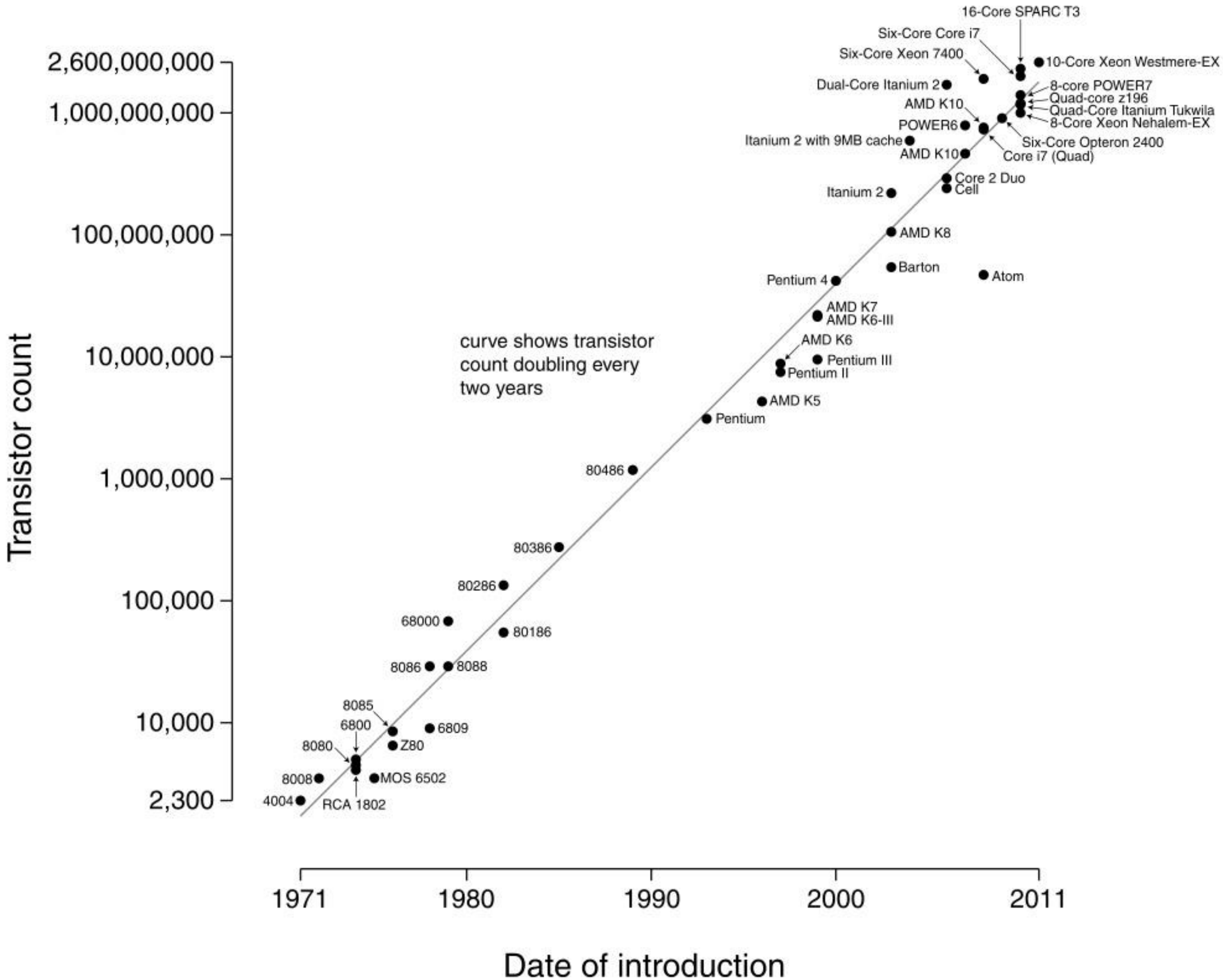
Moore’s law

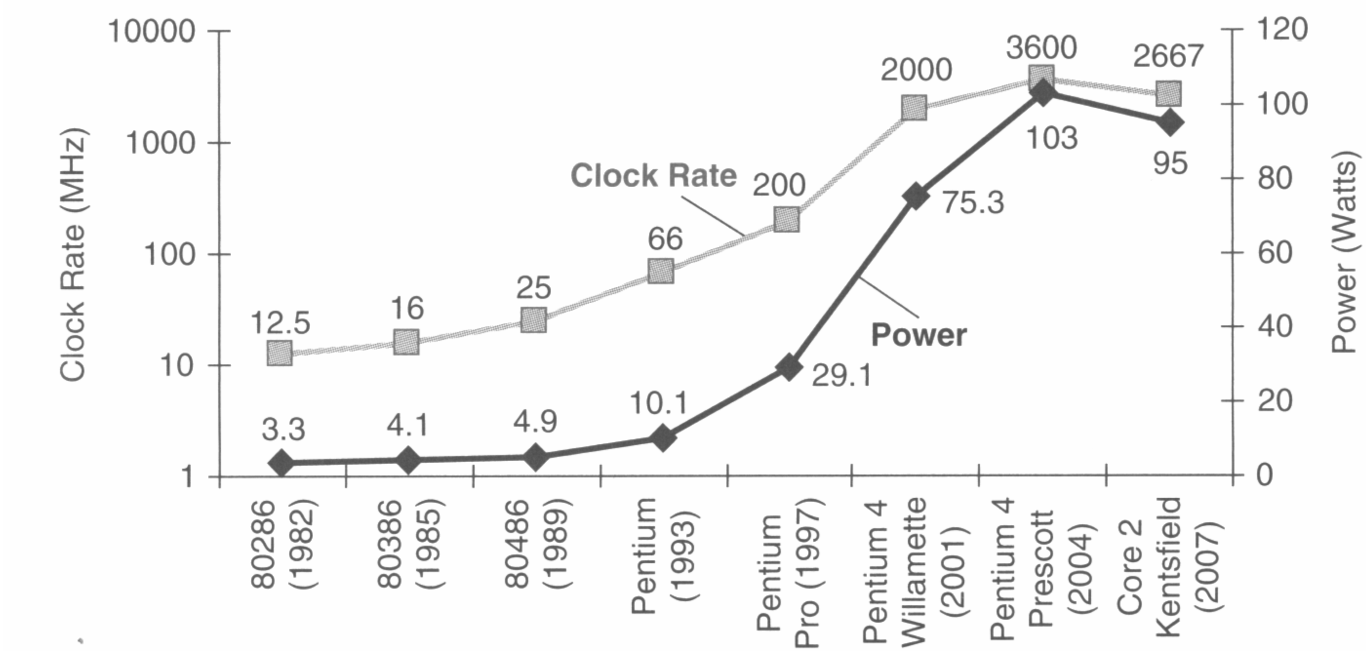
- Describes the rate at which component density increases
  - Number of transistors (per sq.mm) is doubled every two years
  - Is also used by the industry to set the design goals
- Affects performance
  - More transistors → more computing power
  - Requires also architectural changes
    - Designs have their limits and increasing the number of transistors may be inefficient
  - More components can be integrated on a single chip → more reliability

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Power wall

- Power consumption and clock rate are correlated
- Practical power limit for cooling commodity microprocessors prevents using much higher clock rates

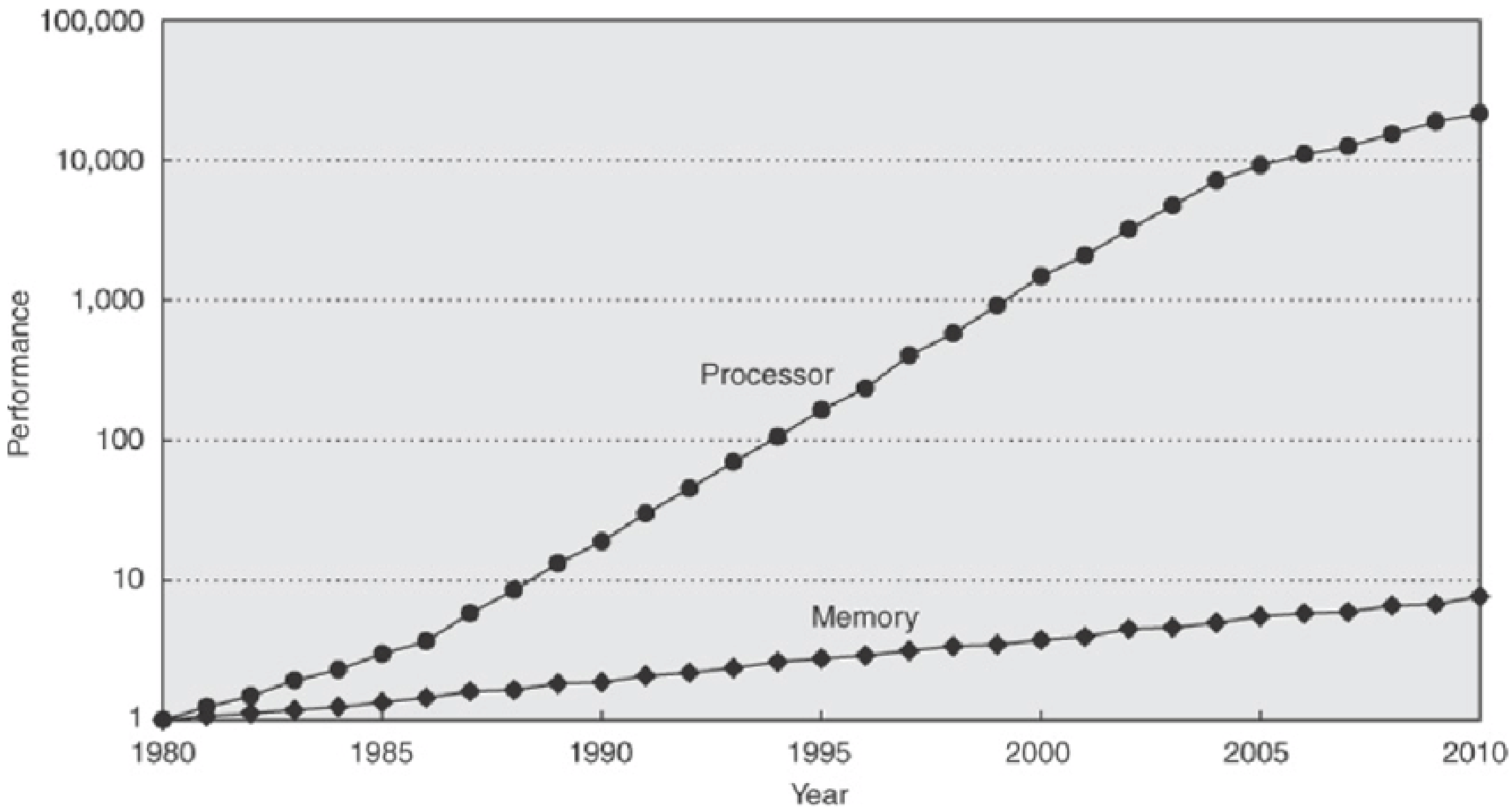


Clock rate and power consumption for Intel x86 microprocessors

Memory wall

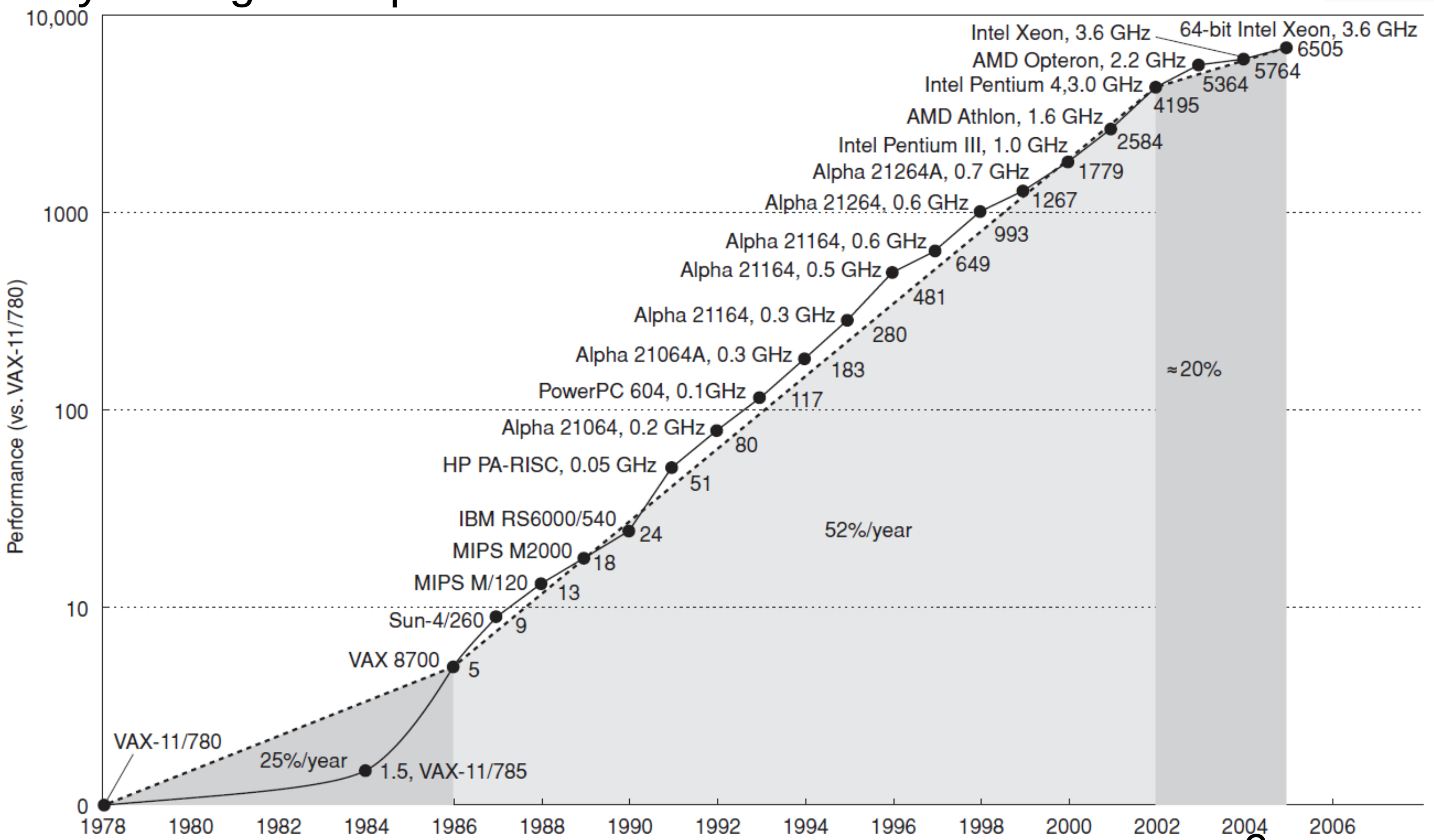
- Processor speed increases rapidly
- Memory storage capacity increases rapidly
- Memory latency decreases only modestly
- Processor is faster than memory → Getting enough instructions and data to processor is difficult
- Caches are used to improve memory access latencies

DRAM and Processor Characteristics



Performance issues

- Limits of power consumption, memory latency and available instruction level parallelism have slowed uniprocessor performance growth → performance is increased by adding more processor cores



Measuring performance

- Traditional measure of performance is MIPS (Millions of Instructions Per Second) or FLOPS (FLoating point Operations Per Second)
  - Does not take into account other parts of computer system
  - Execution time varies depending on the instructions
  - Mixture of used instructions is application dependent → different result with different programs

Measuring performance

- Benchmarks are commonly used to measure performance
- Benchmarks are “standardized” programs or methods for measuring system performance
- Different benchmarks are affected by different components and design choices of a computer system → there is no all purpose benchmark
  - Desktop benchmarks typically divide to two classes
    - Processor intensive
    - Graphics intensive benchmarks
  - Server benchmarks are typically application specific
    - File server (focuses on IO and network performance)
    - Web server
    - Transaction processing

Quantitative principles of computer design

- Take advantage of parallelism
  - System level
    - Add more processors, hard disks etc.
    - Scalability (important for servers)
  - Processor level
    - Instruction level parallelism (overlap execution of instructions)
      - Most instructions are not dependent on its immediate predecessors so partial or completely parallel execution is possible
  - Digital design level
    - For example carry-lookahead adders
- Principle of locality
  - Programs tend to reuse data and instructions they have used recently
- Focus on the common case
  - Improving the performance of most common case yields larger speedup than rare events

Amdahl's law

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used
- Speedup depends on two factors
  - The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement
  - The improvement gained by the enhanced execution mode
    - How much faster the task would run if the enhanced mode were used for the entire program

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Amdahl's law

- Amdahl's law is a general formula which can be applied to wide variety of calculations (instruction set improvements, multiprocessors, reliability, etc.)

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

The overall speedup is the ratio of the execution times:

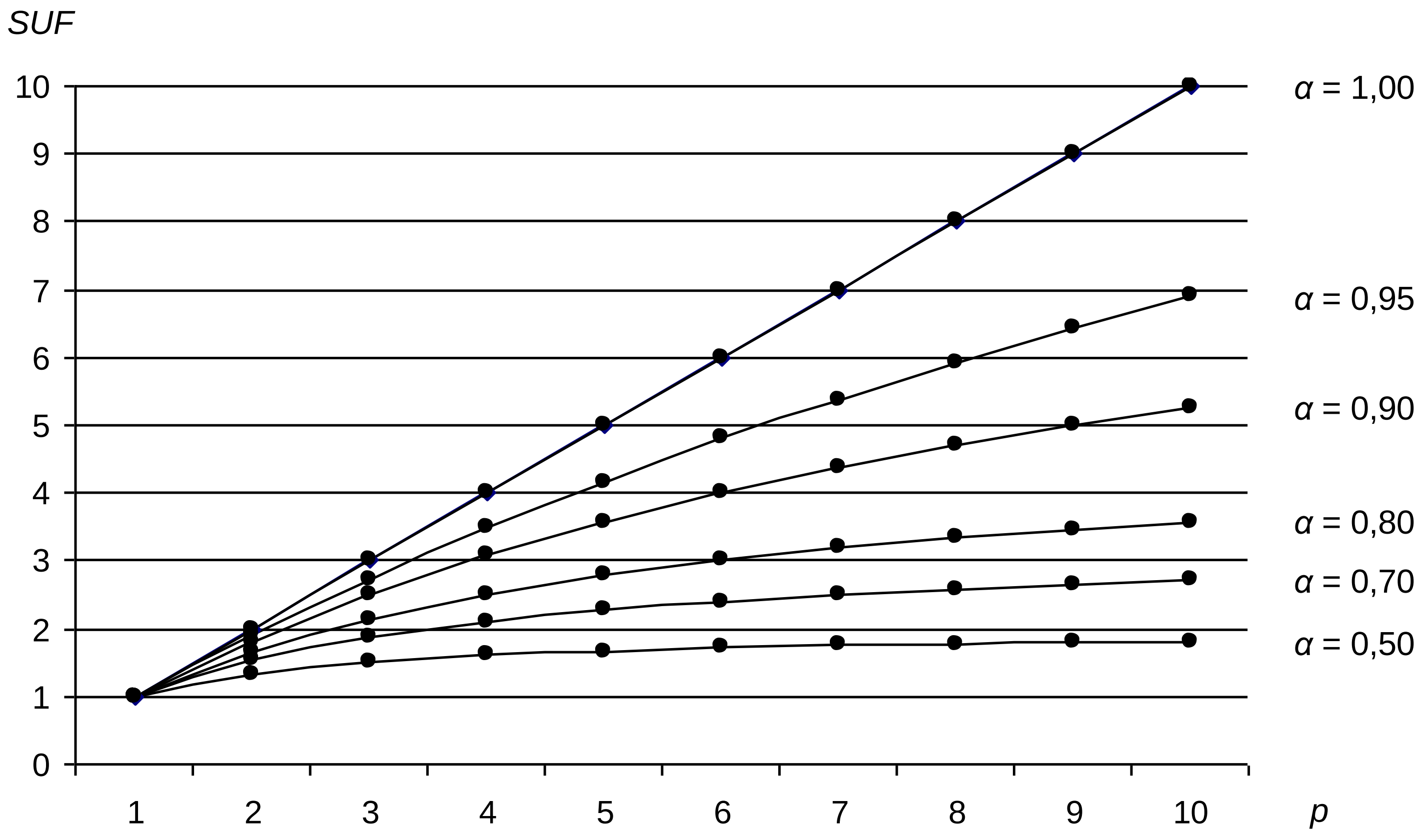
$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example

- Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

$$\text{Fraction}_{\text{enhanced}} = 0.4,\ \text{Speedup}_{\text{enhanced}} = 10,\ \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

Amdahl's law



- α is the fraction of enhanced mode
- p is the speedup of enhanced mode
- SUF is the overall speedup

Amdahl's law

- Amdahl's law was originally introduced as a method to estimate the limits of speedup gained by adding more processor cores
- In (simplified) case of multiprocessors the speedup of enhancement mode (p) is equal to the number of processor cores and fraction to enhance (α) is the portion of your task that can be parallelized
- The serial portion of the task is ($1 - \alpha$) which is the limiting factor of speedup that can be gained
- When p increases to infinity we can see that $1/(1 - \alpha)$ is the limit of speedup that we can gain with an application that has serial part

$$SUF = \frac{1}{(1-\alpha) + \frac{\alpha}{p}}$$

Amdahl's law

- Note that Amdahl's law applies to only to a case of a single task
- Since Amdahl's law does not take into account the fact that you can run multiple different tasks, it gives a pessimistic result on a computer with modern operating system that is serving large number of users and is able to balance CPU loads
- However it very much applies to computing intensive tasks
  - For example algorithms that run on a GPU must be highly parallelizable in order to exploit the full potential of GPU computing power

Processor performance equation

- All computer are constructed using a clock running at constant clock rate

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

or

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- Modern processors include counters for number of clock cycles and number of executed instructions which helps software optimization

Processor performance equation

- The average number of clock cycles per instruction provides insight into different styles of instruction sets and implementations

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

- Pipeline effects, cache misses and other memory inefficiencies also contribute to CPI value
  - CPI has to be measured – not calculated using instruction set reference manual

Cortex-M3 debug counters

8.2 DWT Programmers' model

Table showing the DWT registers. Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero.

Table 8-1 DWT register summary

Address	Name	Type	Reset	Description
`0xE0001000`	DWT_CTRL	RW	Possible reset values are: • 0x40000000 if four comparators for watchpoints and triggers are present. • 0x4F000000 if four comparators for watchpoints only are present. • 0x10000000 if only one comparator is present. • 0x1F000000 if one comparator for watchpoints and not triggers is present. • 0x00000000 if DWT is not present.	Control Register.
`0xE0001004`	DWT_CYCCNT	RW	`0x00000000`	Cycle Count Register
`0xE0001008`	DWT_CPICNT	RW	-	CPI Count Register
`0xE000100C`	DWT_EXCCNT	RW	-	Exception Overhead Count Register
`0xE0001010`	DWT_SLEEPCNT	RW	-	Sleep Count Register
`0xE0001014`	DWT_LSUCNT	RW	-	LSU Count Register
`0xE0001018`	DWT_FOLDCNT	RW	-	Folded-instruction Count Register
`0xE000101C`	DWT_PCSR	RO	-	Program Counter Sample Register
`0xE0001020`	DWT_COMP0	RW	-	Comparator Register0
`0xE0001024`			-	Mask Register0

Processor performance equation

- CPU time is equally dependent on these three characteristics:
  - A 10% improvement in any one of them leads to a 10% improvement in CPU time.

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

Processor performance equation

- Clock cycle time
  - Hardware technology and organization
- CPI
  - Organization and instruction set architecture
- Instruction count
  - Instruction set architecture and compiler technology

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle