Performance of Parallel Programs

Michelle Kuttel

Analyzing algorithms

- Like all algorithms, parallel algorithms should be:
  - Correct
  - Efficient

- First we will talk about efficiency

Work and Span

Let $\mathbf{T_P}$ be the running time if there are **P** processors available

Two key measures of run-time:

- Work: How long it would take 1 processor = $\mathbf{T_1}$

  e.g. for divide-and-conquer, just "sequentialize" the recursive forking

- Span: How long it would take infinity processors = $\mathbf{T_\infty}$
  - The longest dependence-chain
  - Example: $O($`log` $n)$ for summing an array with divide-and-conquer method
    - Notice that for this case, having > $n/2$ processors is no additional help
  - Also called "critical path length" or "computational depth"

Speedup

Speedup is the factor by which the time is reduced compared to a single processor

Speedup for *P* processes =

$$\frac{\text{time for 1 process}}{\text{time for } P \text{ processes}}$$

$= T_1/T_P$.

In the ideal situation, as P increases, so T_P should decrease by a factor of P.

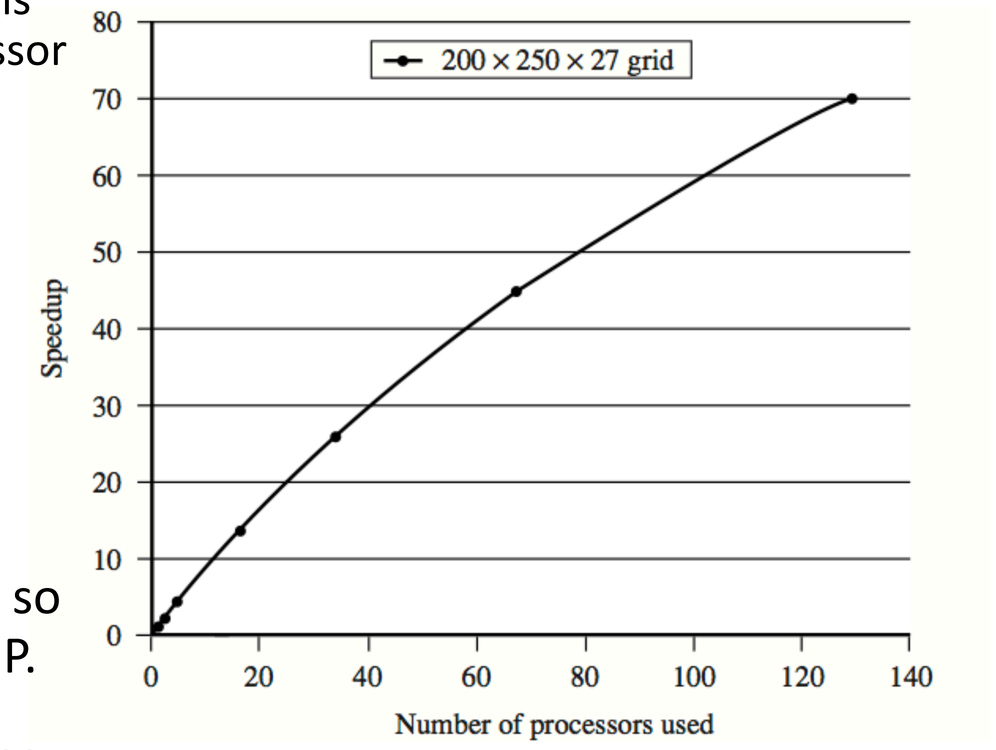


Figure 1.1 Performance of the MM5 weather code.

Figure from "Parallel Programming in OpenMP, by Chandra et al.

Scalability

- **Scalability** is the speed-up of a program as the number of processors being used increases.
  - perfect linear speedup = P
  - Perfect linear speed-up means *doubling* **P** *halves* running time
  - Usually our goal; hard to get in practice
- an algorithm is termed **scalable** if the level of parallelism increases at **least linearly** with the problem size.
  - running time is inversely proportional to the number of processors used.
- In practice, few algorithms achieve linear scalability; most reach an optimal level of performance and then deteriorate, sometimes very rapidly.

Parallelism

Parallelism is the maximum possible speed-up:

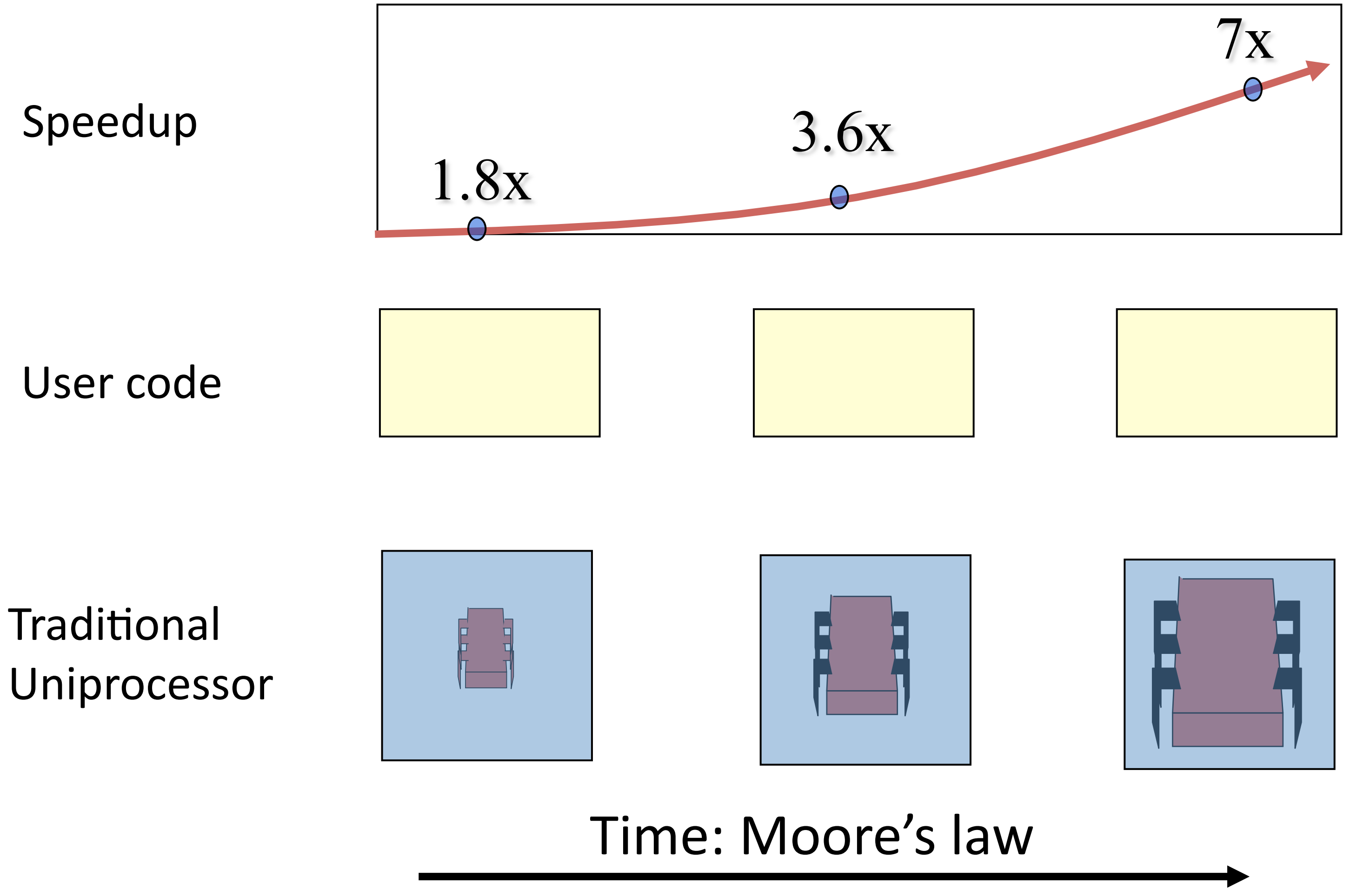
T_1 / T_∞

- At some point, adding processors won't help
- What that point is depends on the span

Parallel algorithms is about decreasing span without

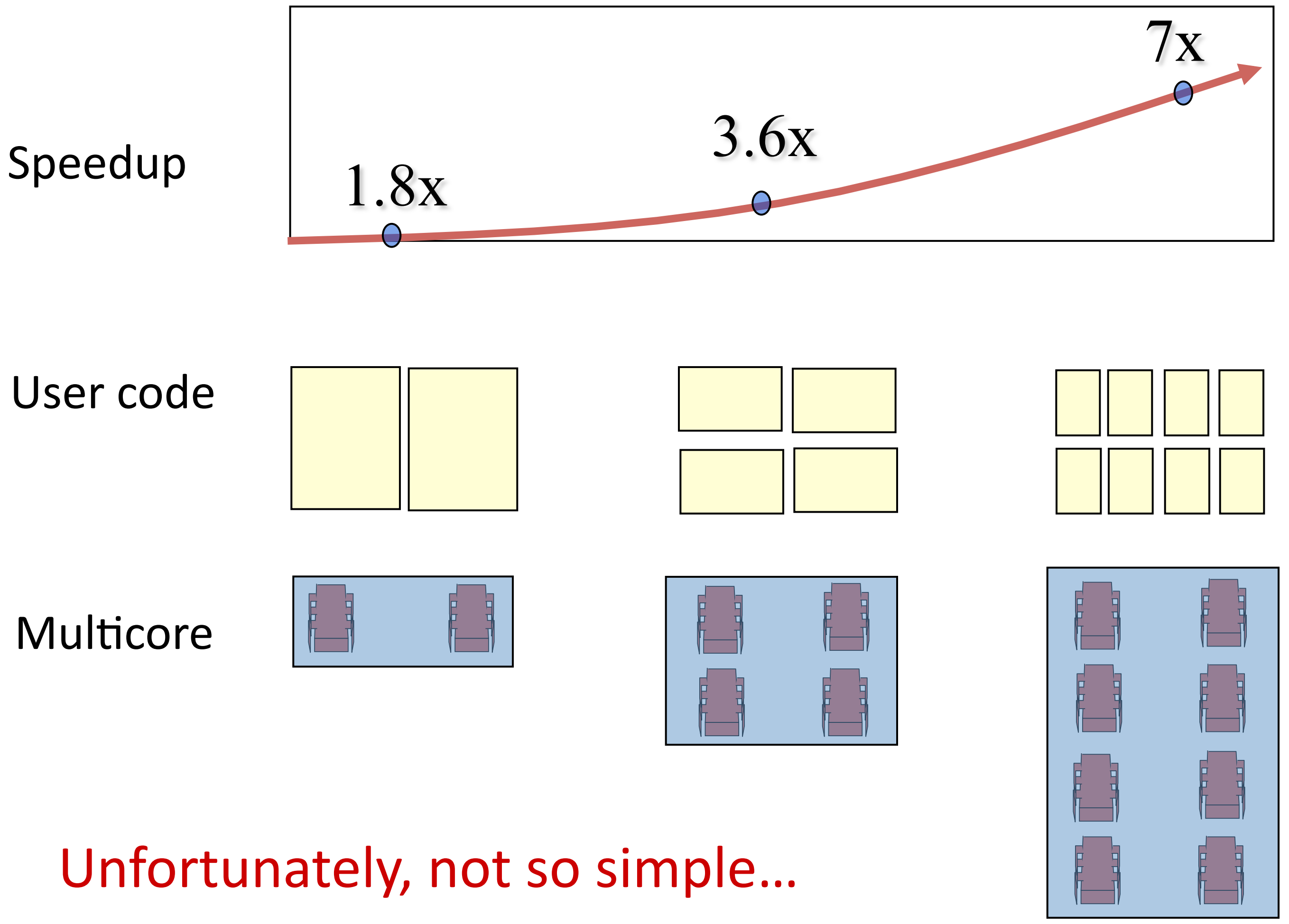
increasing work too much

Traditional Scaling Process



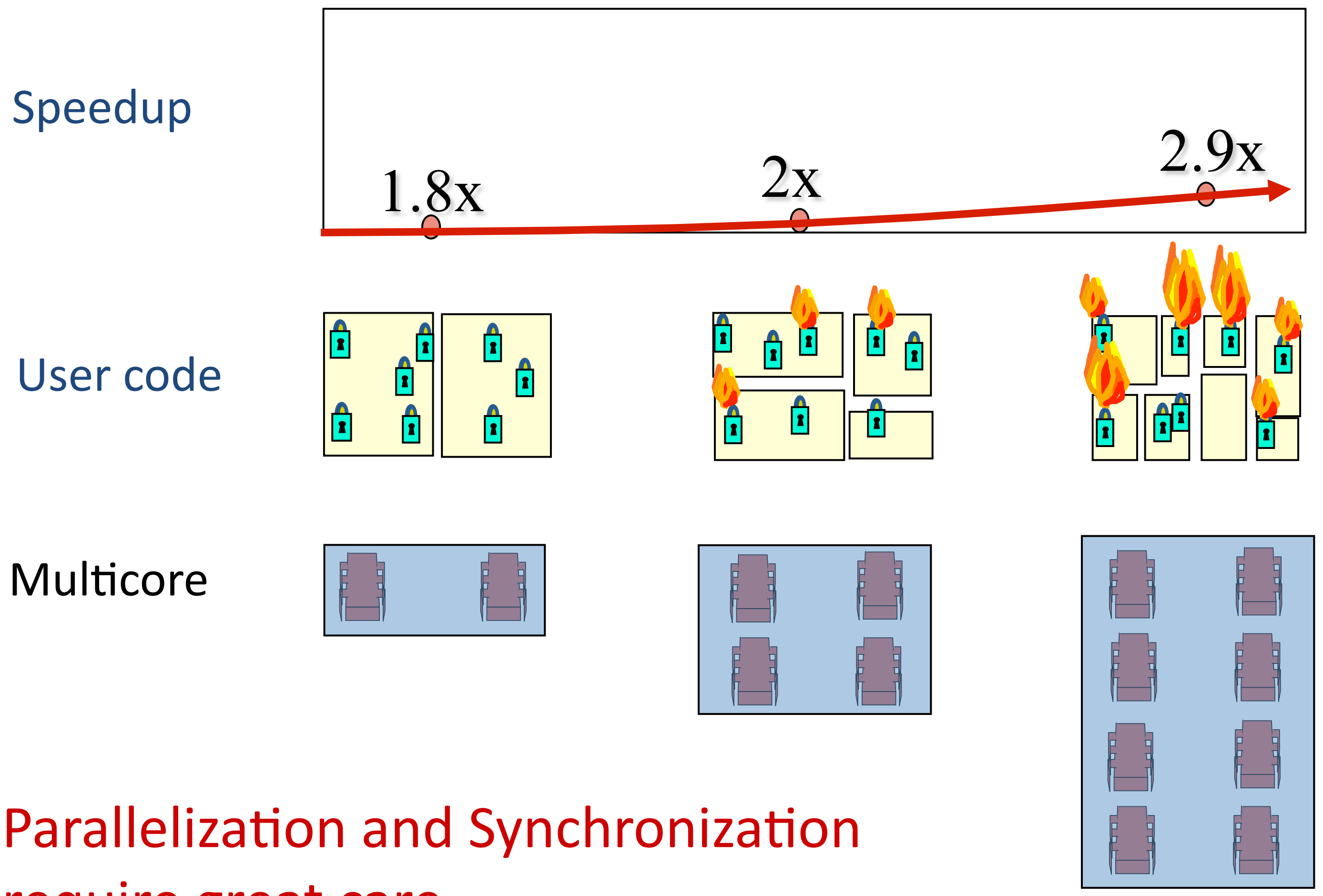
slide from: Art of Multiprocessor Programming

Multicore Scaling Process



Unfortunately, not so simple…

Real-World Scaling Process



Parallelization and Synchronization require great care…

Early Parallel Computing

Why was it not pursued more thoroughly before 1990's?

- Because of dramatic increase in uniprocessor speed, the need for parallelism turned out to be less than expected
- and …

Amdahl’s law

For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit co-operative solution...The nature of this overhead (in parallelism) appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor...At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome.

Gene Amdahl,1967

IBM - the designer of IBM 360 series of mainframe architecture

Amdahl’s Law (mostly bad news)

So far: analyze parallel programs in terms of work and span

- In practice, typically have parts of programs that parallelize well…
  - e.g. maps/reductions over arrays and trees

…and parts that are inherently sequential
  - e.g. reading a linked list, getting input, doing computations where each needs the previous step, etc.

“Nine women can’t make one baby in one month”

We also have the **parallelization overhead**

Refers to the amount of time required to coordinate parallel tasks, as opposed to doing useful work. e.g.

- starting threads
- stopping threads
- synchronization and locks

Amdahl's law*

Total running time for a program can be divided into two parts:

Serial part (s)

Parallel part (p)

$$T_1=s+p$$

For n processors:

$$T_n= s+p/n$$

If set $T_1=1$, then s= 1-p

and

$$T_n= 1-p+p/n$$

* G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", *AFIPS Proc. Of the SJCC,* **30**,438-485,1967

Amdahl’s Law

Speedup= $$\frac{1}{1-p+\frac{p}{n}}$$

Amdahl's Law

Parallel fraction

$$\text{Speedup}=\frac{1}{1-p+\frac{\boxed{p}}{n}}$$

Amdahl's Law

Sequential fraction

Parallel fraction

$$\text{Speedup}=\frac{1}{\boxed{1-p}+\frac{\boxed{p}}{n}}$$

Amdahl's Law

Sequential fraction

Parallel fraction

$$\text{Speedup}=\frac{1}{\boxed{1-p}+\frac{\boxed{p}}{\boxed{n}}}$$

Number of processors

Amdahl’s law*

parallelism (infinite processors) = 1/s

* G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities”, *AFIPS Proc. Of the SJCC,* **30**,438-485,1967

Example

- Ten processors
- 60% concurrent, 40% sequential
- How close to 10-fold speedup?

Example

- Ten processors
- 80% concurrent, 20% sequential
- How close to 10-fold speedup?

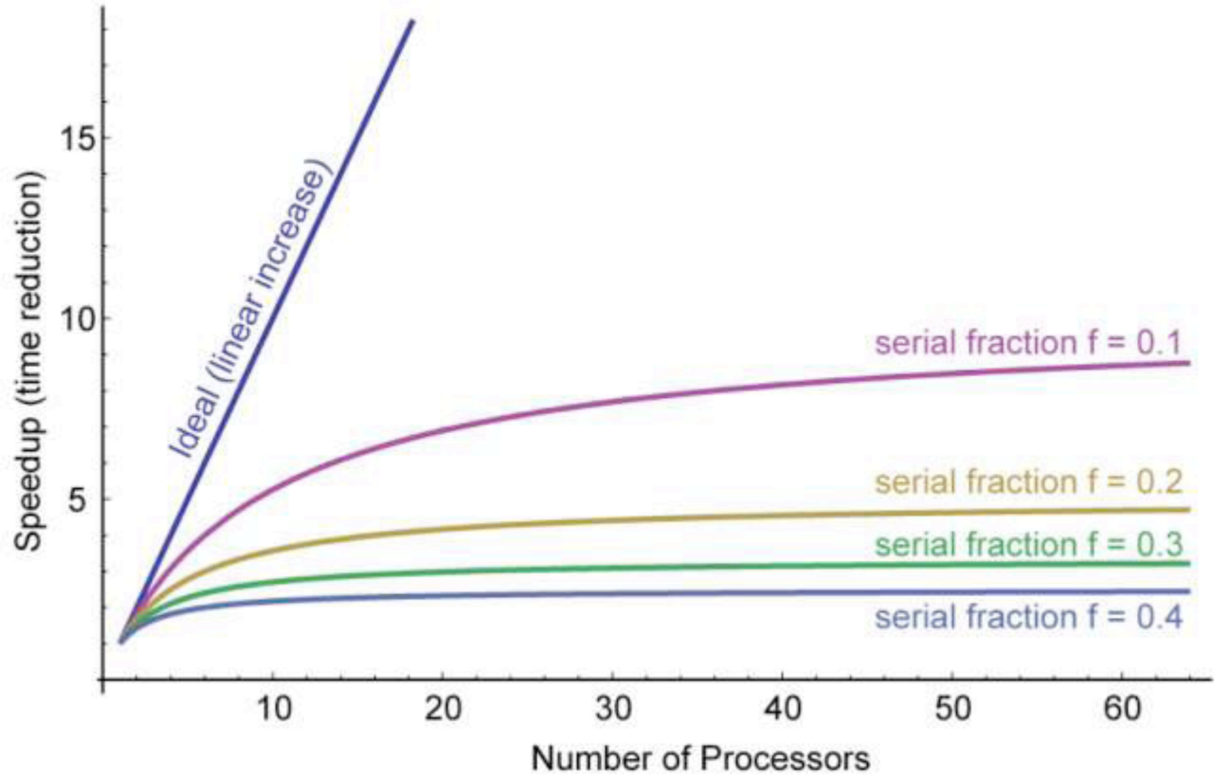
Example

- Ten processors
- 90% concurrent, 10% sequential
- How close to 10-fold speedup?

Example

- Ten processors
- 99% concurrent, 01% sequential
- How close to 10-fold speedup?

Graphing Amdahl's Law



graphic from lecture slides: Defining Computer "Speed": An Unsolved Challenge, Dr. John L. Gustafson, Director Intel Labs, 30 Jan 2011

Why such bad news

$T_1 / T_P = 1 / (S + (1-S)/P)$ $T_1 / T_\infty = 1 / S$

- Suppose 33% of a program is sequential
  - Then a billion processors won't give a speedup over 3
- Suppose you miss the good old days (1980-2005) where 12ish years was long enough to get 100x speedup
  - Now suppose in 12 years, clock speed is the same but you get 256 processors instead of 1
  - For 256 processors to get at least 100x speedup, we need

    $100 \leq 1 / (\mathbf{S} + (1-\mathbf{S})/256)$

    Which means $\mathbf{S} \leq .0061$ (i.e., 99.4% perfectly parallelizable)

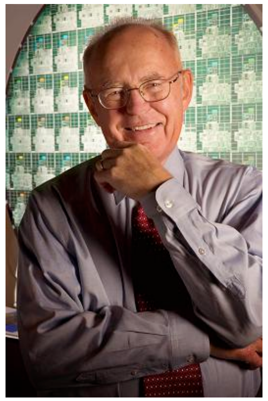
Amdahl’s law*

- This indicated that parallel processing has no future because it is not possible to speedup an application indefinitely by using more and more processors.
- Amdahl argued that for typical Fortran codes, $T_s = 40\%$ and that you therefore cannot use more than **a small number** of processors efficiently.

Amdahl’s Law

- It is now accepted that Amdahl was probably quite correct from a historical point of view.
- In the late 1960’s problem sizes were too small for parallel computing and, had it existed, it would not have been used efficiently.

Moore and Amdahl



- Moore's "Law" is an observation about the progress of the semiconductor industry
  - Transistor density doubles roughly every 18 months
- Amdahl's Law is a mathematical theorem
  - Diminishing returns of adding more processors
- Both are incredibly important in designing computer systems

Ahmadl’s law is based on relative speed

- Compute *relative speed by taking the* ratio of times

Speed1 = work1 / time1.

Speed2 = work1 / time2.

Speedup = Speed1 / Speed2 = time2 / time1.

Sounds reasonable, right?
Yet, this approach held back
parallel processing for 20 years!

From lecture slides: Defining Computer “Speed”: An Unsolved Challenge
Dr. John L. Gustafson, Director
Intel Labs, 30 Jan 2011

The absurdity of basing parallel performance on time reduction:

Ambrose Bierce, from ***The Devil's Dictionary (1911):***

Logic, ***n. The art of thinking and reasoning in strict accordance with the limitations and incapacities of the*** human misunderstanding. The basis of logic is the syllogism, consisting of a major and a minor premise and a conclusion--thus:

Major Premise: Sixty men can do a piece of work sixty times at quickly as one man.

Minor Premise: One man can dig a post-hole in sixty seconds.

Conclusion: Sixty men can dig a post-hole in one second.

This may be called the syllogism arithmetical, in which, by combining logic and mathematics, we obtain a double certainty and are twice blessed.

From lecture slides: Defining Computer "Speed": An Unsolved Challenge, Dr. John L. Gustafson, Director Intel Labs, 30 Jan 2011

Flaws in Amdahl’s law: Gustafson’s Law

- Amdahl’s law has a assumption that may not hold true:
  - the ratio of T_S to T_P is not constant for the same program and usually varies with problem size.
  - *Typically, the T_P grows faster that T_S.*

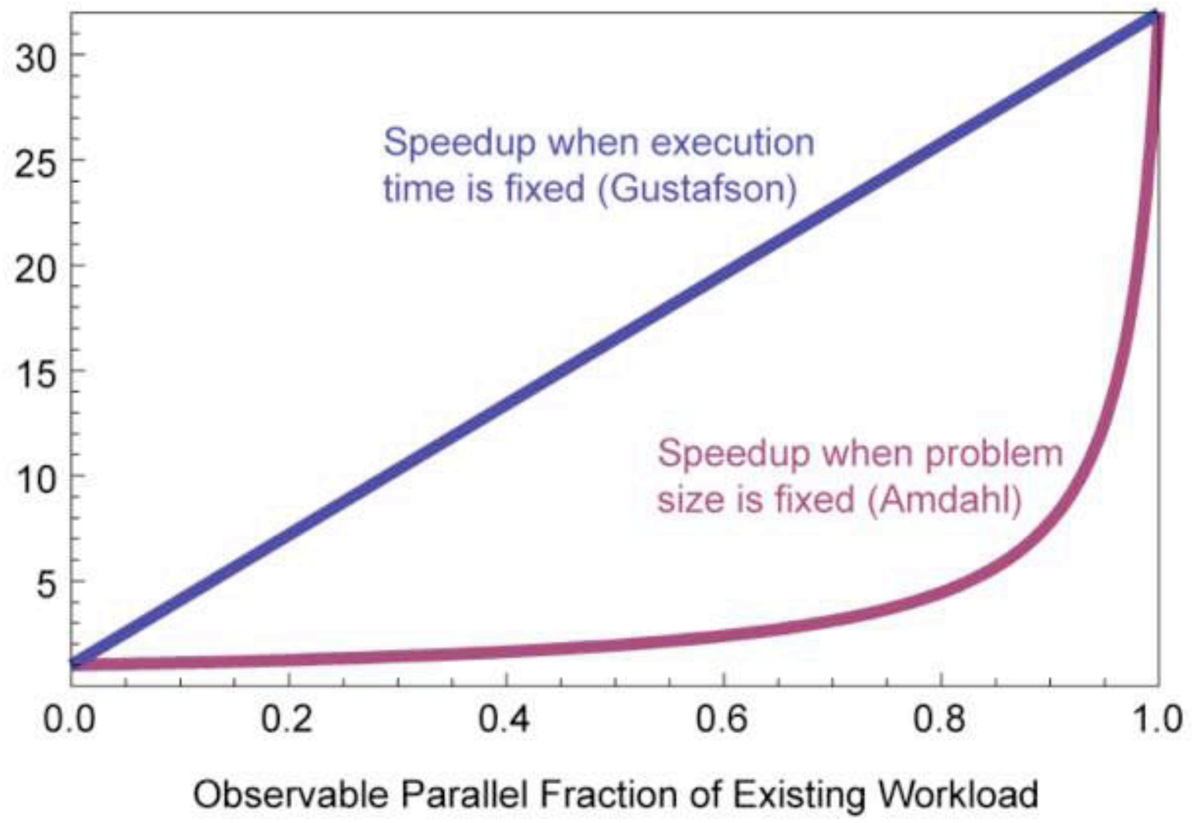
  i.e. the computational problem grows in size through the growth of parallel components – *Amdahl effect*

- Also, the serial algorithm may not be the best solution for a problem

Scalability

- **strong scaling**:
  - defined as how the solution time varies with the number of processors for a fixed *total* problem size.
- **weak scaling:**
  - defined as how the solution time varies with the number of processors for a fixed problem size *per processor*.

- What if *t1 = t2, and* ***work*** *is what changes?*



From lecture slides: Defining Computer "Speed": An Unsolved Challenge, Dr. John L. Gustafson, Director Intel Labs, 30 Jan 2011

People in parallel benchmarking

- John L. Gustafson

American computer scientist and businessman, chiefly known the invention of Gustafson's Law, introducing the first commercial computer cluster. Since March 2009, he has been Director of Intel's Extreme Technologies Research laboratory at Intel's headquarters in Santa Clara.

Performance Issues

- coverage
  - Coverage is the percentage of a program that is parallel.
- granularity
  - how much work is in each parallel region.
- load balancing
  - how evenly balanced the work load is among the different processors.
  - loop scheduling determines how iterations of a parallel loop are assigned to threads
    - if load is balanced a loop runs faster than when the load is unbalanced
- locality and synchronization
  - cost to communicate information between different processors on the underlying system.
    - synchronization overhead
    - memory cache utilization
  - need to understand machine architecture

Superlinear speedup

- Is it possible?
  - “For a *P*-processor algorithm, simply execute the work of each processor on a single processor and the time will obviously be no worse than *P* times greater. It will usually be *less*, because sources of parallel inefficiency are eliminated.”

Superlinear Speedup

- Sometimes superlinear speedups can be observed!
- sources of superlinear speedup:
  - inefficiencies in the serial solution
  - hidden memory latency and the effect of tiered memory structures
    - More processors typically also provide more memory/cache.
    - Total computation time decreases due to more page/cache hits.
  - subdivision of system overhead
  - shift in time fraction spent on different speed tasks