

## PERFORMANCE

- Performance is a measure of how fast something works
- Who cares?
  - ▷ Understanding it enables you to get the best deal
    - Which system has the most “bang for the buck”?
  - ▷ Performance determines whether a problem is numerically solvable and what methods can be used to solve it
    - Program takes too long to run  $\Rightarrow$  can't solve problem
  - ▷ Leads to understanding of the underlying motivation for a given computer organization
    - Why is one system better than another?
    - Which hardware components are a problem?
    - How does the instruction set affect performance?
    - Why is some hardware better than others for different programs?

## PERFORMANCE IS TIME

- Transportation performance metrics
  - ▷ Throughput
    - Passenger miles per month
    - Performance improves with higher volume and/or distance
  - ▷ Passengers (users) measure performance in terms of time in-flight
- Long-distance telecommunication performance metric
  - ▷  $BL =$  bits per second ( $B$ ) times distance ( $L$ )
  - ▷ Performance improves with higher speed ( $B$ ) and greater distance ( $L$ )
- Computer performance metrics
  - ▷ For users: Execution time or response time
  - ▷ Transaction processing systems: Throughput

## A USER'S DEFINITION OF PERFORMANCE

- Execution time := total time required to run program
  - ▷ “wall-clock time”
  - ▷ Most significant parameter for:
    - Product development
    - Research
- Performance :=  $1/(\text{execution time})$ 
  - ▷ Affected by:
    - Processor speed
    - Concurrency of execution
    - I/O speed
    - Type of program
    - System workload

## RELATIVE PERFORMANCE

- Relative performance is the performance of one system compared to another, and is given by the performance ratio.
  - ▷ The performance ratio of System A to System B is:

$$\frac{\text{Performance}_A}{\text{Performance}_B} = n = \frac{\text{Execution}_B}{\text{Execution}_A}$$

- ▷ System A is  $n$  times faster than System B.

## EXAMPLE OF COST-PERFORMANCE ANALYSIS (1)

- Consider:
  - ▷ Systems  $S_1, S_2$ , with costs  $C_1$  and  $C_2$
  - ▷ Programs  $P_1$  and  $P_2$ , with execution times  $T_1$  and  $T_2$
- The **cost-performance ratio** for program  $P_j$  on system  $S_i$  is

$$\frac{\text{cost of } S_i}{\text{performance on } P_j} = C_i T_j$$

A *lower* value of  $C_i T_j$  means better performance for a given cost, or lower cost for a given performance

- Example:

System	$C_i$ (\$)	$T_1$ (s)	$C_i T_1$	$T_2$ (s)	$C_i T_2$
$S_1$	1,500	10	15,000	5	7,500
$S_2$	3,000	3	9,000	3	9,000

**UNIX TIME COMMAND (1)**

- `wotan% time linpackd.exe`  
`12.7u 0.1s 0:13 99% 0+764k 1+0io 0pf+0w`
- Interpretation of this example:
  - ▷ 12.7 seconds **user time** (u) — time spent running the program `linpackd.exe`
  - ▷ 0.1 seconds **system time** (s) — time spent by the Unix kernel on behalf of `linpackd.exe`
  - ▷ 13 seconds **elapsed time**
  - ▷ **CPU time** := user time + system time = 99% of elapsed time
  - ▷ Average memory used = 0 kb unshared + 764 kb shared
  - ▷ block I/O (io): 0 blocks input, 1 blocks output
  - ▷ 0 **page faults** (pf)
  - ▷ 0 **swaps** (w)

## UNIX TIME COMMAND (2)

- `tulia% time lincpackd.exe`  
`8.5u 0.3s 0:13 66% 0+804k 4+0io 41pf+0w`
- Interpretation of this example:
  - ▷ 8.5 seconds user time
    - $\text{tulias CPU performance} = \frac{12.7}{8.5} \times \text{wotans CPU performance}$
  - ▷ 13 seconds elapsed time
    - tulia and wotan had the same performance on this job
  - ▷ tulias CPU time for this program = 66% of elapsed time
    - 34 % of the elapsed time was spent waiting for other jobs
  - ▷ 41 page faults (pf)
    - The hard disk was accessed 41 times for data in virtual memory

## CPU PERFORMANCE

- CPU performance  $:= \frac{1}{\text{user CPU time} + \text{system CPU time}}$ 
  - ▷ Gives no information about performance of subprograms
  - ▷ Depends strongly on program, compiler options, and workload

## COMPUTER CZAR'S DEFINITION OF PERFORMANCE

- Throughput := total work done per unit time
- Utilization := % of available time spent executing user jobs
  - ▷ Increased by:
    - Having jobs enqueued, awaiting execution
    - Decreasing parallelization (no. of processors used by each job)
- Maximum throughput means:
  - ▷ High utilization
  - ▷ Low performance from user's point of view
  - ▷ Longer times to develop products or obtain research results

## CLOCK CYCLES

- Instead of using seconds to measure execution time, often we use clock cycles, aka clock ticks, clock periods, clocks, or cycles.
  - ▷ **Clock rate** (frequency) = cycles per second.
  - ▷ Measured in Hertz (1 Hz = 1 cycle/s).
- **Clock period** is the time between ticks of the clock and is measured in seconds per cycle.
  - ▷ Period = 1/frequency
- Example: A 200 MHz (MegaHertz) clock has a clock period of
$$\frac{1}{200 \times 10^6 \text{ Hz}} = 5 \times 10^{-9} \text{ seconds} = 5 \text{ nanoseconds.}$$
- **Warning:** Some people refer to the clock period as the clock rate; they are not the same thing!

**FUNDAMENTAL EQUATION FOR CPU TIME (1)**

- CPU time required to run a program:

$$\begin{aligned} \text{CPU execution time} &= \frac{\text{Instructions}}{\text{Program}} \\ &\times \frac{\text{Clock periods}}{\text{Instruction}} \\ &\times \frac{\text{Seconds}}{\text{Clock Period}} \end{aligned}$$

- $\frac{\text{Instructions}}{\text{Program}}$  is determined by:
  - ▷ The instruction set architecture
  - ▷ The compiler
  - ▷ The program

## FUNDAMENTAL EQUATION FOR CPU TIME (2)

- $\frac{\text{Clock periods}}{\text{Instruction}}$  is determined by:
  - ▷ Logic design (cost tradeoffs)
  - ▷ Instruction mix
- $\frac{\text{Seconds}}{\text{Clock period}}$  is determined by:
  - ▷ Semiconductor device properties
  - ▷ Logic design

## CLOCK PERIODS PER INSTRUCTION (CPI) (1)

- **CPI** is the *average* number of clock periods per instruction:

$$\text{CPI} := \frac{\text{clock periods}}{\text{instruction count}} = \frac{C}{I}$$

▷  $I$  = total number of instructions of all types

▷  $C$  = total number of clock periods

- Suppose there are  $N$  types of instructions

▷  $I_k$  = number of instructions of type  $k$ ;  $I = \sum_k I_k$

▷  $\text{CPI}_k$  = clock periods to execute *one* instruction of type  $k$

$$C = \sum_{k=1}^N I_k \times \text{CPI}_k \quad \Rightarrow \quad \text{CPI} = \frac{C}{I} = \sum_{k=1}^N \left( \frac{I_k}{I} \right) \times \text{CPI}_k$$

$$\text{CPI} = \sum_{k=1}^N F_k \times \text{CPI}_k$$

where  $F_k$  = fraction of type  $k$  instructions

$$F_k = \frac{I_k}{I} \Rightarrow 0 \leq F_k \leq 1$$

**CLOCK PERIODS PER INSTRUCTION (CPI) (2)**

- Example for a MIPS R2000 processor with a particular instruction mix:

$k$	Instr. type	$F_k$	$\text{CPI}_k$	$F_k \times \text{CPI}_k$
1	Load	0.30	2	0.60
2	Store	0.15	2	0.30
3	ALU op.	0.40	1	0.40
4	Branch	0.15	2	0.30
$\text{CPI} = \sum_{k=1}^4 F_k \times \text{CPI}_k$				1.6

## NUMBER OF CPU CLOCK CYCLES

- Two independent methods of computing  $NC =$  number of cycles in one program:

- ▷ In terms of execution time and clock frequency:

$$NC = ET * CF$$

- ▷ In terms of instruction count and clocks per instruction:

$$NC = IC * CPI$$

- ▷ From the equation

$$ET * CF = IC * CPI$$

one can find any one of  $ET$ ,  $CF$ ,  $IC$  or  $CPI$ , given the other three

## CLOCK PERIODS PER INSTRUCTION (CPI) (3)

- Computation of CPI using CPU time for two different systems  $S_1$ ,  $S_2$  running a program with  $100 \times 10^6$  instructions

System	Clock rate (MHz)	CPU time (s)	Clock periods(NC)	CPI
$S_1$	100	10	$1000 \times 10^6$	10
$S_2$	200	3	$600 \times 10^6$	6

## A PEAK PERFORMANCE METRIC: PEAK MIPS (1)

- MIPS := Millions of Instructions Per Second
  - ▷ The **peak MIPS** rating is calculated using the theoretical **peak rate** at which instructions can be issued
- For **average MIPS**, use the fundamental performance equation:

$$\begin{aligned} \text{MIPS} &= \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6} = \frac{\text{Instruction Count}}{\text{Clock Periods} \times \text{Cycle Time} \times 10^6} \\ &= \frac{\text{Instruction Count} \times \text{Clock Frequency}}{\text{Instruction Count} \times \text{CPI} \times 10^6} \end{aligned}$$

Then

$$\text{average MIPS} = \frac{\text{Clock Frequency}}{\text{CPI} \times 10^6}$$

## A PEAK PERFORMANCE METRIC: PEAK MIPS (2)

- MIPS := Millions of Instructions Per Second
  - ▷ The **peak MIPS** rating is calculated using the theoretical **peak rate** at which instructions can be issued
  - ▷ Neglects most real software and hardware properties:
    - Memory-cache or memory-CPU bandwidth
    - I/O speed
    - Type of program (locality of instruction references)
    - System workload
  - ▷ Also known as “Meaningless Indicator of Processor Speed”

## MEGAFLOPS

- MFLOPS := Millions of Floating-Point Operations Per Second
  - ▷ Peak (“guaranteed not to exceed”) MFLOPS:
    - Theoretical maximum rate at which floating-point operations can be performed (assuming all FP operations take equal time)
    - Example: Cray Y-MP, 1 processor
      - ◇ Clock period = 6 ns
      - ◇ 1 addition + 1 multiplication possible on each cycle
      - ◇ Peak MFLOPS =  $2 \times \text{clock frequency} = 333 \text{ MFLOPS}$
  - ▷ MFLOPS measured by a benchmarking program:
    - A program that can do useful work, *e.g.*, LINPACK
    - A synthetic benchmark, such as:
      - ◇ Livermore loops
      - ◇ SPEC

**THE LINPACK BENCHMARK (1)**

- LINPACK solves a system of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

using the Gaussian elimination algorithm

- ▷ Generates a random coefficient matrix  $\mathbf{A}$  and right-hand side  $\mathbf{b}$
- ▷ Timing carried out in the program (not externally):

```
call matgen(a,lda,n,b,norma)
t1 = secnds(0.0)
call dgefa(a,lda,n,ipvt,info)
time(1,1) = secnds(0.0) - t1
:
```

- ▷ Executes  $2n^3/3 + 2n^2$  floating-point operations (where matrix  $\mathbf{A}$  is  $n \times n$ )

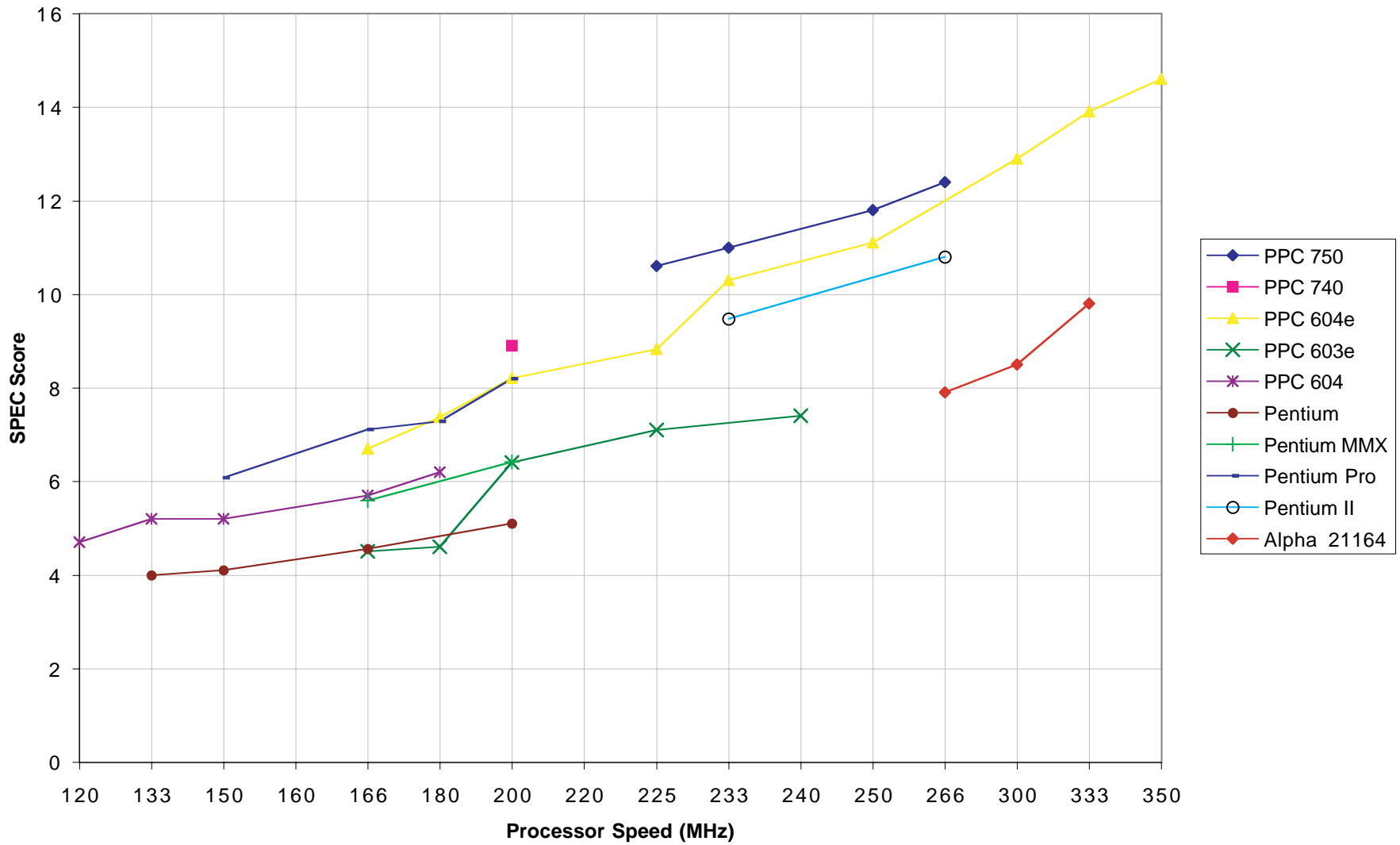
## THE LINPACK BENCHMARK (2)

- 2 standard benchmark problem sizes:
  - ▷  $100 \times 100$ :
    - No code changes allowed
    - Array size so small that pipeline latency is significant
    - May be the most indicative standard benchmark for computationally intensive engineering/scientific programs
  - ▷  $1000 \times 1000$ :
    - Any code change is allowed that does not change the problem being solved

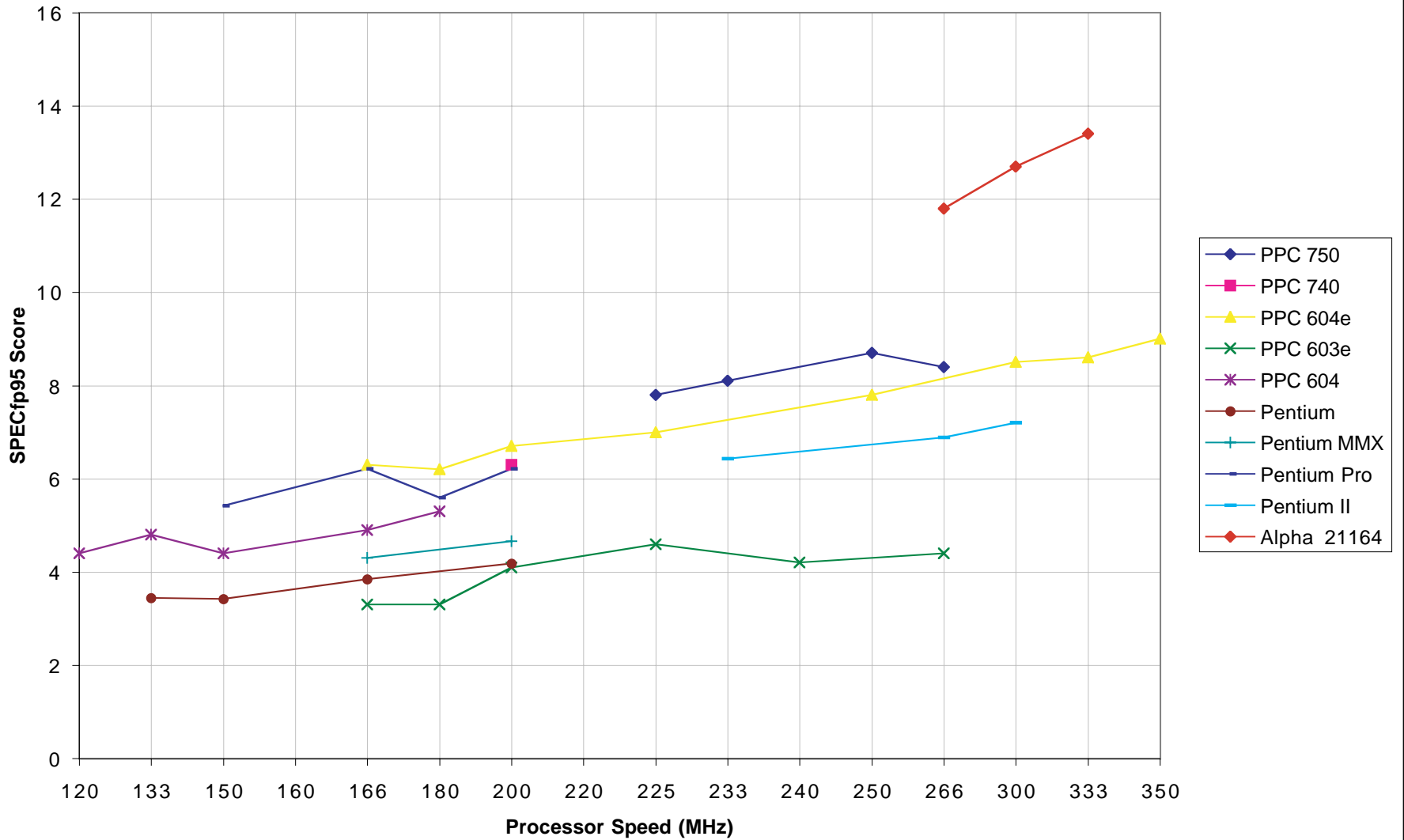
## THE SPEC BENCHMARK

- SPEC := Standard Performance Evaluation Corporation
  - ▷ Current CPU intensive benchmark suites:
    - SPECint95                    integer
    - SPECfp95                    floating point
  - ▷ Other benchmarks:
    - SDM                    UNIX Software Development Workloads
    - SFS                    System level file server (NFS) workload
    - SPEChpc96            High-performance computing benchmarks
- SPEC benchmarks are small “kernels” of code extracted from real programs
  - ▷ May run unrealistically fast
  - ▷ Programs with large arrays that are accessed using a large stride in memory are likely to run much more slowly than the same program with smaller arrays and a smaller stride

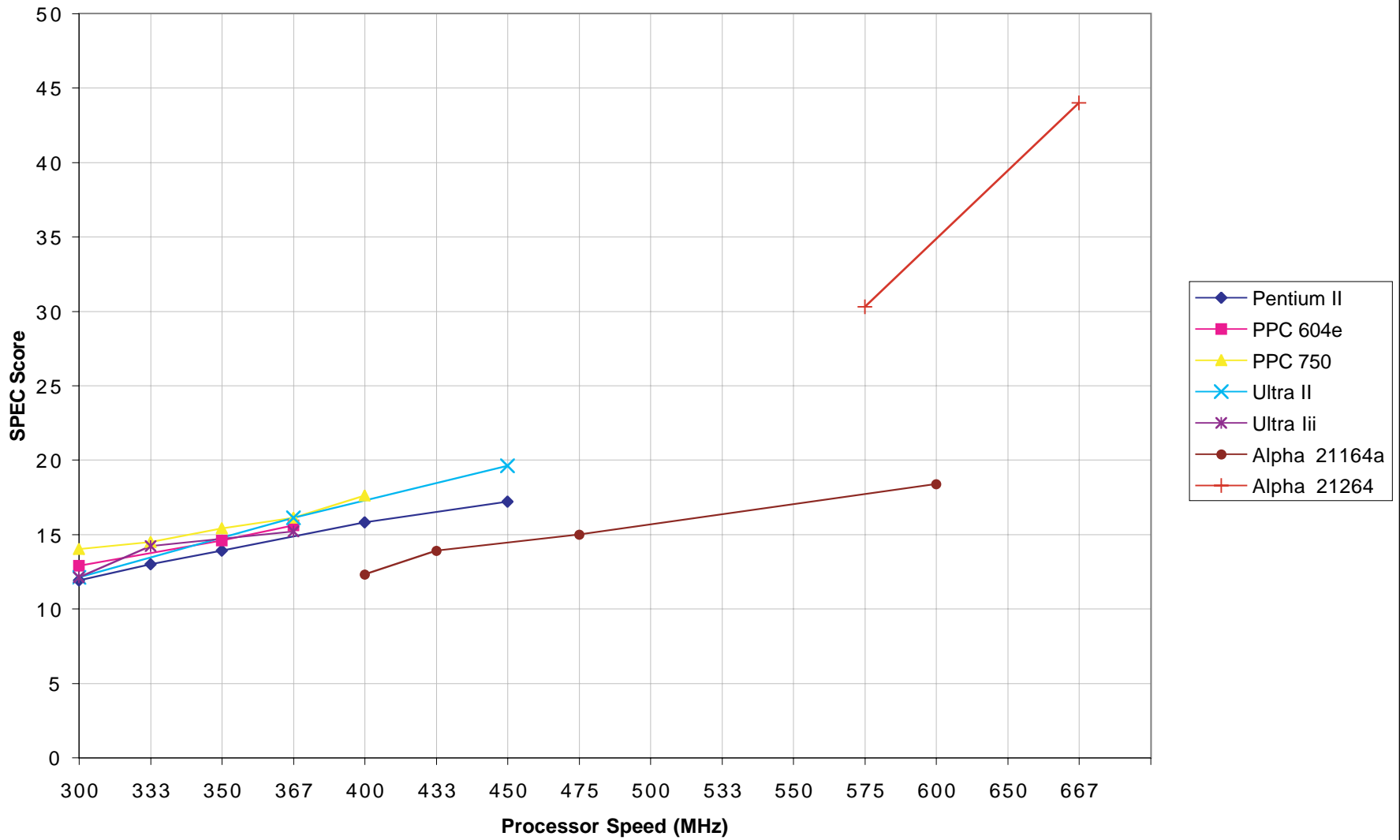
### SPECint95 Comparison



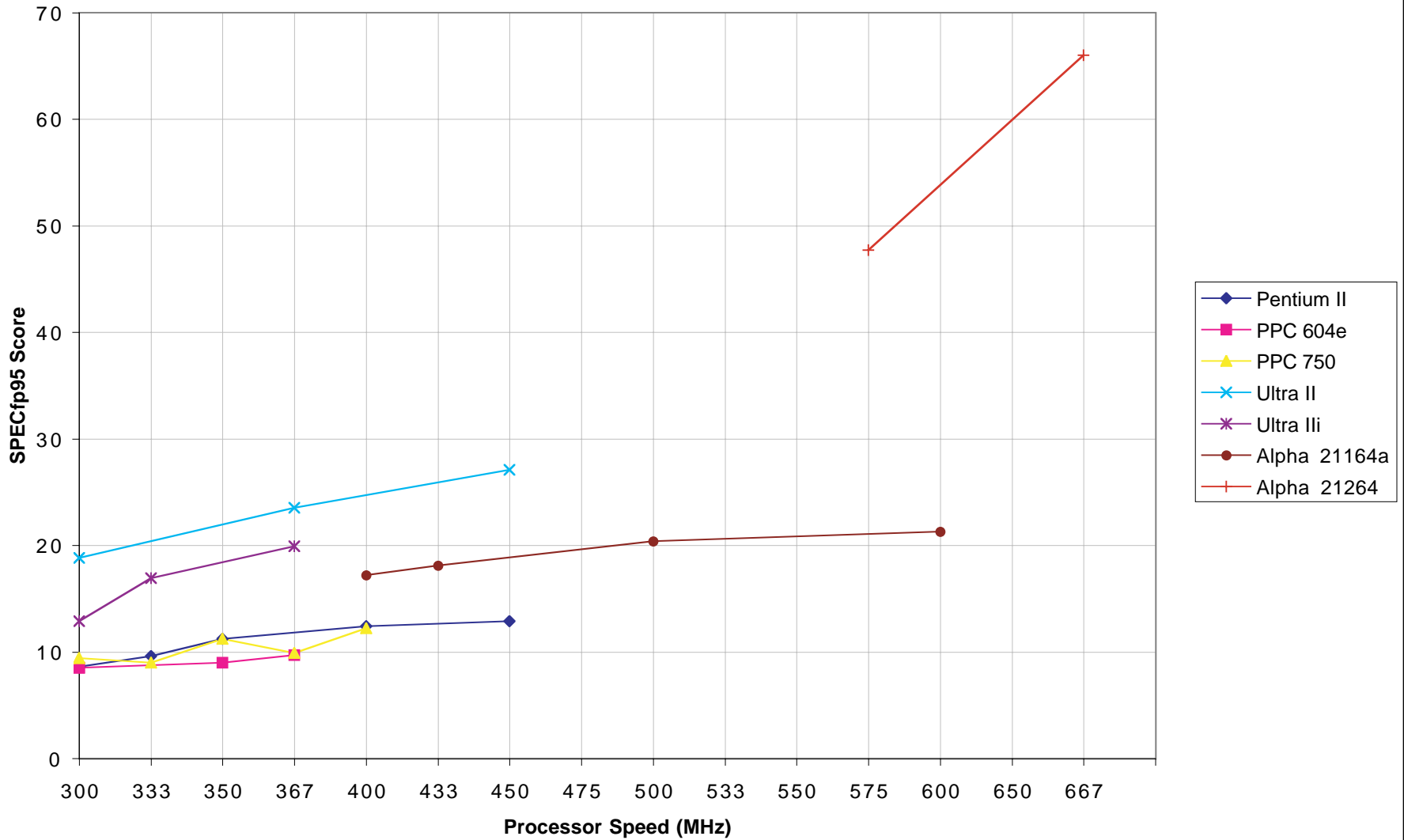
SPECfp95 comparison



### SPECint95 Comparison



SPECfp95 comparison



## SPEEDUP

- Definition of speedup:

$$\text{Speedup} := \frac{\text{Original Execution Time}}{\text{Improved Execution Time}}$$

- The maximum speedup that can be obtained by using a faster mode of execution is limited by the fraction of the time the faster mode can be used.

**AMDAHL'S LAW (1)**

- Calculation of speedup due to an enhancement of a part of a system or program:

$$\text{Overall speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

where

New execution time

= Execution time of enhanced part

+ Execution time of unenhanced part

= (Old execution time)

$\times \left( (1 - \text{Fraction enhanced}) + \frac{\text{Fraction enhanced}}{\text{Speedup of the enhanced part}} \right)$

**AMDAHL'S LAW (2)**

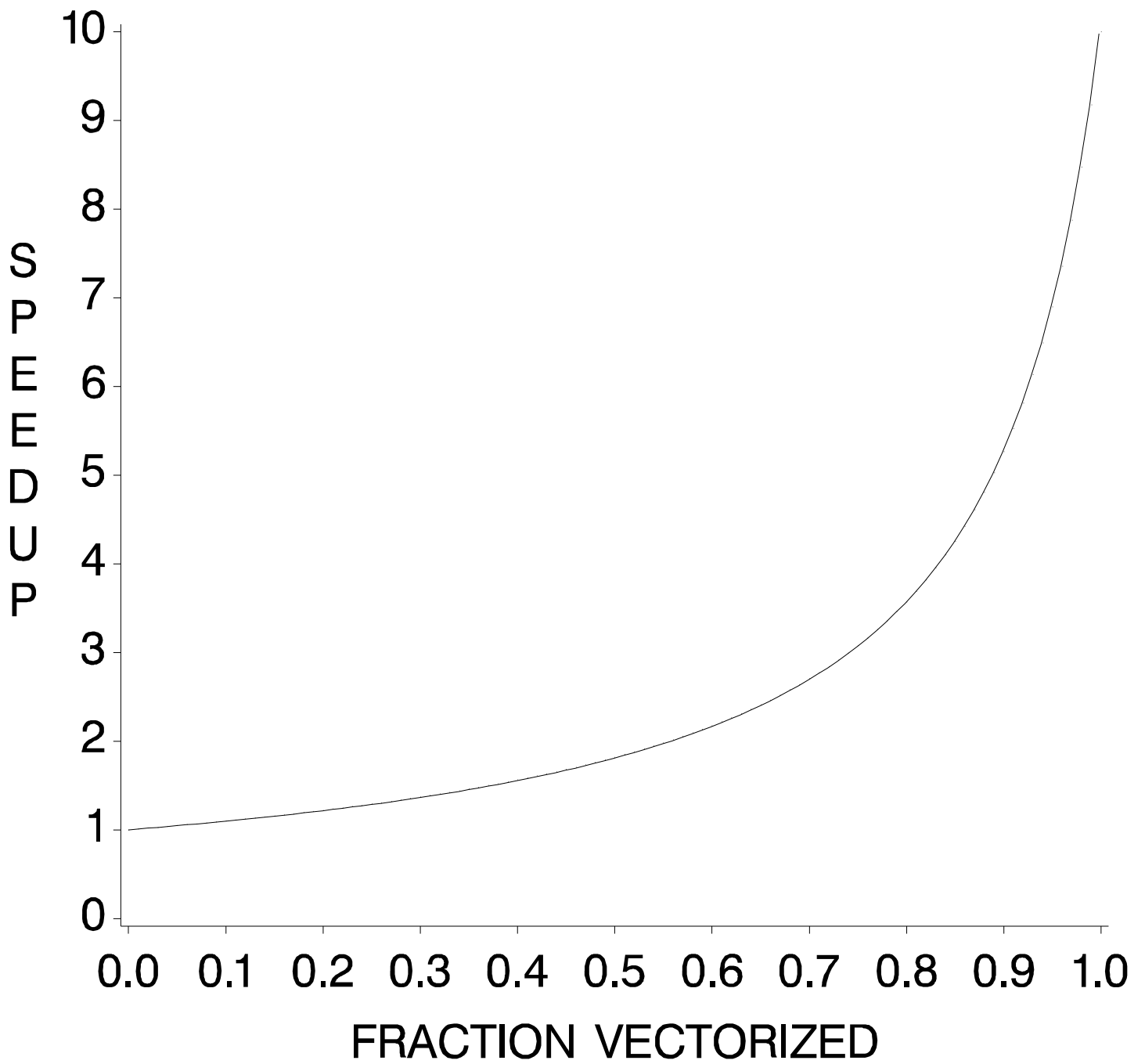
- Overall speedup due to an enhancement:

overall speedup

$$= \frac{1}{(1 - \text{fraction enhanced}) + \left( \frac{\text{fraction enhanced}}{\text{speedup of the enhanced part}} \right)}$$

- The maximum speedup that can be obtained by using a faster mode of execution is limited by the fraction of the time the faster mode can be used. Examples include:
  - ▷ Improving your Web-surfing speed
  - ▷ Parallelization of computer programs
  - ▷ Performance of symmetric-multiprocessor systems

# AMDAHL'S LAW



Ratio of vector to scalar speed: 10

**AMDAHL'S LAW (3)**

- Example 1: Speedup from the use of cache memory, assuming that
  - ▷ Cache is 5 times faster than main memory
  - ▷ Cache can be used 90% of the time

Result:

$$\text{Overall speedup} = \frac{1}{(1 - 0.9) + \frac{0.9}{5}} = 3.57$$

- Example 2: Same as example 1, but assuming that cache can be used only 50% of the time:

$$\text{Overall speedup} = \frac{1}{(1 - 0.5) + \frac{0.5}{5}} = 1.67$$

**AMDAHL'S LAW (4)**

## ● Example 3 (Real-world):

At the Los Alamos and Lawrence Livermore National Laboratories, 10 years of work by highly skilled programmers resulted in only 70% vectorization on most of the workload

▷ Vectorized code runs up to 10 times faster than scalar code

▷ Amdahl's Law predicts that the average speedup from vectorization is

$$\text{Overall speedup} = \frac{1}{(1 - 0.7) + \frac{0.7}{10}} = 2.70$$

▷ Result: The “killer micros”, whose scalar speed is equal to or better than the scalar speed of the CRAY vector processors, took over a significant share of CRAY's market at the national laboratories

## BENCHMARKING A PARALLELIZED PROGRAM

- Benchmark results for a program running on one processor:  
6629.0u 5.0s 1:51:21 99% 0+0k 0+0io 0pf+0w
- Benchmark results for the same program running in parallel on four processors (same system):  
7779.0u 9.0s 32:57 393% 0+0k 0+0io 0pf+0w
- Analysis:
  - ▷ One processor: User time = 6629 seconds
  - ▷ Four processors: Sum of user times = 7779 seconds
  - ▷ Sum of times on 4 CPUs = 393% of elapsed time
  - ▷ Speedup = ratio of elapsed times = 3.38
  - ▷ Theoretical maximum speedup = 4
  - ▷ Actual speedup = 84% of theoretical maximum

## PRODUCT DEVELOPMENT (1)

- Performance evaluation as part of the product development process
  - ▷ **Goal: Optimize performance within the cost envelope for your market**
    - Optimize aspects of performance with the greatest effect on sales
    - Understand your market!
      - ◇ Customer contacts
      - ◇ Usenet newsgroups
    - Make an engineering analysis of costs vs. performance for a range of designs

## CONTRIBUTIONS TO PRODUCT COST (1)

- Component costs
- Direct costs – costs directly related to making a product
  - ▷ Labor
  - ▷ Purchasing costs
  - ▷ Warranty service and replacement
  - ▷ Scrap
- Gross margin – indirect costs that cannot be billed to one product
  - ▷ R & D (typically 8% to 15% of revenues)
  - ▷ Marketing & sales
  - ▷ Equipment maintenance, rental, etc.
  - ▷ Financing costs, taxes, fees, ...
  - ▷ Profit

## LESSONS FROM COMPUTER HISTORY

- Be aware of new technology
  - ▷ Opportunities and threats
  - ▷ Current skills  $\Rightarrow$  personal mobility
- Minimize your product's time to market
  - ▷ Miss the window of opportunity, and your job may be history
- Use open or widely-adopted standards
  - ▷ Compatibility with other vendors' products; remember the TI PC!
  - ▷ Lower parts cost
  - ▷ Respect hardware, OS, networking standards
- Niche markets can be highly profitable
  - ▷ CRAY, Apple, Sun, SGI
  - ▷ Vulnerable to new technology