CS636: Systems Basics

Swarnendu Biswas

Semester 2018-2019-II

CSE, IIT Kanpur

Content influenced by many excellent references, see References slide for acknowledgements.

Two things primary in Systems:

- Performance
- Power

What is Performance?

Execution time = Time (s) taken by a program to execute

Measure Performance

Execution time = Time (s) taken by a program to execute

- Execution time – time from start till end of the computation
- Speedup - $\frac{T_1}{T_p}$

What is Performance?

Execution time = Time (s) taken by a program to execute

$$Exec\ time = Time\ to\ execute\ \#\ instrs\ in\ the\ program$$

What is Performance?

Execution time = Time (s) taken by a program to execute

$$Exec\ time = Time\ to\ execute\ \#\ instrs\ in\ the\ program$$

$$Exec\ time = \frac{\#\ instrs}{program} * Time\ to\ execute\ 1\ instr$$

What is Performance?

Execution time = Time (s) taken by a program to execute

$$Exec\ time = \frac{\#\ instrs}{program} * Time\ to\ execute\ 1\ instr$$

$$Exec\ time = \frac{\#\ instrs}{program} * \frac{\#\ cycles}{instr} * Time\ to\ execute\ 1\ cycle$$

What is Performance?

Execution time = Time (s) taken by a program to execute

$$Exec\ time = \frac{\#\ instrs}{program} * \frac{\#\ cycles}{instr} * Time\ to\ execute\ 1\ cycle$$

$$Exec\ time = \frac{\#\ instrs}{program} * \frac{\#\ cycles}{instr} * \frac{\text{time (s)}}{\text{cycle}}$$

What is Performance?

Execution time = Time (s) taken by a program to execute

$$Exec\ time = \frac{\#\ instrs}{program} * CPI * \frac{1}{freq}$$

What is Performance?

Ex

E

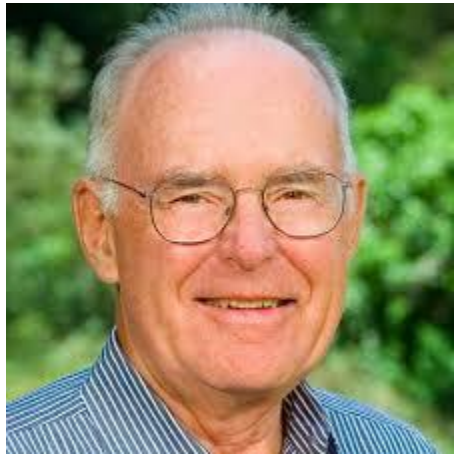
Performance $\propto frequency$

Performance $\propto \frac{1}{CPI}$

Buying Performance with Technological Innovations

- 1986 – 2005
  - Performance of microprocessors increased by ~50% per year
- Programs ran faster by themselves
  - We did not worry about performance
- Parallel computing, concurrent programming, and HPC were jobs for specialists

Moore’s Law



Any volunteers?

Moore's Law

- Number of transistors on chip doubles every year
  - 1965
  - Recalibrated it later in 70's to say "doubles every two years"

- David House from Intel said "improvements would cause performance to double every 18 months"

Moore's Law

- Number of transistors on chip doubles every year
  - 1965
  - Recalibrated it later in 70's to say "doubles every two years"

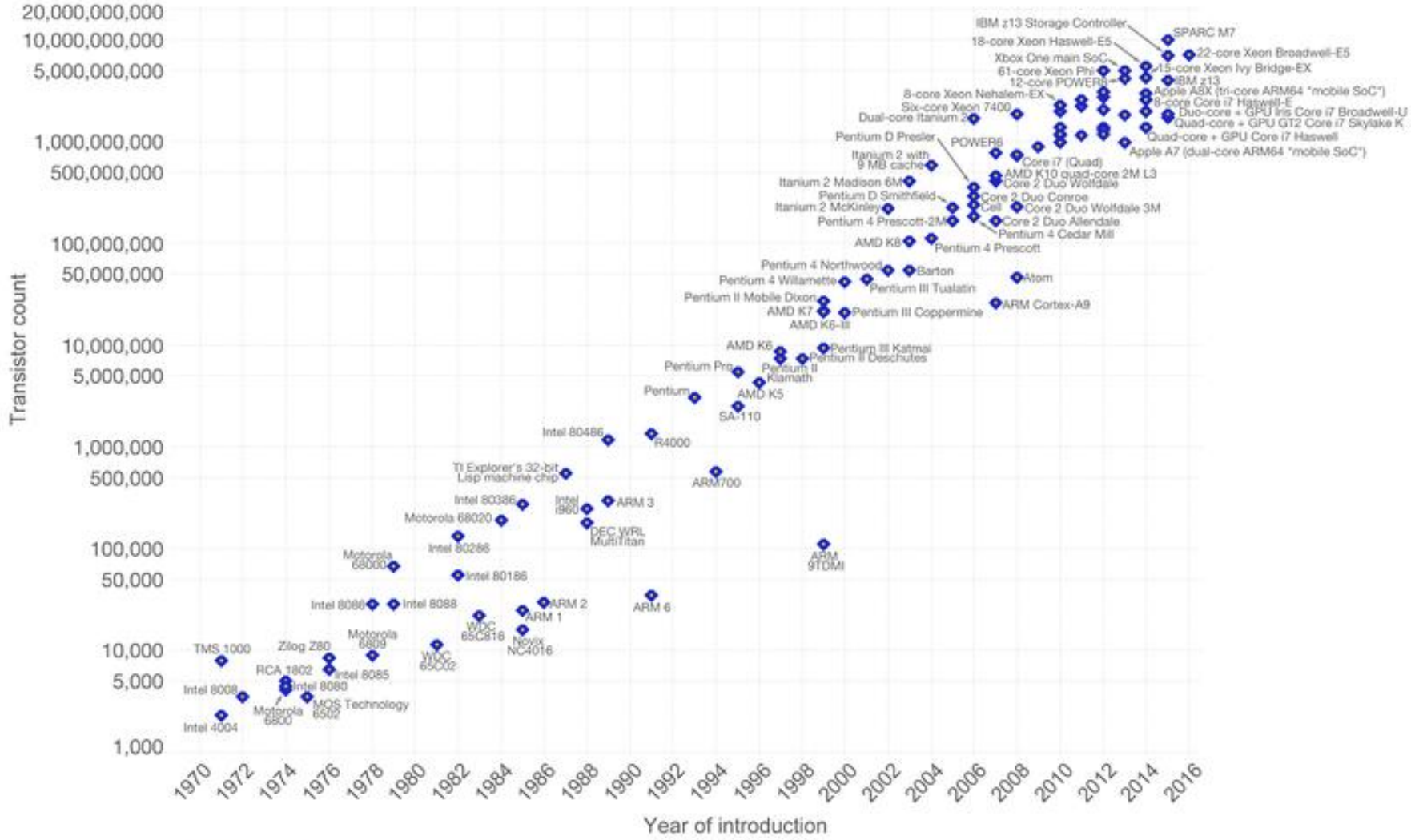
- David House from Intel said "improvements would cause performance to double every 18 months"

"Moore's law is a violation of Murphy's law."

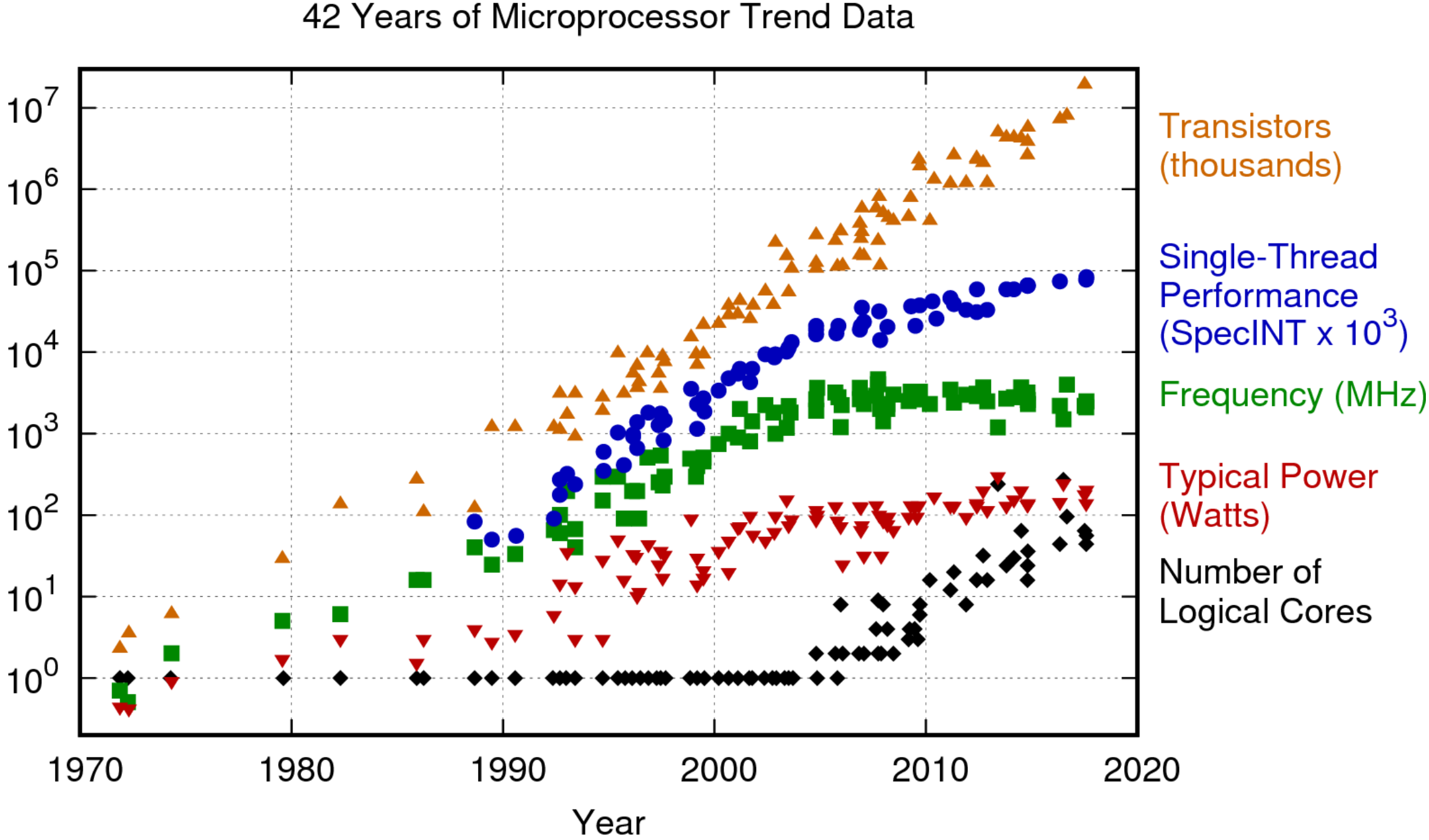
Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

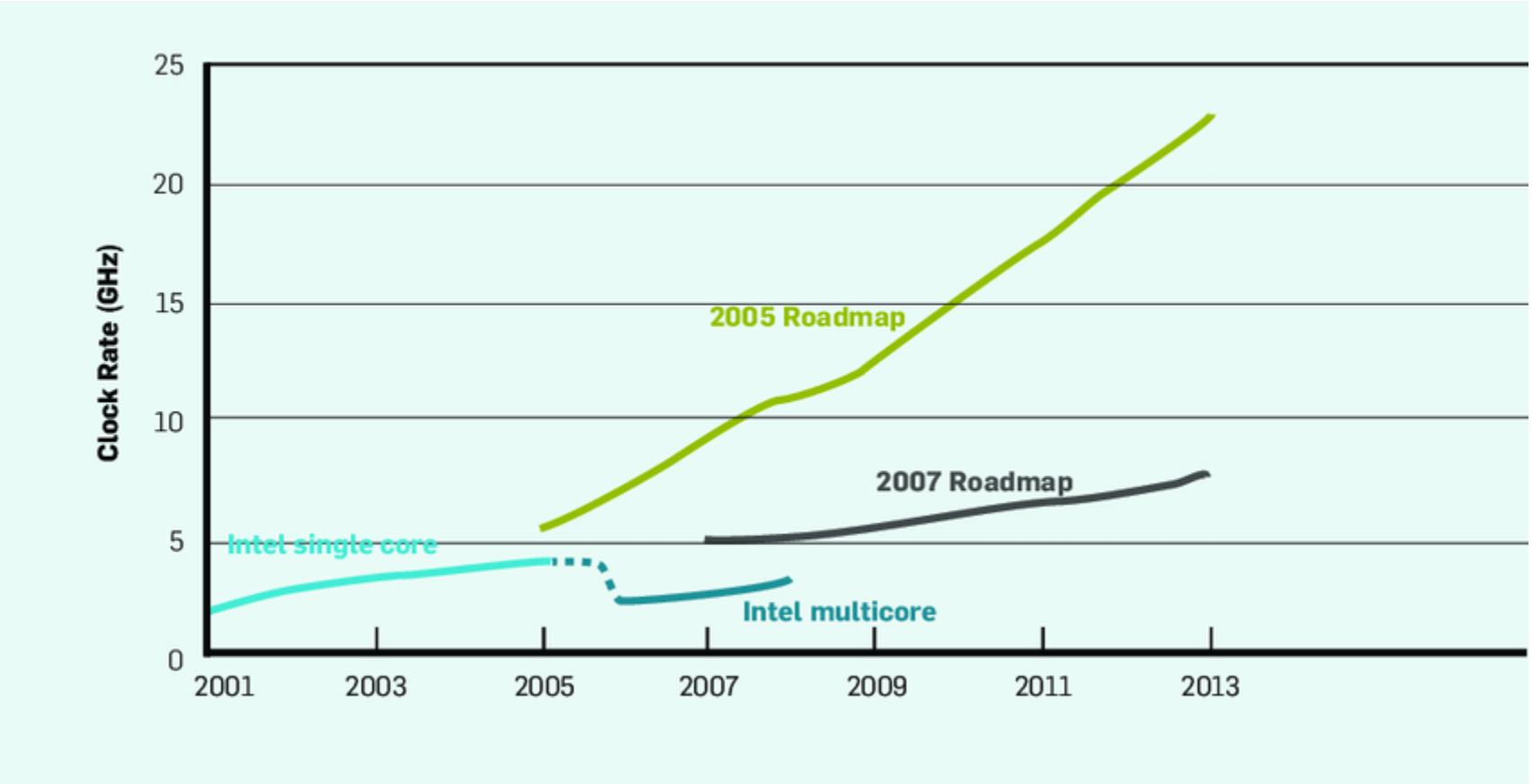


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Challenges to Growth in Performance

Clock speeds are not increasing any more

Clock speeds are stagnating!

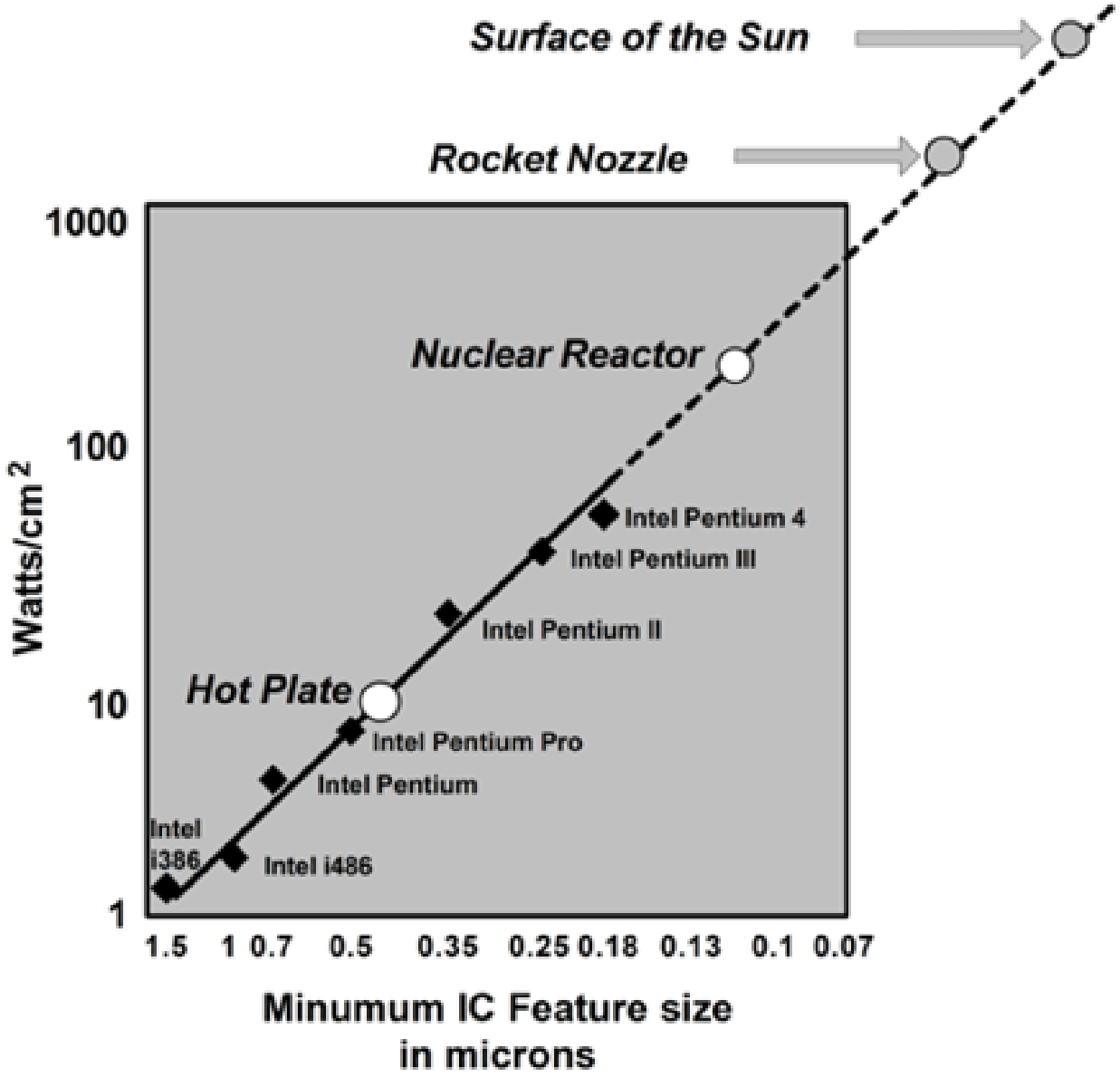


K. Asanovic et al. A View of the Parallel Computing Laandscape. CACM, Oct 2009.

Power Wall

- Power, and not manufacturing, limits microarchitectural improvements – F. Pollack

Dynamic power = Capacitive load x Voltage x Frequency



Mark Smotherman. https://people.cs.clemson.edu/~mark/330/power_density.gif

Reducing Power

- Suppose a new CPU version has
  - 90% capacitive load of the old CPU version
  - 10% reduction in voltage
  - 10% reduction in frequency
  - What is the impact to the power consumption of the new CPU version?

Other Challenges Going Forward!

- Reliability challenges with smaller processes
  - ~7nm?
  - More interference, structural defects with Process-Voltage-Temperature (PVT) variations

Hardware Trends in the Last Ten Years!

- 2005 – 2018
  - Single core performance increase is ~20%
- Programs do not run any faster by itself

Microarchitectural Techniques

- Add more functional units to improve ILP
  - Superscalar architecture
  - VLIW
  - More cache structures (e.g., L4 caches)
  - Deeper pipelines

Microarchitectural Techniques

- Add more functional units to improve ILP
  - Superscalar architecture
  - VLIW
  - More cache s
  - Deeper pipel

Law of diminishing returns!

Multicore Architecture

- Make effective use of the extra transistors
- New prediction: # cores will double every two years

- We also have manycore machines

What is the software side of the story?

Develop Parallel Programs

From my perspective, parallelism is the biggest challenge since high-level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful.
…
Industry is building parallel hardware, assuming people can use it. And I think there's a chance they'll fail since the software is not necessarily in place. So this is a gigantic challenge facing the computer science community.

– David Patterson, ACM Queue, 2006.

Develop Parallel Programs

To save the IT industry, researchers must demonstrate greater end-user value of from an increasing number of cores –
A View of Parallel Computing Landscape, CACM 2009.

New Challenges in Software Development

- Adapt to the changing hardware landscape
- Most applications are single-threaded

How can we develop software that makes effective use of the extra hardware?

Challenges in Developing Parallel Programs

- Programmers tend to **think sequentially**
  - Correctness issues – concurrency bugs like data races and deadlocks
  - Performance issues – minimize communication across cores
- Amdahl's law
- Overheads of parallel execution
- Other challenges: load balancing

Programmer's tend to think sequentially

Dekker's Algorithm

TABLE 3.3: Can Both r1 and r2 be Set to 0?

Core C1	Core C2	Comments
S1: x = NEW; L1: r1 = y;	S2: y = NEW; L2: r2 = x;	/* Initially, x = 0 & y = 0*/

A Java Snippet

```
Object X = null;
boolean done= false;
```

Thread T1

```
X = new Object();
done = true;
```

Thread T2

```
while (!done) {}
X.compute();
```

A Java Snippet

```
Object X = null;
boolean done= false;
```

Thread T1

```
X = new Object();
done = true;
```

Thread T2

```
while (!done) {}
X.compute();
```

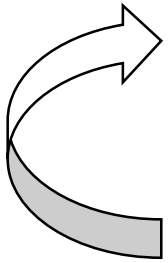
What are some possible outcomes?

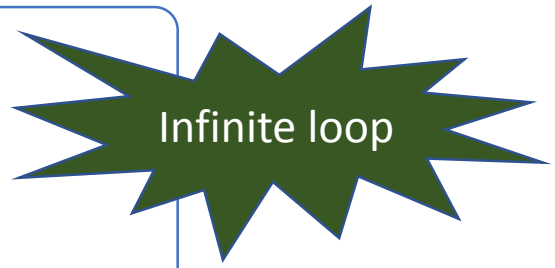
Thread T1

```
X = new Object();


done = true;
```

Thread T2

```
temp = done;

while (!temp) {}
```



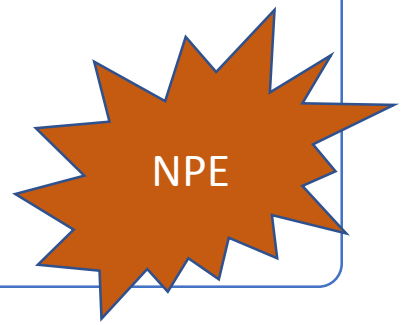


Thread T1

```
done = true;


X = new Object();
```

Thread T2

```
while (!done) {}
X.compute();
```



We will see more of this later!

Amdahl's Law

- Intuitive observation

Any volunteers?

Amdahl's Law

- Intuitive observation

Even if most of the program can be parallelized, the benefits of parallel execution are limited

Example of Amdahl's law

- Suppose you are traveling from Kanpur to Lucknow.
  - You travel from Kanpur to Lucknow at 50 kmph.
  - You travel back from Lucknow to Kanpur at the speed of light.
  - What is your average speed?

Amdahl's Law Formulation

- Let a program P have N operations
- **Assumption:** An operation (executed serially) takes one time unit
- Time taken to execute P: N

- Let proportion P of the program be parallelizable
- Time taken for the serial portion = $(1 - P)N$
- Assume parallel portion can be accelerated by a factor of s
- Time taken by optimized implementation: $(1 - P)N + \frac{PN}{s}$

Amdahl's Law Formulation

- Speedup factor: $= \frac{Execution\ time\ of\ original\ program}{Execution\ time\ of\ optimized\ program}$

$$= \frac{N}{(I-P)N + \frac{P}{s}}$$

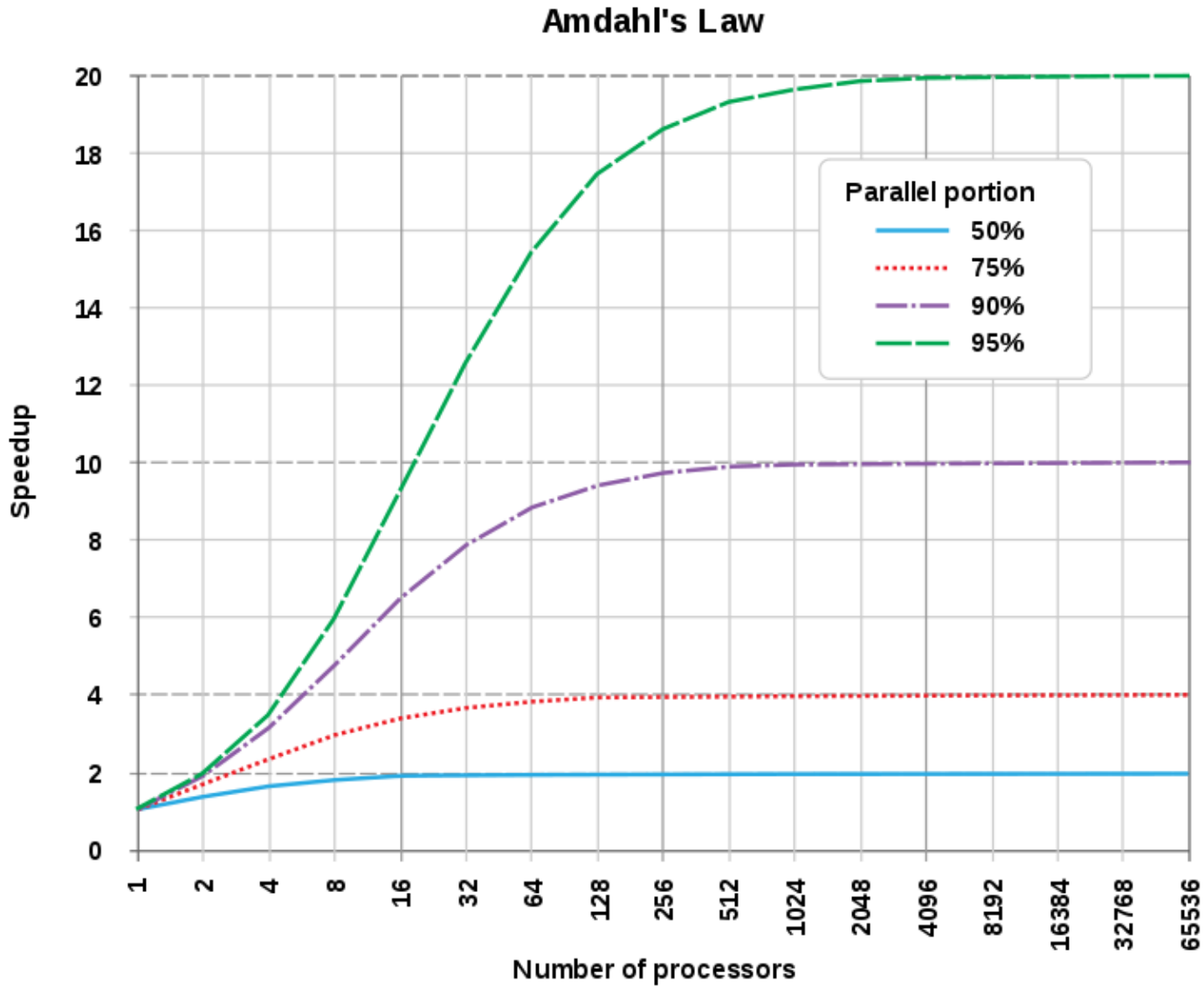
$$= \frac{1}{(1-P) + \frac{P}{s}}$$

Nuances of Amdahl's Law

$$\text{Speed up} = \frac{1}{(1-P)+\frac{P}{s}}$$

Even if speed up factor s → ∞

$$\text{Speed up} = \frac{1}{1-P}$$



https://en.wikipedia.org/wiki/Amdahl%27s_law

Compute speed up

- Suppose we have a program P that is composed of three modules A, B, and C. A takes up 20%, B takes 30%, and C takes 50%.
- Suppose we run P on new hardware that speeds up A by 50% and B by 4X but does not impact C.
- What is the overall speedup of P?

Gustafson's Law

- Amdahl's Law assumes that the problem size **does not** change with number of resources

- Gustafson's Law
  - **Computation time is constant instead of problem size**
  - Increase resources to solve bigger computational problems in the same time

System Assumptions (same as Amdahl's Law derivation)

- Let us assume a program with N operations
- **Assumption:** An operation (executed serially) takes one time unit
- Time taken to execute the program: N

- Let proportion P of the program be parallelizable
- Assume parallel portion can be accelerated by a factor of s

Gustafson's Law

- Now that we have more resources, the execution time will hopefully decrease
- But the goal is to do more work in the original execution time

- Original problem size (Amdahl's law): $PN + (1 - P)N$
- New problem size: $PsN + (1 - P)N$
- Execution time on single processor: $PsN + (1 - P)N$
- Execution time on multiprocessor: $\frac{PsN}{s} + (1 - P)N$

Gustafson's Law

- Original problem size (Amdahl's law): $PN + (1 - P)N$
- New problem size: $PsN + (1 - P)N$
- Execution time on single processor: $PsN + (1 - P)N$
- Execution time on multiprocessor: $\frac{PsN}{s} + (1 - P)N$
- Speedup: $\frac{PsN + (1-P)N}{\frac{PsN}{s} + (1-P)N} = \frac{Ps + (1-P)}{P + (1-P)} = Ps + (1 - P)$

Amdahl's vs Gustafson's Law

Amdahl's law – If you have 10 more CPUs, then how fast can you solve a given problem?

Gustafson's law – Having 10 more CPUs allows you to solve a larger problem in the same amount of time.

References

- Keshav Pingali – CS 377P: Programming for Performance, UT Austin.
- D. Sorin et al. – A Primer on Memory Consistency and Cache Coherence