seconds)	1000	100
Total time (seconds)	1001	110

- Wrong summary can present a confusing picture
  - ➔ A is 10 times faster than B for program 1
  - ➔ B is 10 times faster than A for program 2
- Total execution time is a consistent summary measure
- The relative execution times for the same workload is an informative performance summary
  - ➔ Assuming that programs 1 and 2 are executing for the same number of times on computers A and B

$$\frac{\text{CPU Performance (B)}}{\text{CPU Performance (A)}} = \frac{\text{Totel execution time (A)}}{\text{Totel execution time (B)}} = \frac{1001}{110} = 9.1$$

Execution time is the only valid and unimpeachable measure of performance

Performance Summary (Cont.)

$$\text{Arithmetic Mean (AM)} = \frac{1}{n}\sum_{i=1}^{n}\text{Execution\_Time}_i$$

$$\text{Weighted Arithmetic Mean (WAM)} = \sum_{i=1}^{n} w_i \times \text{Execution\_Time}_i$$

Where: n is the number of programs executed

w_i is a weighting factor that indicates the frequency of executing program #i

$$\text{with } \sum_{i=1}^{n} w_i = 1 \quad \text{and} \quad 0 \le w_i \le 1$$

- Weighted arithmetic means summarize performance while tracking exec. time
- Through the use of weights, a weighted arithmetic mean can adjust for different running times, balancing the contribution of each benchmark in the summary
- Never calculate AM after normalizing exec. time relative to a reference machine

	Time on A	Time on B	Norm. to A		Norm. to B	
			A	B	A	B
Program 1	1	10	1	10	0.1	1
Program 2	1000	100	1	0.1	10	1
AM of time or normalized time	500.5	55	1	5.05	5.05	1

Performance Summary (Cont.)

$$\text{Geometric Mean (GM)} = \sqrt[n]{\prod_{i=1}^{n} \text{Execution\_Time\_ratio}_i}$$

Where: *n* is the number of programs executed

With $$\frac{\text{Geometric Mean}(X_i)}{\text{Geometric Mean}(Y_i)} = \text{Geometric Mean}\left(\frac{X_i}{Y_i}\right)$$

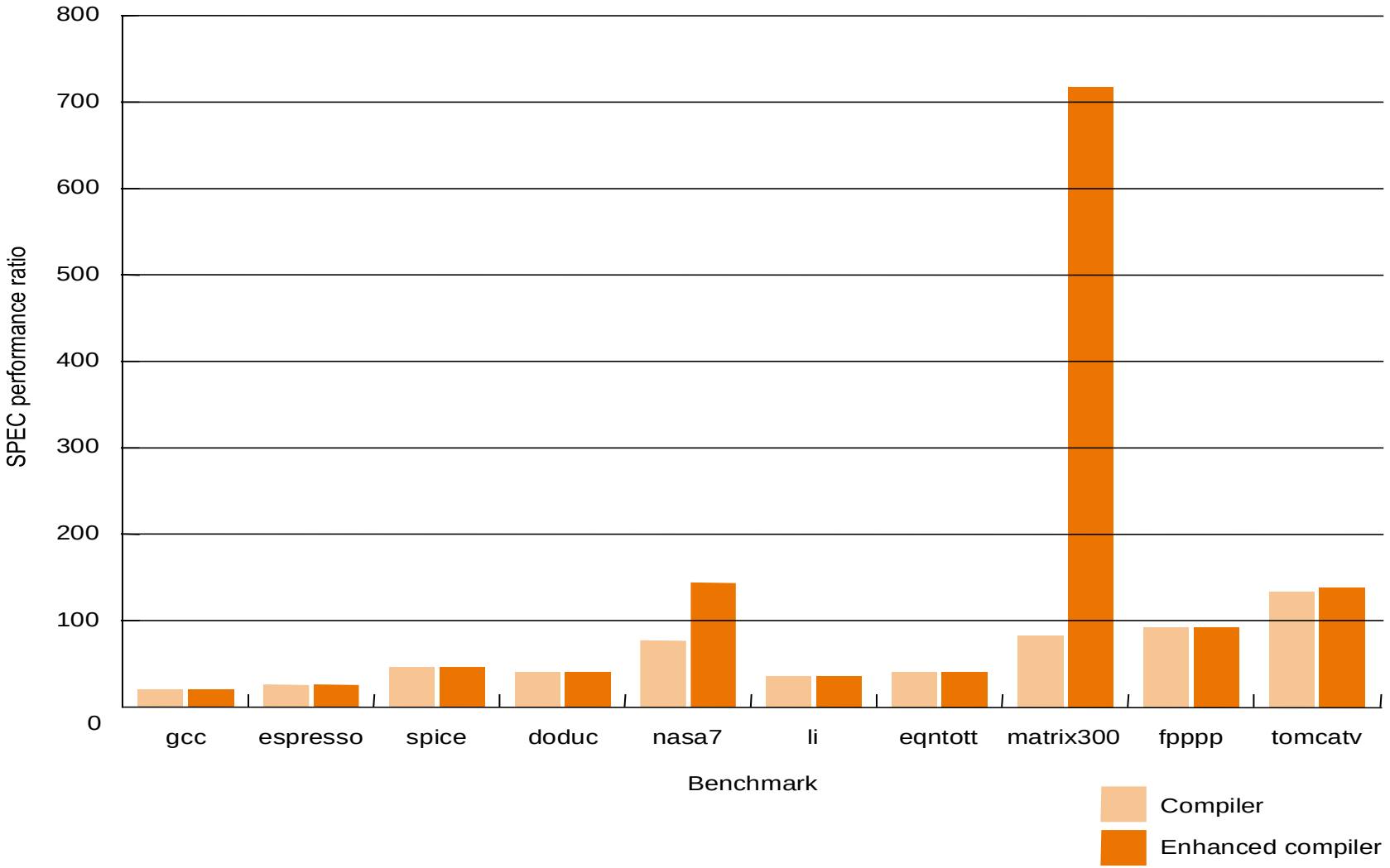
➔ Geometric mean is suitable for reporting average normalized execution time

	Time on A	Time on B	Norm. to A		Norm. to B	
			A	B	A	B
Program 1	1	10	1	10	0.1	1
Program 2	1000	100	1	0.1	10	1
AM of time or normalized time	500.5	55	1	5.05	5.05	1
GM of time or normalized time	31.62	31.62	1	1	1	1

Performance Benchmarks

- Many widely-used benchmarks are small programs that have significant locality of instruction and data reference
- Universal benchmarks can be misleading since hardware and compiler vendors might optimize their design for ONLY these programs
- The best types of benchmarks are real applications since they reflect the end-user interest
- Architectures might perform well for some applications and poorly for others
- Compilation can boost performance by taking advantage of architecture-specific features
- Application-specific compiler optimization are becoming more popular

Effect of Compilation



App. and arch. specific optimization can dramatically impact performance

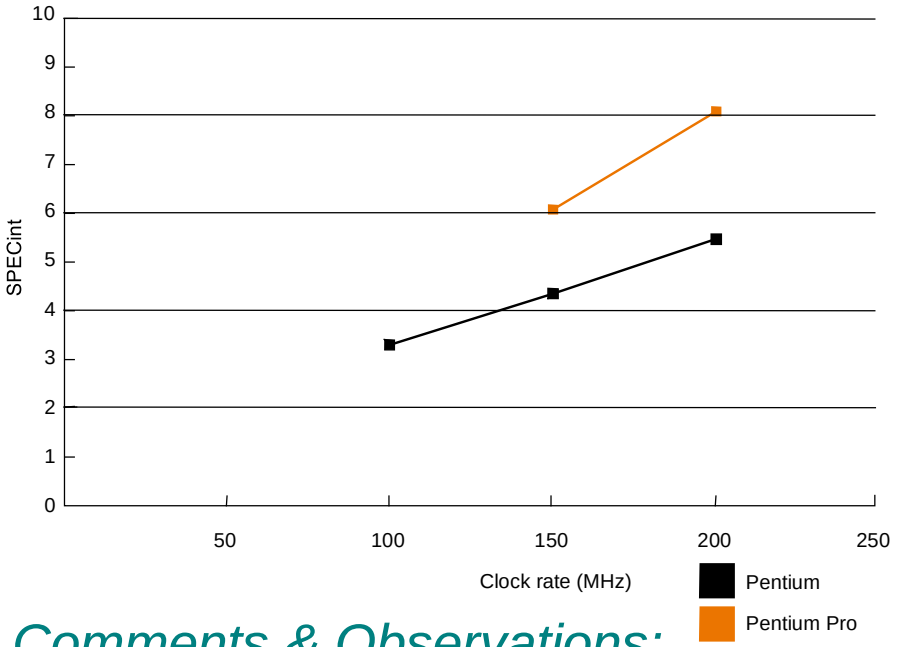
The SPEC Benchmarks

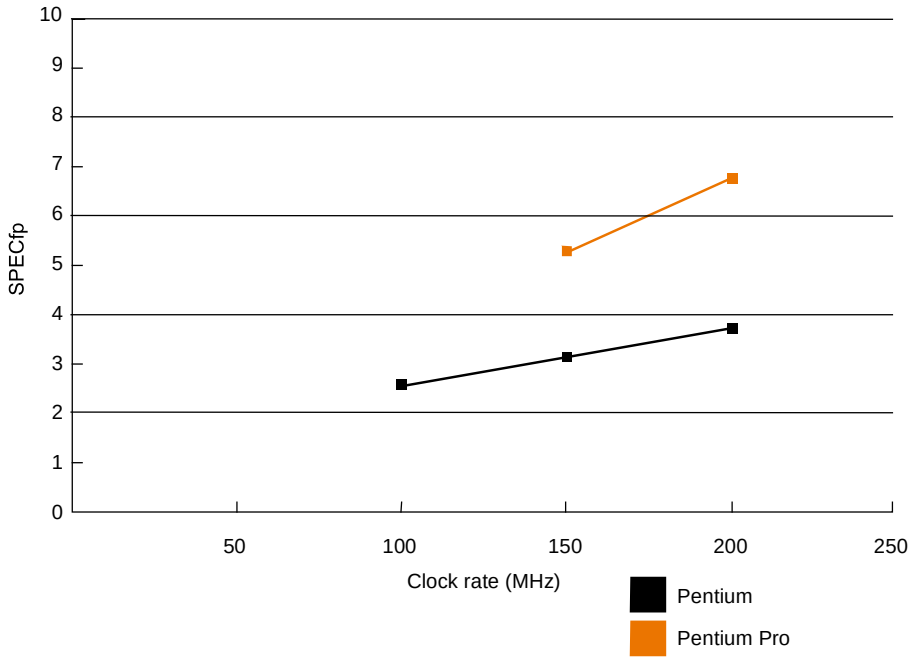
- SPEC stands for System Performance Evaluation Cooperative suite of benchmarks
- SPEC is created by a set of companies to improve the measurement and reporting of CPU performance
- SPEC2006 is the latest suite that consists of 12 integer and17 floating-point (in) programs (written in C, C++ and Fortran 77)
- Customized SPEC suites have been recently introduced to assess performance of graphics and transaction systems
- Since SPEC requires running applications on real hardware, the memory system has a significant effect on performance

$$\text{SPEC ratio} = \frac{\text{Execution time on SUN SPARCstation 10/40}}{\text{Execution time on the measure machine}}$$

- Bigger numeric values of the SPEC ratio indicate faster machine (performance = 1/execution time)

SPEC95 for Pentium and Pentium Pro





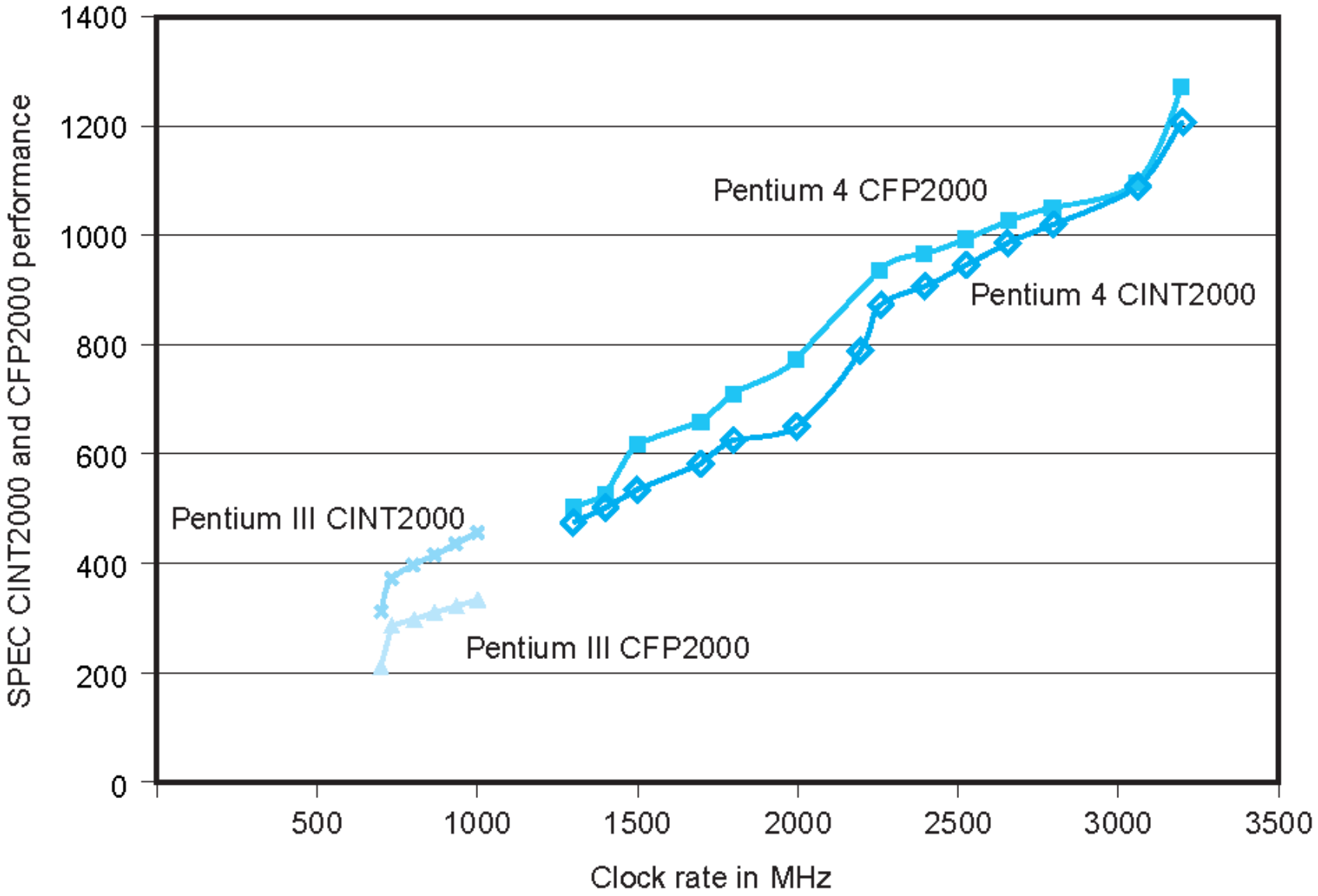
Comments & Observations:

- The performance measured may be different on other Pentium-based hardware with different memory system and using different compilers
- At the same clock rate, the SPECint95 measure shows that Pentium Pro is 1.4-1.5 times faster while the SPECfp95 shows that it is 1.7-1.8 times faster
  (mostly due to enhanced internal architecture)
- When the clock rate is increased by a certain factor, the processor performance increases by a lower factor most notably in the SPECfp95
  (due to memory system)
- Performance of large applications is more sensitive to memory system

SPEC Benchmarks www.spec.org

Integer benchmarks		FP benchmarks	
gzip	compression	wupwise	Quantum chromodynamics
vpr	FPGA place & route	swim	Shallow water model
gcc	GNU C compiler	mgrid	Multigrid solver in 3D fields
mcf	Combinatorial optimization	applu	Parabolic/elliptic pde
crafty	Chess program	mesa	3D graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition (NN)
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory interpreter	facerec	Facial image recognition
vortex	Object oriented database	ammp	Computational chemistry
bzip2	compression	lucas	Primality testing
twolf	Circuit place & route	fma3d	Crash simulation fem
		sixtrack	Nuclear physics accel
		apsi	Pollutant distribution

Example SPEC 2000 Ratings



* Slide is courtesy of Mary Jane Irwin

Using MIPS as a Performance Metric

- MIPS stands for Million Instructions Per Second and is one of the simplest metrics, which is valid in a limited context

$$\text{MIPS (native MIPS)} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

- There are three problems with MIPS:
  - ➔ MIPS specifies the instruction execution rate but does not take into account the capabilities of the instructions
  - ➔ Computers does not have the same MIPS rating, as MIPS varies between programs on the same computer
  - ➔ MIPS can vary inversely with performance (see next example)

The use of MIPS is simple and intuitive, faster machines have bigger MIPS

Example

Consider the machine with the following three instruction classes and CPI:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

Now suppose we measure the code for the same program from two different compilers and obtain the following data:

Code from	Instruction count in (billions) for each instruction class		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

Assume that the machine's clock rate is 500 MHz. Which code sequence will execute faster according to MIPS? According to execution time?

Answer:

Using the formula: $$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Sequence 1: CPU clock cycles = $(5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$ cycles

Sequence 2: CPU clock cycles = $(10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$ cycles

Example (Cont.)

Using the formula: $$\text{Exection time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

Sequence 1: Execution time = $(10\times10^9)/(500\times10^6)$ = 20 seconds

Sequence 2: Execution time = $(15\times10^9)/(500\times10^6)$ = 30 seconds

Therefore compiler 1 generates a faster program

Using the formula: $$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

Sequence 1: $$\text{MIPS} = \frac{(5+1+1)\times10^9}{20\times10^6} = 350$$

Sequence 2: $$\text{MIPS} = \frac{(10+1+1)\times10^9}{30\times10^6} = 400$$

Although compiler 2 has a higher MIPS rating, the code from generated by compiler 1 runs faster

Historic Perspective

- In early computers most instructions of a machine took the same execution time of others
- The measure of performance for old machines was the time required for performing an individual operation (e.g. addition)
- New computers have diverse set of instructions that require different execution time
- The relative frequency of instructions across many programs was calculated
- The average instruction execution time was measured by multiplying the time of each instruction by its frequency
- The CPI is the average instruction execution time measured in clock cycles
- The average instruction execution time was a small step to MIPS that grew in popularity

Native, Peak and Relative MIPS, & FLOPS

- Peak MIPS is obtained by choosing an instruction mix that maximizes the CPI, even if the mix is impractical
- To make MIPS more practical among different instruction sets, a relative MIPS is introduced to compare machines to an agreed-upon reference machine (e.g. Vax 11/780)

$$\text{Relative MIPS} = \frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_{\text{unrated}}} \times \text{MIPS}_{\text{reference}}$$

- With the fast development in the computer technology, reference machine cannot be guaranteed to exist
- Relative MIPS is practical for evolving design of the same computer
- With the introduction of supercomputers around speeding up floating point computation, the term MFLOP is introduced analogous to MIPS

Synthetic Benchmarks

- Synthetic benchmarks are artificial programs that are constructed to match the characteristics of large set of programs
- Whetstone (scientific programs in Algol → Fortran) and Dhrystone (systems programs in Ada → C) are the most popular synthetic benchmarks
- Whetstone performance is measured in "Whetstone per second" – the number of executions of one iteration of the whetstone benchmark
- Synthetic benchmarks suffer the following drawbacks:
  1. They do not reflect the user interest since they are not real applications
  2. They do not reflect real program behavior (e.g. memory access pattern)
  3. Compiler and hardware can inflate the performance of these programs far beyond what the same optimization can achieve for real-programs

Examples from the Dhrystone set:

- By assuming word alignment in string copy a 20-30% performance improvement could be achieved, although 99.70-99.98% of typical string copies could NOT use such optimization
- Compiler optimization could easily discard 25% of the Dhrystone code for single iteration loops and inline procedure expansion

Amdahl's Law

The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by the improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

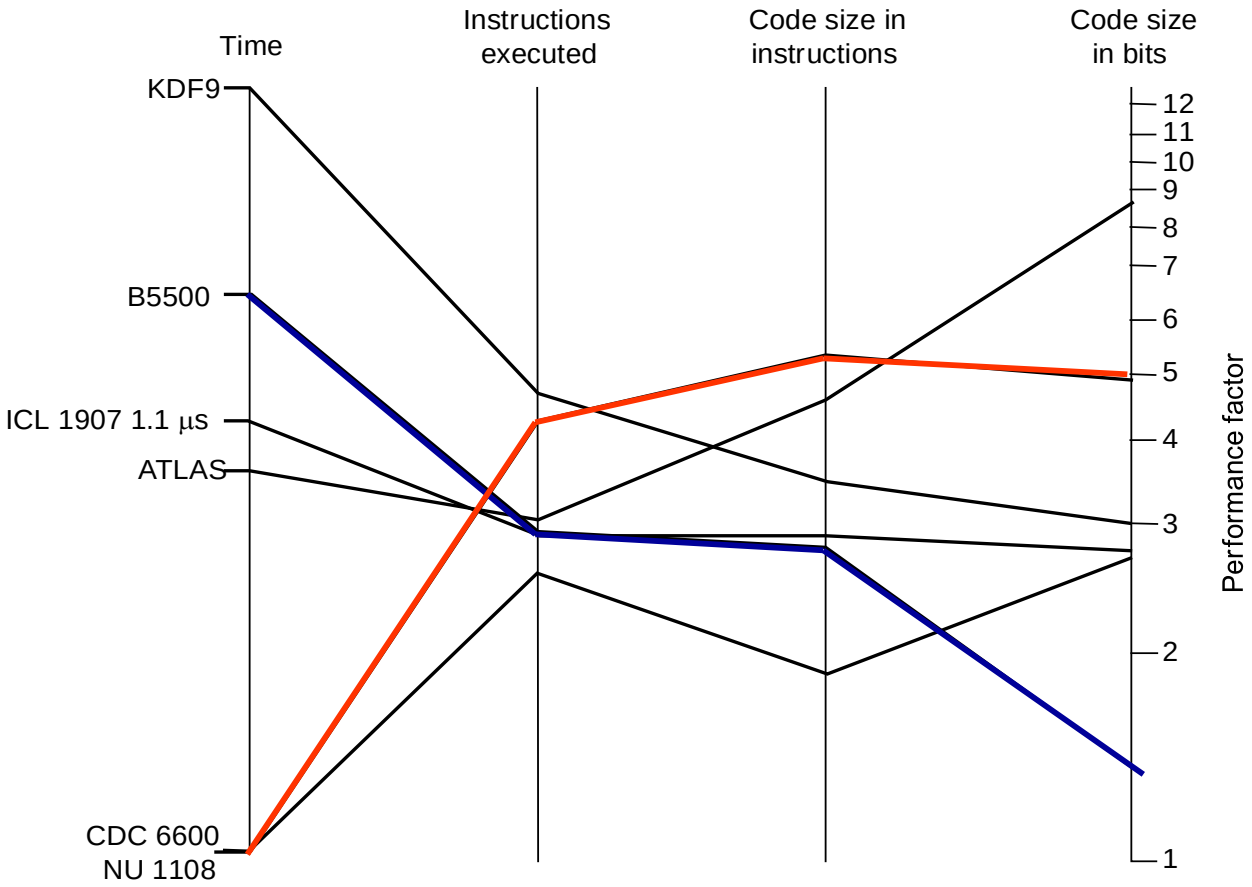
- A common theme in Hardware design is to *make the common case fast*
- Increasing the clock rate would not affect memory access time
- Using a floating point processing unit does not speed integer ALU operations

Example: Floating point instructions improved to run 2X; but only 10% of actual instructions are floating point

$$\textit{Exec-Time}_{new} = \textit{Exec-Time}_{old} \times (0.9 + .1/2) = 0.95 \times \textit{Exec-Time}_{old}$$

$$\textit{Speedup}_{overall} = \textit{Exec-Time}_{old} / \textit{Exec-Time}_{new} = 1/0.95 = 1.053$$

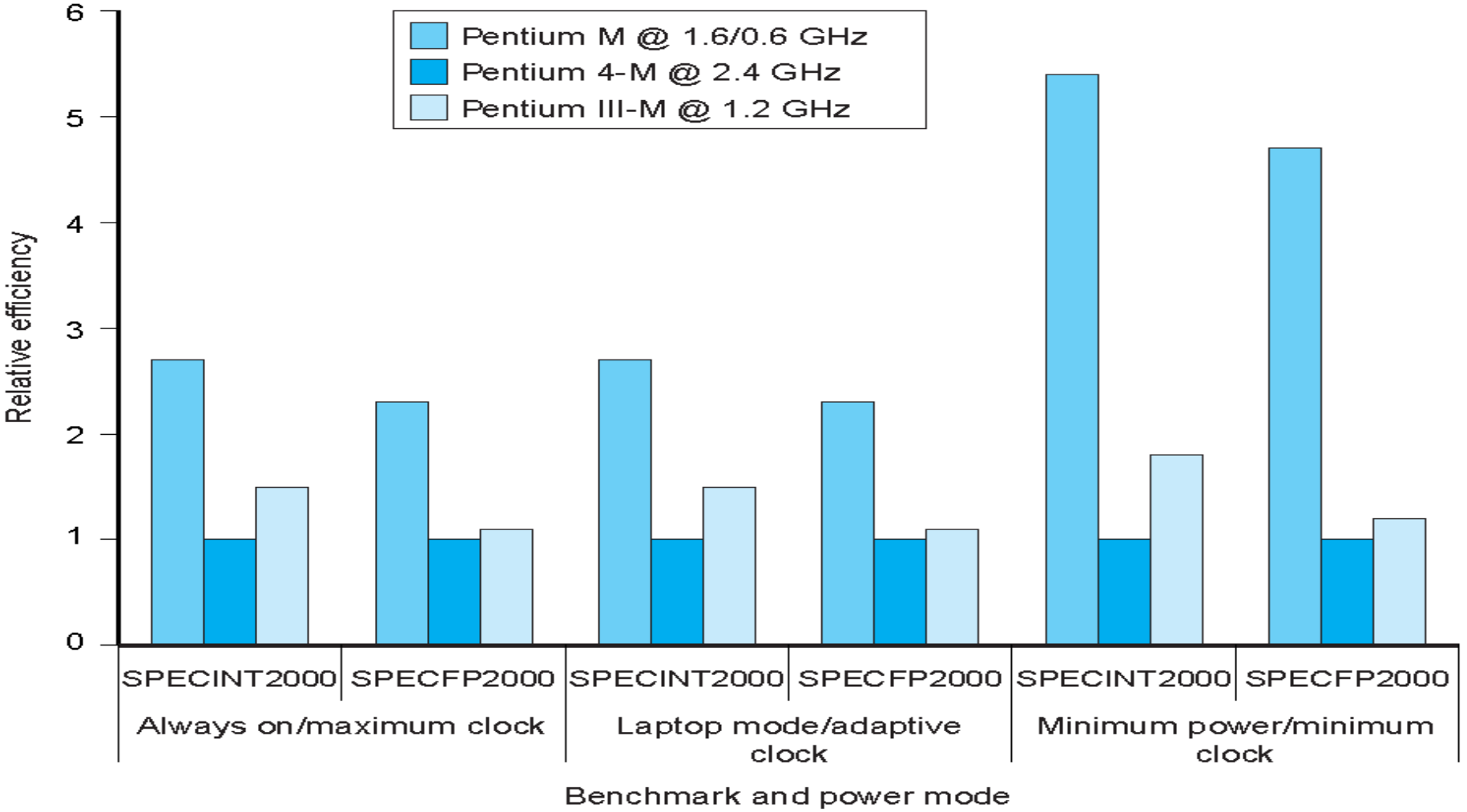
Can Hardware-Indep Metrics Predict Performance?



- The Burroughs B5500 is designed specifically for Algol 60 programs
- Although CDC 6600's programs are over 3 times as big as those of B5500, yet the CDC machine runs them almost 6 times faster
- Code size cannot be used as an indication for performance

Other Performance Metrics

- Power consumption – especially in the embedded market where battery life is important (and passive cooling)



* Slide is courtesy of Mary Jane Irwin

Final Remarks

- Designing for performance only without considering cost is unrealistic
- In the supercomputing industry performance is the primary and dominant goal
- Low-end personal and embedded computers are extremely cost driven
- Performance depends on three major factors: number of instructions, cycles consumed by instruction execution and the size of a clock cycle
- The art of computer design lies not in plugging numbers in a performance equation, but in accurately determining how design alternatives will affect performance and cost
- Design cost depends on many technical and non-technical factors and is very challenging to evaluate

Conclusion

- ***Summary***
  - ➔ Performance reports, summary and comparison
    (Experiment reproducibility, arithmetic and weighted arithmetic means)
  - ➔ Widely used benchmark programs
    (SPEC, Whetstone and Dhrystone)
  - ➔ Example industry metrics
    (e.g. MIPS, MFLOP, etc.)
  - ➔ Increasing CPU performance can come from three sources
    1. Increases in clock rate
    2. Improvement in processor utilization that lower the CPI
    3. Compiler enhancement that lower the instruction count or generate instructions with lower CPI
- ***Next Lecture***
  - ➔ Number representation and Computer Arithmetic
  - ➔ Addition, Subtraction and logical operations

Read section 1.9 in 5th Ed., or 1.8 in 4th Ed. of the textbook