CSE 6230 – HPC Tools and Applications

Ramakrishnan Kannan

Shruti Shivakumar

Introduction

- Course Instructor – Ramakrishnan Kannan(rkannan3@gatech.edu)
- Teaching Assistant – Shruti Shivakumar (sshivakumar9@gatech.edu)
- Number of Registered Students – 60+
- Time and Venue
  - Tuesdays and Thursdays from 5:00-6:15pm in Ford Environment Science and Technology Building, Room L1255.
- Mode
  - Hybrid – All classes will be in-class and as well as over zoom
  - Check the website schedule for in-class lectures

Grading

- Three components
  - 4 Programming assignments – 30%
    - Shared memory – 5%
    - Accelerated – 5%
    - Distributed Memory – 10%
    - Final Complexity and performance analysis – 10%
  - Midterm – 10%
  - Final Group Project – 60%
    - Proposal – 15%
    - Demonstration – 15%
    - Report – 15%
    - Final Presentation – 15%
- All deliverables and exams will be graded by TA. The project proposal and final presentation will be graded by both TA and Instructor

Prerequisites

- Programming experience – C/C++.
- Algorithm Analysis
- Preferred Courses
  - HPC Architecture – Tom Conte or Rich Vuduc
  - Algorithms – One of CS 6550 / CSE6140 / CSE6220
- Time requirements
  - Can vary based on student's background and experience
  - Programming and debugging experience
  - Don't wait till last minute for programming assignments and project

Previous 6230 Courses

- Professor Ümit V. Çatalyürek lectures
- Professor Chow’s lectures
- Professor Vuduc’s lectures

Office Hours

- Instructor Office hours and location
  - Thursday 2-3pm EST
  - Check Piazza for Zoom Link
- TA office hours and location
  - Tuesday 3:45-4:45pm EST
  - Check Piazza for Zoom Link

Announcement and Discussions

- Website - https://ramkikannan.com/teaching/
- Piazza - https://piazza.com/gatech/spring2023/cse6230
- Canvas/(Github classroom?) -

Late Policy and Due dates

- All assignments will be available on canvas after the class at 6pm on the announcement date. Check the website for details.
- All assignment are due before the class at 4:55pm on the deadline
- Two days extension can be provided if notified a day before the deadline to the TA for a 20% penalty.
- No penalties for medical reasons or emergencies.

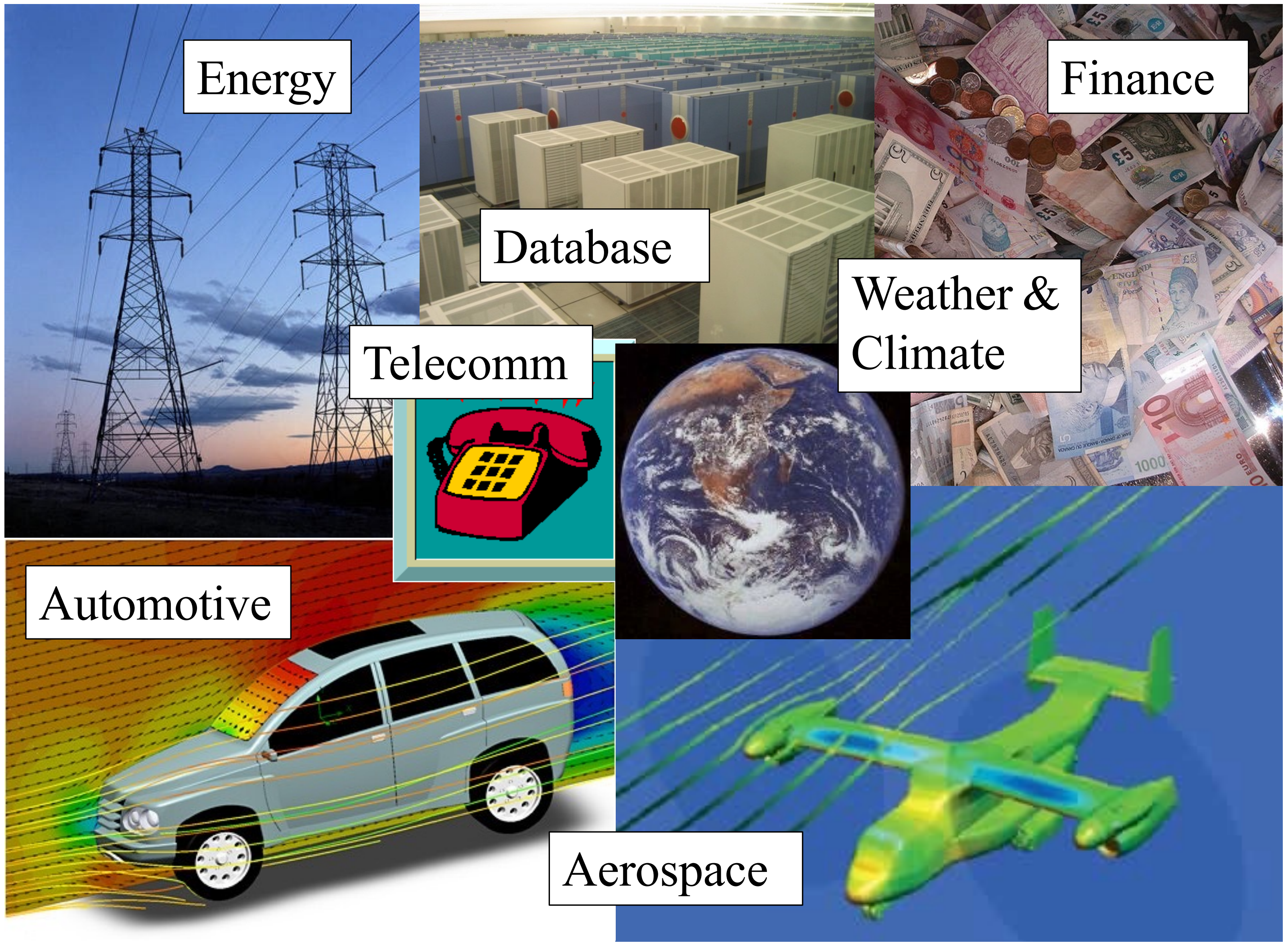
Introduction to HPC

Motivated out of Rich Vuduc’s Lectures, SC’22 Parallel Computing 101 Tutorial, https://hpc.llnl.gov/training/tutorials/

Introduction

- In this first half we introduce parallel computing and some useful terminology.
- We examine many of the variations in system architecture, and how they affect the programming options.
- We will look at a representative example of a large scientific/engineering code, and examine how it was parallelized. We also consider some additional examples.
- https://hpc.llnl.gov/training/tutorials/
- Jack Donggarra’s Turing Award Lecture

Scientific Applications



Motivation for Parallel Computing

- Natural fit: Real world is inherently parallel (weather, traffic jams, assembly lines, ant colonies, tutorials, ...).
- Parallel computers can be the only way to achieve specific computational goals
  - ExaFLOPS (10^18 Floating Point Operations per Second) for complex problems
  - Handling of exabytes of storage in data centers
  - Mega-transactions per second for search engines, ATM networks, digital multimedia, social media
  - Next Generation AI problems
- All computers are parallel -- cellphone, M1, AMD and Intel, and the parallelism is increasing
- Save time to solution and/or money – Accelerating scientific discoveries
- Solve larger or more complex problems (e.g. finer grids)
- CPU Scavenging Grid – Better Utilization of Idle Computers

Basic Terminologies

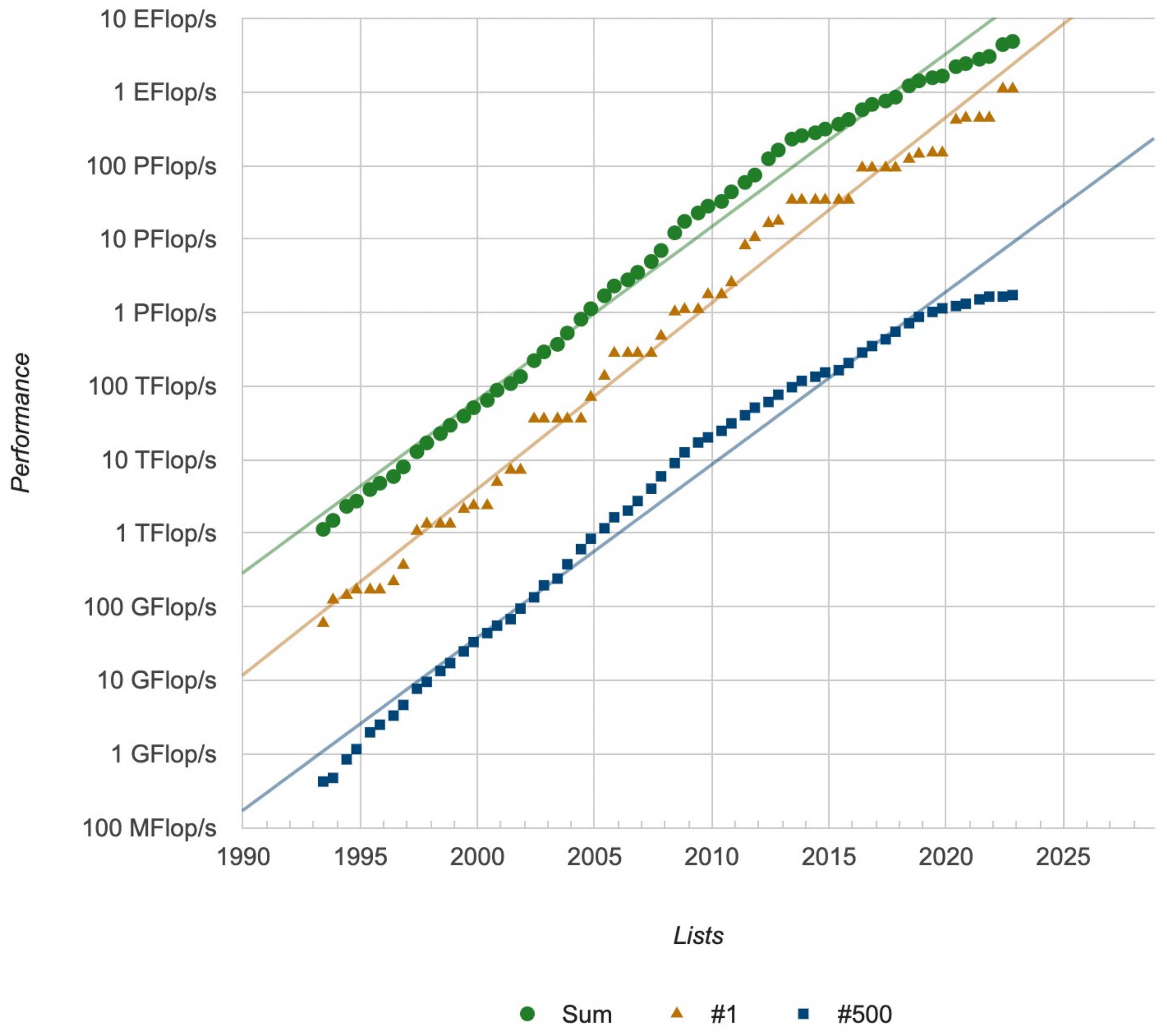
- Classical Problem - The definitions are fuzzy, many terms are not standardized, definitions often change over time.
- Many algorithms, software, and hardware systems do not match the categories, often blending approaches.

Basic Terminologies - II

- **Parallel Computing**- Solving a task by simultaneous use of multiple processors in a unified architecture.
- **High Performance Computing**- Solving large problems via supercomputers + fast networks + massive storage.
- **Embarrassingly Parallel** - Solving many similar, but independent, tasks. E.g., parameter sweeps.
- **Multi-core/Many-core Processors** - Almost all processors today. Multiple compute cores on a single chip. They share memory, operating system and network.
- **Cluster Computing** - Combination of commodity units (e.g. multi-core processors) to build parallel system.
- **Pipelining (streaming)** - Breaking a task into steps performed by different units, much like an assembly line.

Pipelining – Automation Industry

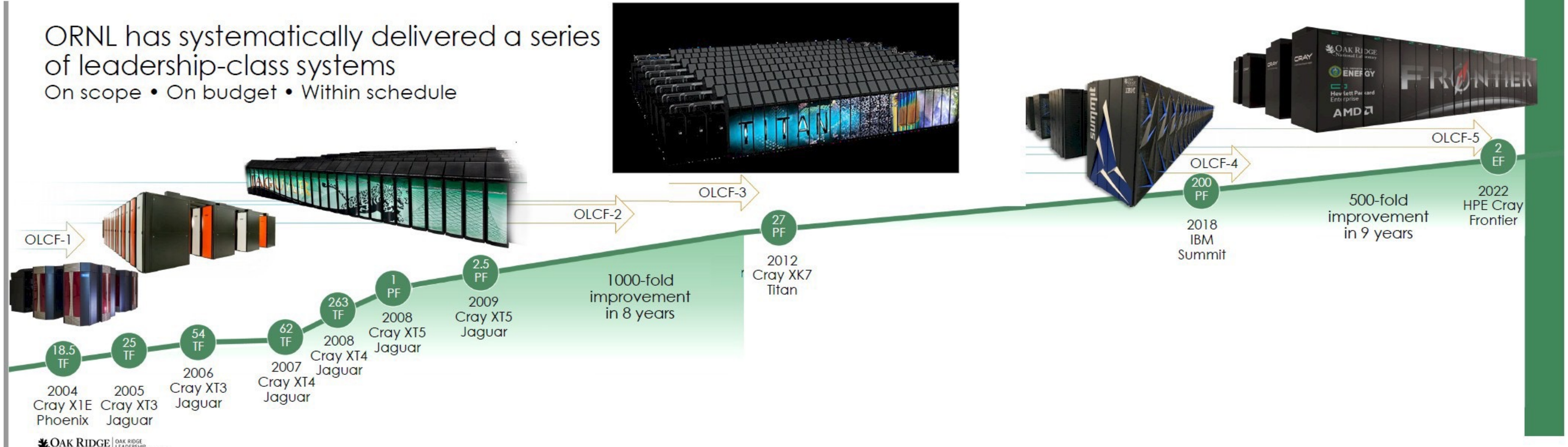
Top500 -- Performance



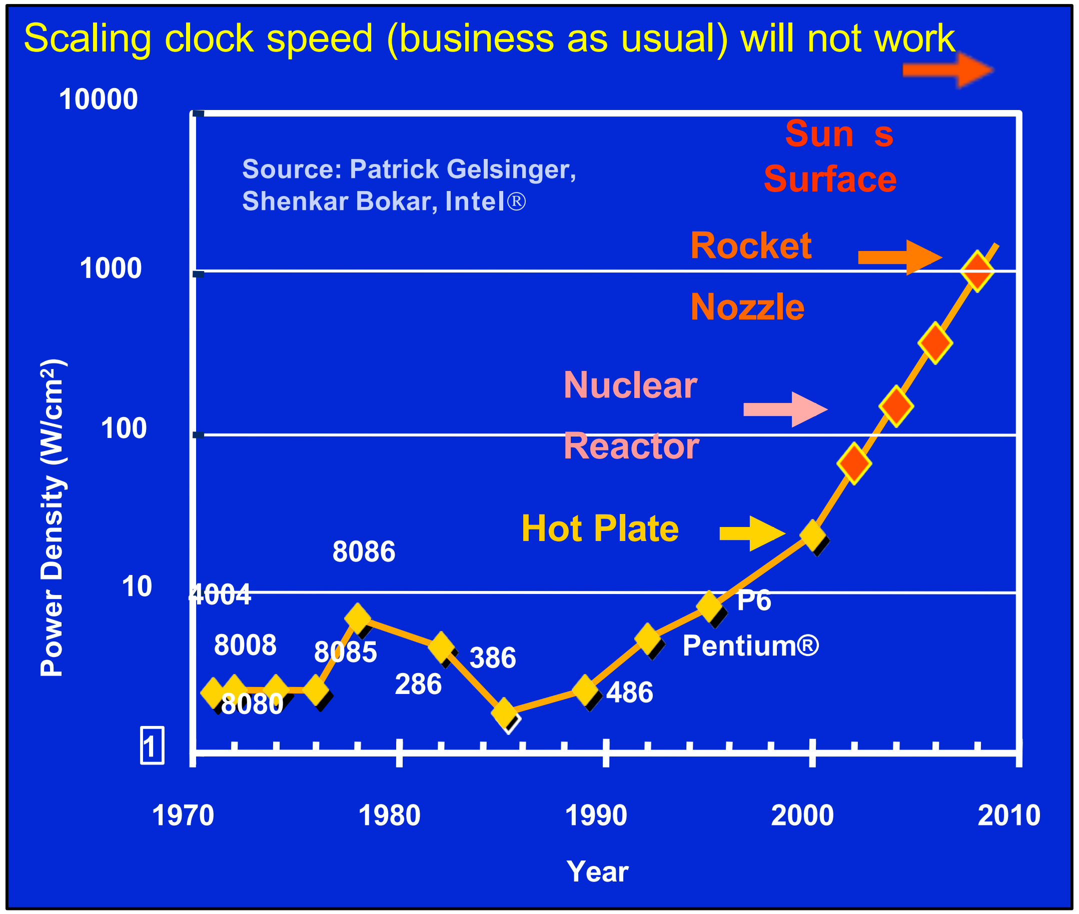
*As of 1/10/22

ORNL has systematically delivered a series of leadership-class systems

On scope • On budget • Within schedule

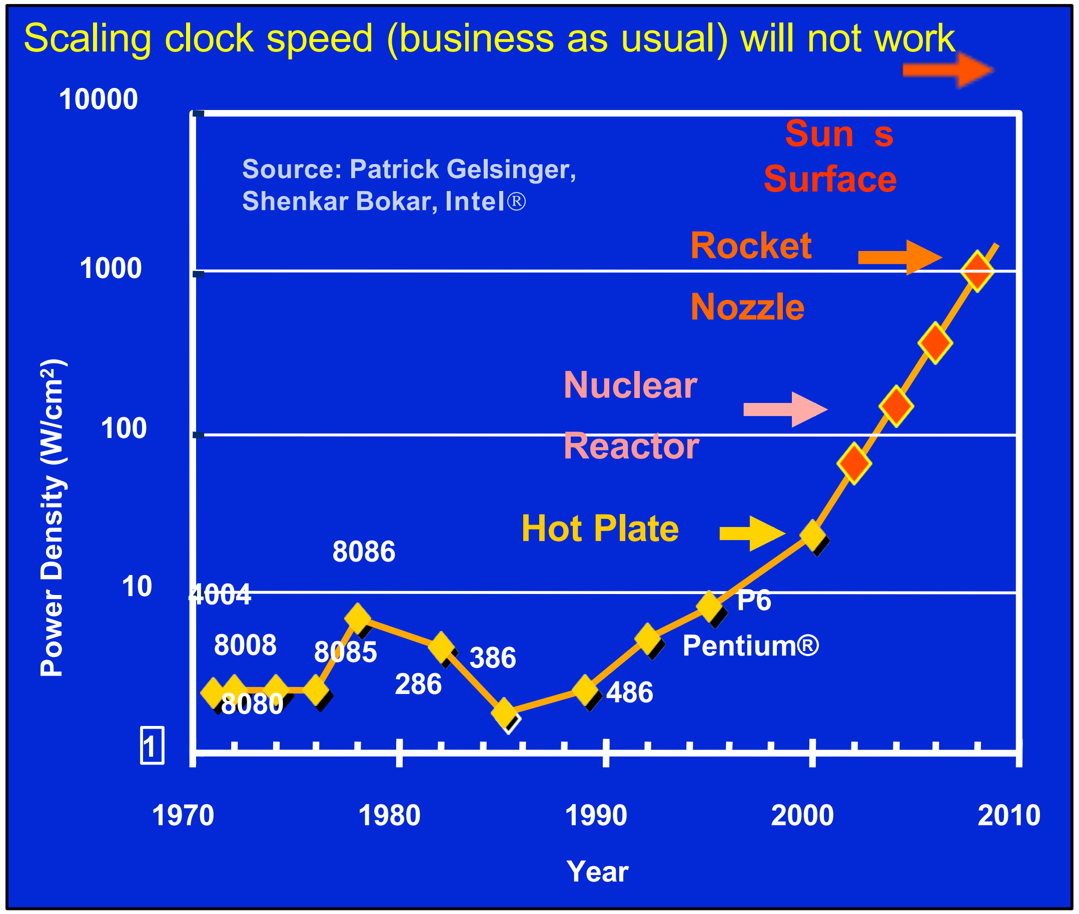


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY



Parallelism & power

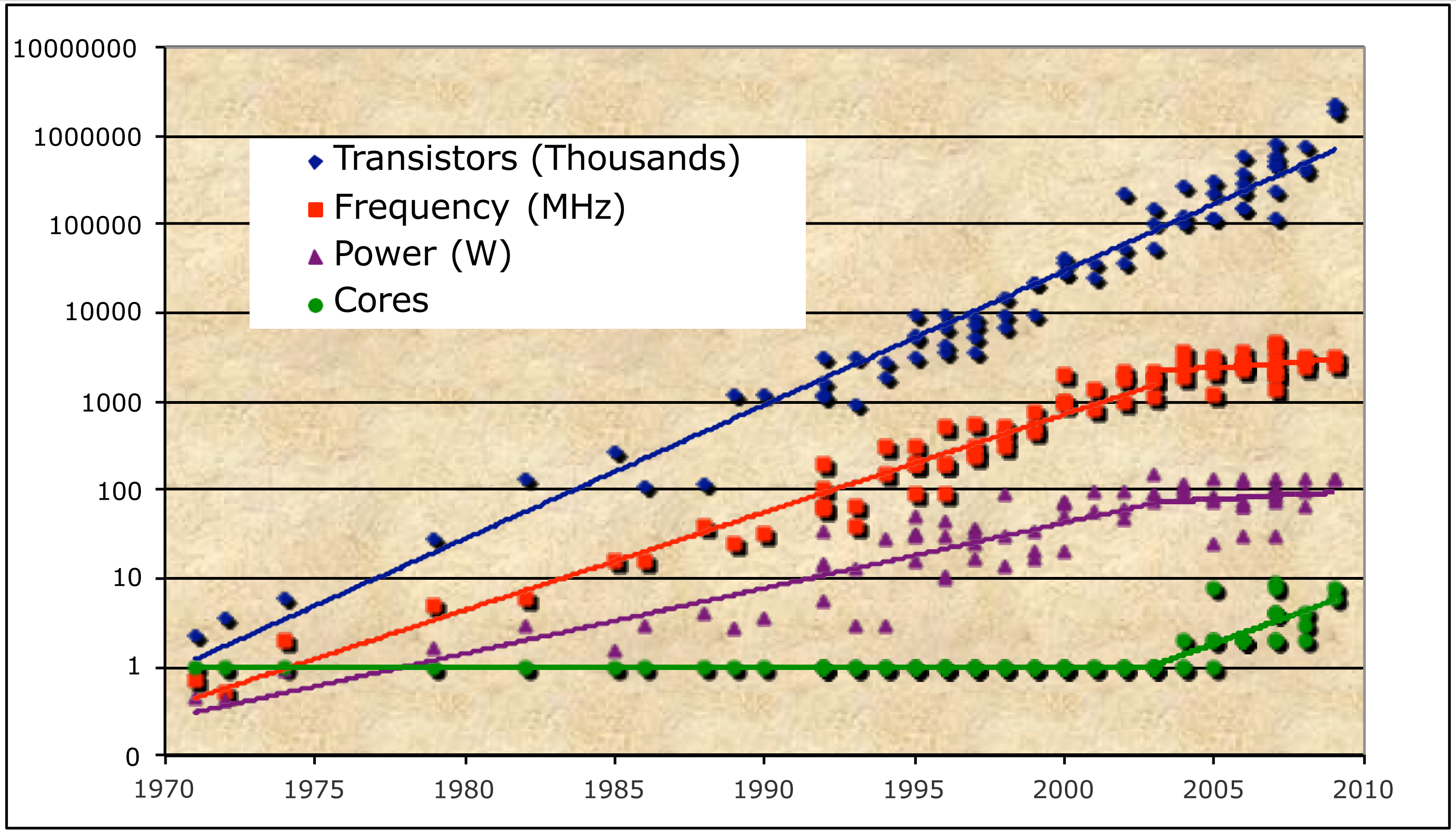
Is it better to increase speed by doubling frequency or cores?



$$\text{Performance} \propto (\text{cores}) \times (\text{freq})$$
$$\text{Power} \propto (\text{cores}) \times (\text{freq}^{2.5})$$

Parallelism & power

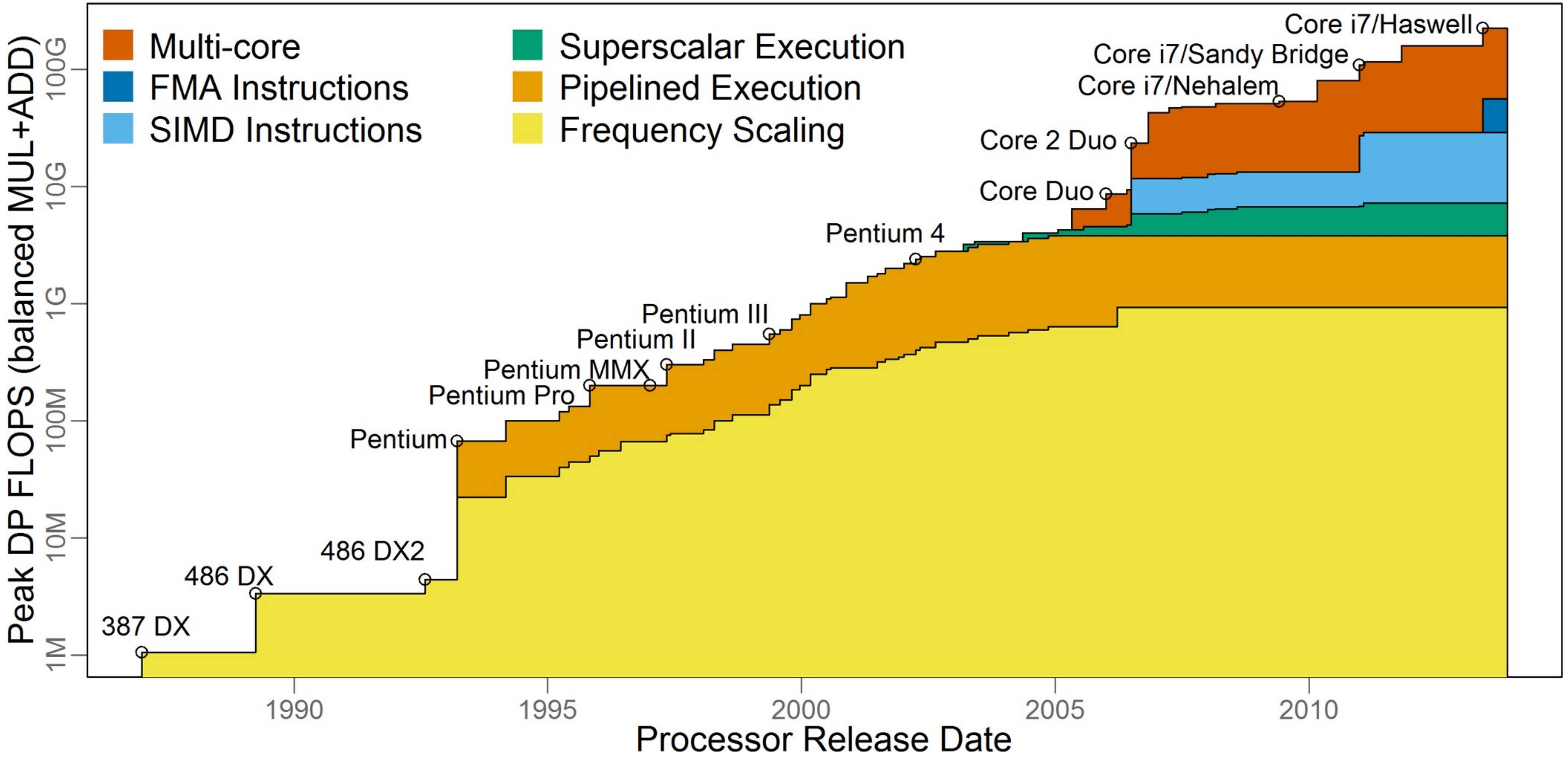
Is it better to increase speed by doubling frequency or cores?



Parallelism & power

Observe transition in ~ 2004.

Source: K. Yelick @ UCB – http://www.cs.berkeley.edu/~demmel/cs267_Spr11/



Source: Marat Dukhan <mdukan3@gatech.edu>

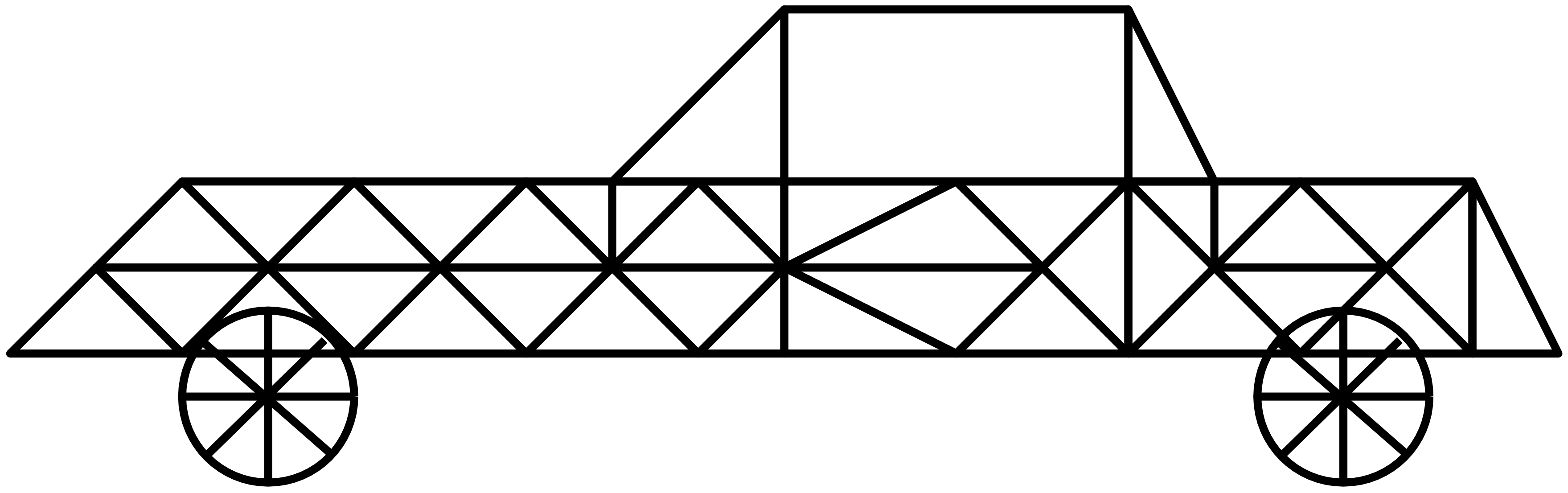
Crash Simulation

- A greatly simplified model, based on parallelizing crash simulation for Ford Motor Company. Simulations save significant money and time compared to testing real cars
- This example illustrates various aspects common to many simulations and other large-scale applications.

Finite Element Representation

- Car is modeled by a triangulated surface (elements).
- The simulation models the movement of the elements, incorporating the forces on them to determine their new position.
- In each time step, the movement of each element depends on its interaction with the other elements that it is physically adjacent to.
- In a crash, elements may end up touching that weren’t touching initially (not good!)
- The state of an element is its location, velocity, and information such as whether it is metal that is bending.

Car and Finite Element Representation



Serial Crash Simulation

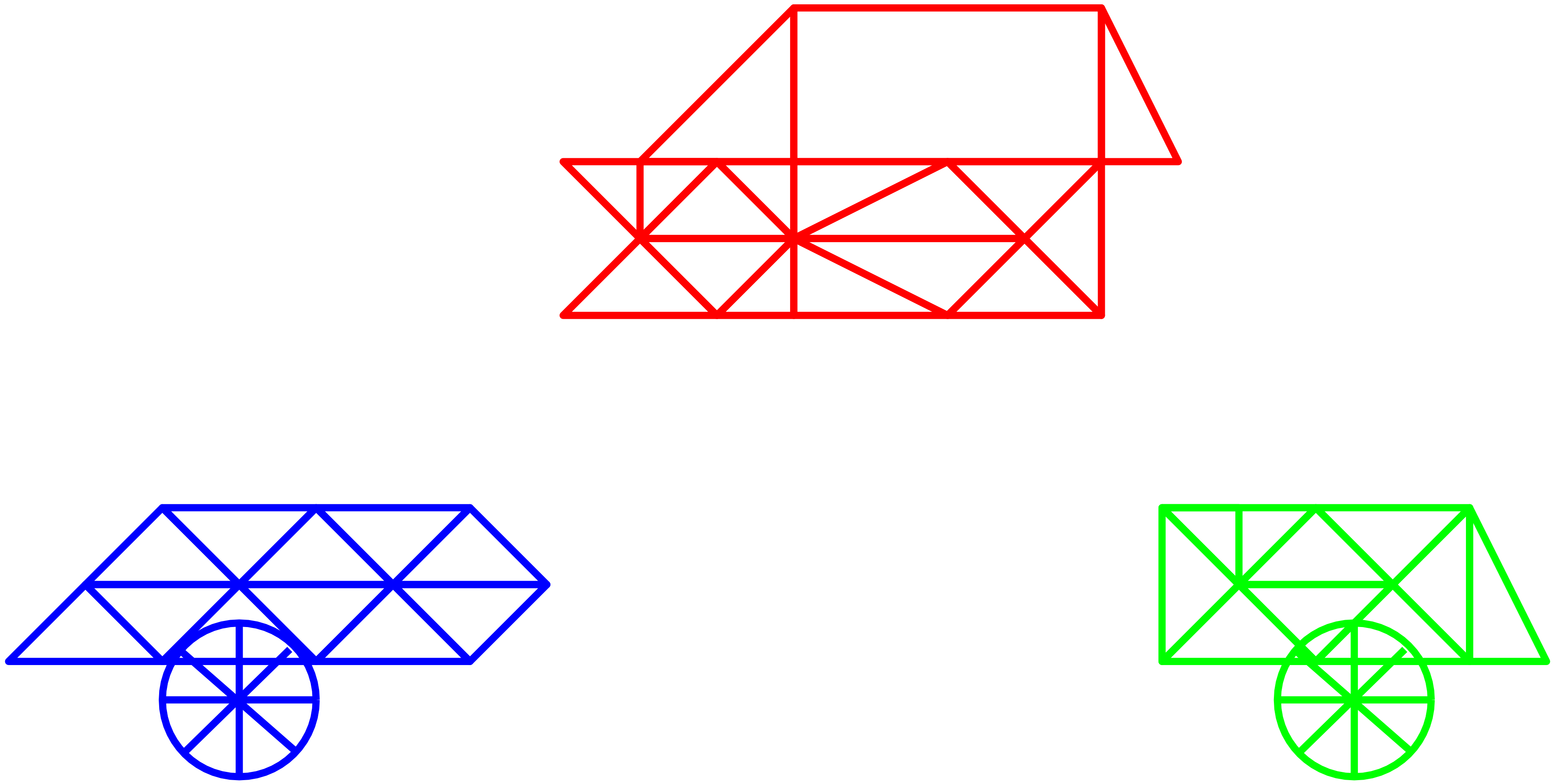
- For all elements
  - Read State(element), Properties(element), Neighbor_list(element)
- For time=1 to end_of_simulation
  - For element = 1 to num_elements
    - Compute State(element) for next time step, based on previous state of element and its neighbors, and on properties of element

Simple Parallelization

- Parallel computer based on PC-like processors linked with a fast network, where processors communicate via messages. Distributed memory or message-passing .
- Distribute elements to processors, each processor updates the positions of the elements it contains: **owner computes** .
- All machines run the same program: SPMD , single program multiple data.

SPMD is the dominant form of parallel computing.

A Distributed Car



Parallel Crash Simulation

Concurrently for all processors P

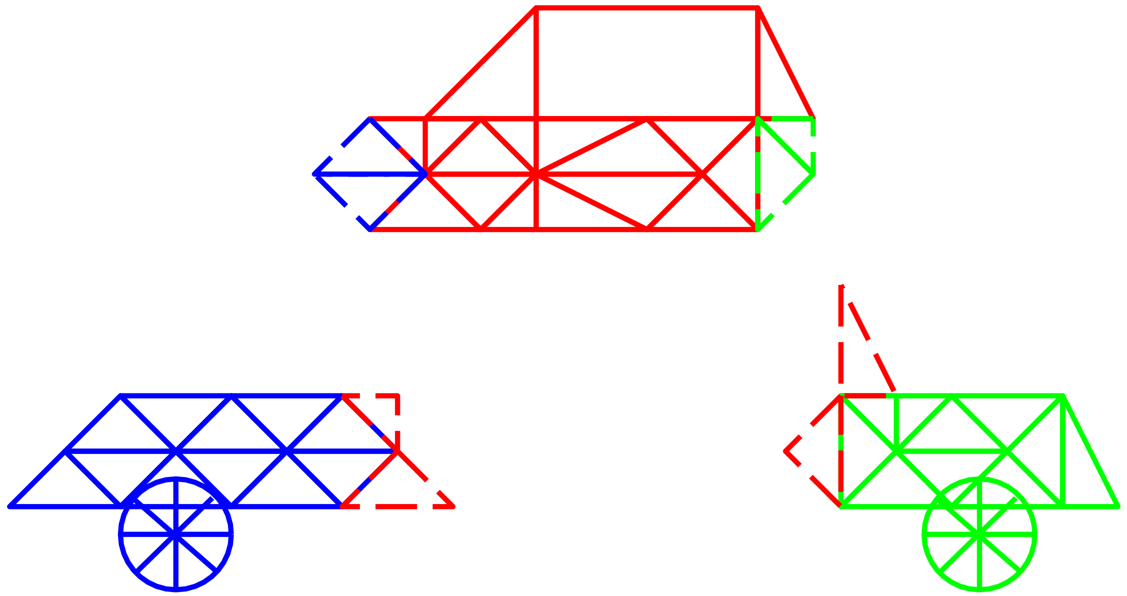
- For all elements assigned to P
  - Read State(element), Properties(element), Neighbor-list(element)
- For time=1 to end-of-simulation
  - For element = 1 to num-elements-in-P
    - Compute State(element) for next time step, based on previous state of element and its neighbors, and on properties of element

Distributing the car

- How is the car distributed across P?
  - Typically element assignment determined by serial preprocessing using domain decomposition approaches described later.
  - Need to keep the load balanced among the processors, otherwise some will be idle waiting for others

Connecting Pieces

- How does processor keep track of elements in other processors?
  - Ghost cells - (halos) are copies of values computed elsewhere
  - Need to minimize communication time



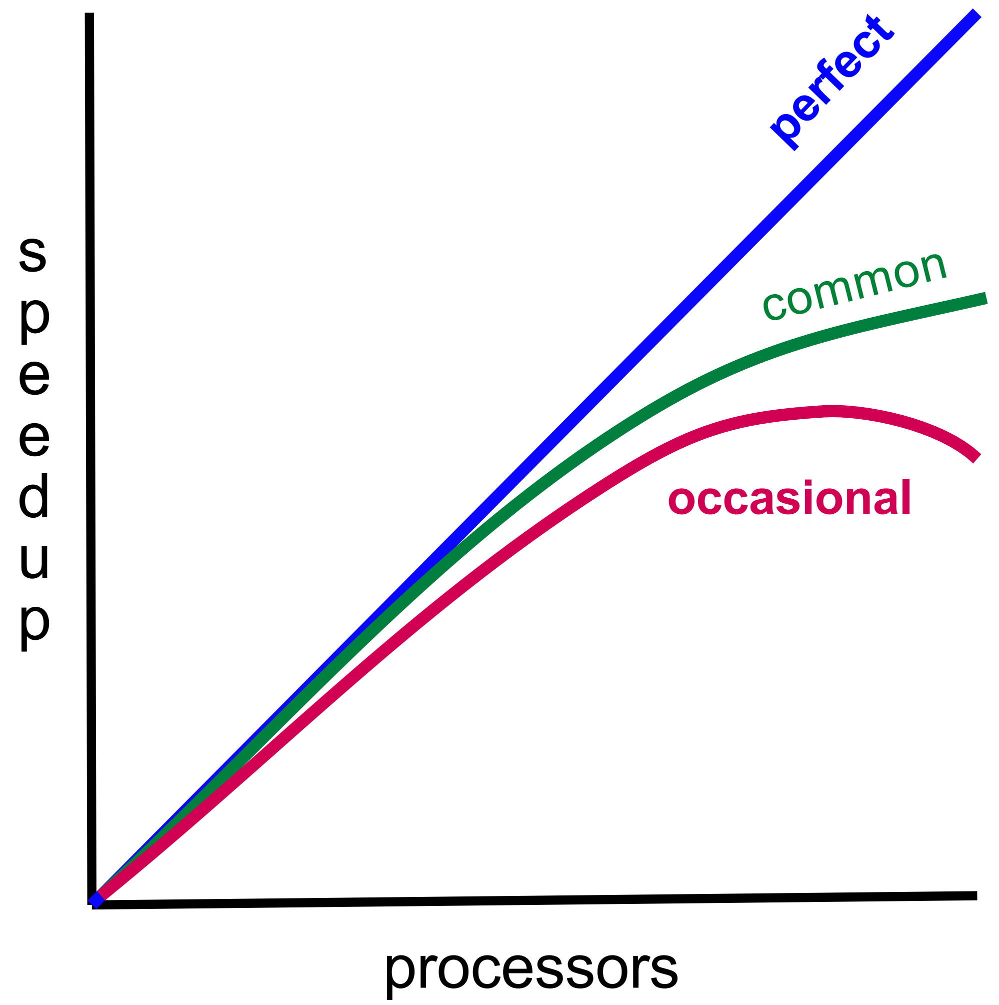
How good is a parallel computation?

- An important component of effective parallel computing is determining whether the program is performing well. If it isn’t, or can’t be scaled to the target number of processors, then one needs to determine the causes of the problem and develop better approaches.

Some definitions

- For a given problem A, let
  - SerTime(n) = Time of best serial program to solve A for input of size n.
  - ParTime(n,p) = Time of the parallel program+architecture to solve A for input of size n, using p processors.
- Note that SerTime(n) ≤ ParTime(n,1).
- *Speedup(n,p): SerTime(n) / ParTime(n,p)* 0 < Speedup ≤ p
- *Work(n,p): p · ParTime(n,p) ← cost* Serial Work ≤ Parallel Work < ∞
- *Efficiency(n,p): SerTime(n) / [p · ParTime(n,p)]* 0 < Efficiency ≤ 1

Speedup



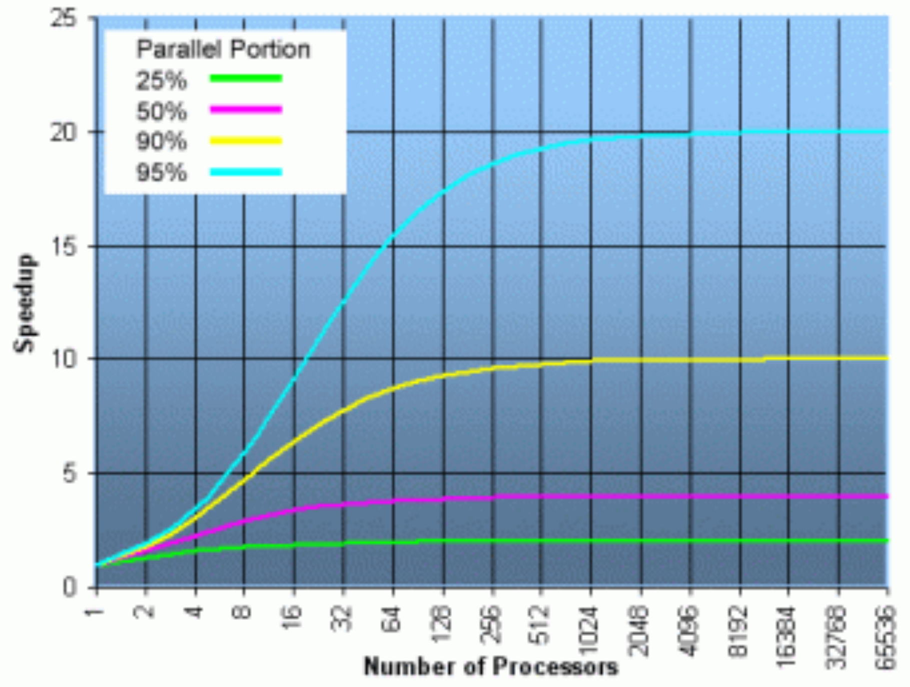
Very rare. Some reasons for speedup > p (efficiency > 1)

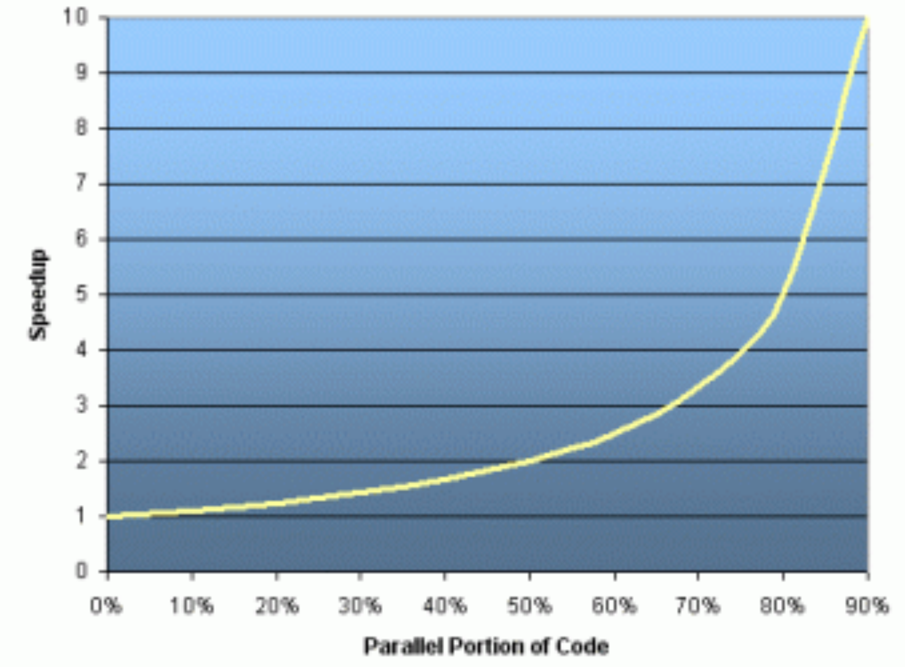
- Parallel computer has *p* times as much RAM so higher fraction of program memory in RAM instead of disk. .An important reason for using parallel computers
- In developing parallel program a better approach was discovered, older serial program was not best possible.
  - A useful side-effect of parallelization

Amdahl’s Law

- Amdahl [1967]: Let f be fraction of time spent on operations that are performed serially. Then for
  - ParTime(p) ≥ SerTime(p) $\cdot \left[f + \frac{1-f}{p}\right]$
- *Speedup(p)* $\leq \frac{1}{f + \frac{1-f}{p}}$
- Which implies

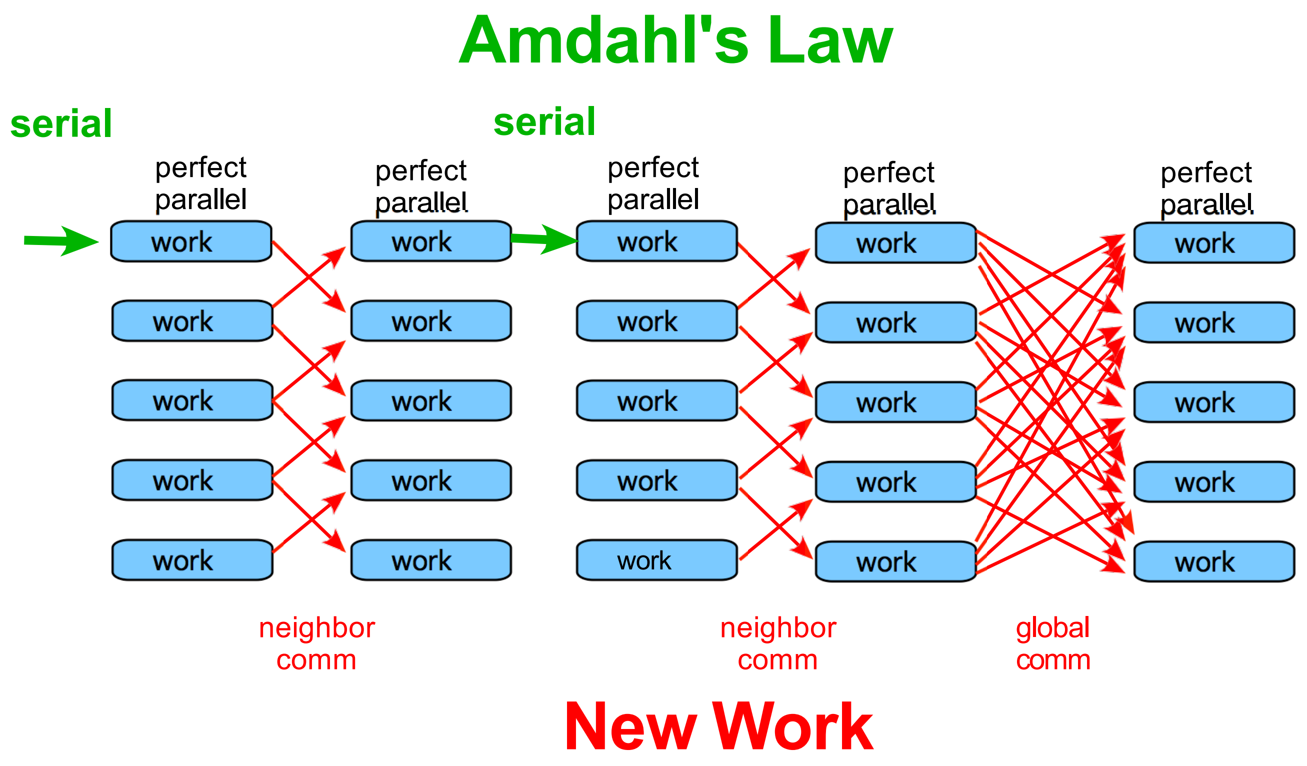
Speedup ≤ 1/*f*





Amdahls Law - II

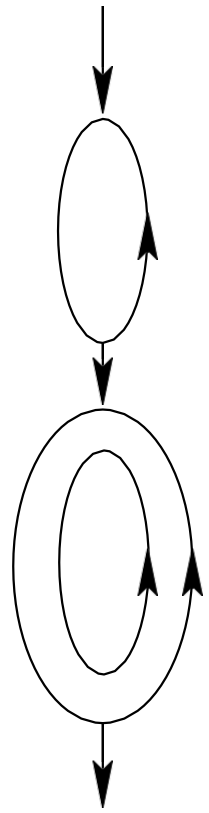
- Parallelization usually adds communication – which Amdahls doesn't consider
- For Crash: ghost cells sent every time step and periodic global communication to check if parts are colliding.



- Hope
  - Algorithm: New algorithms with much smaller values of f .
  - Necessity is the mother of invention.
  - Memory hierarchy: More time spent in RAM than disk.
  - Scaling: Usually time spent in serial portion of code is a decreasing fraction of the total time as problem size increases.

Program Structure

Often serial part grows with n much slower than total time.



Serial, time grows slowly with n

Parallelizable loop, grows with n

Serial, fixed time

Parallelizable loop within loop,
time grows very rapidly with n

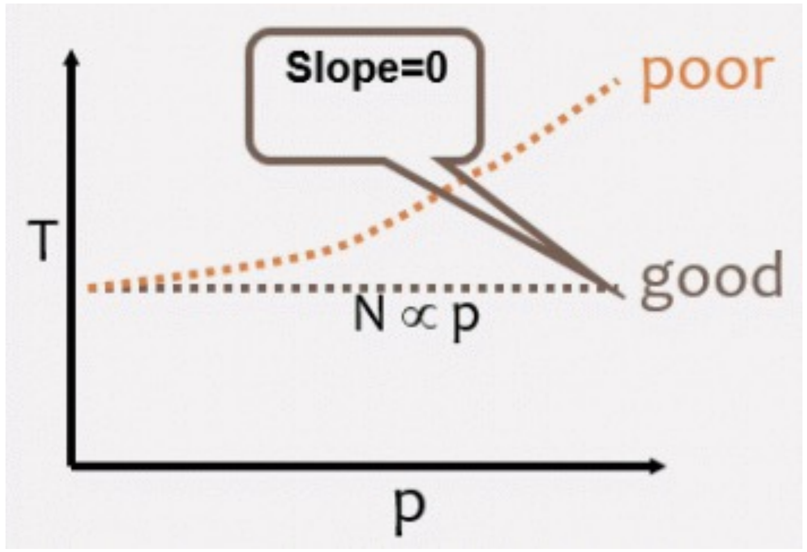
Serial, time grows slowly with n

I.e., as n $\nearrow$ Amdahl's "f" $\searrow$

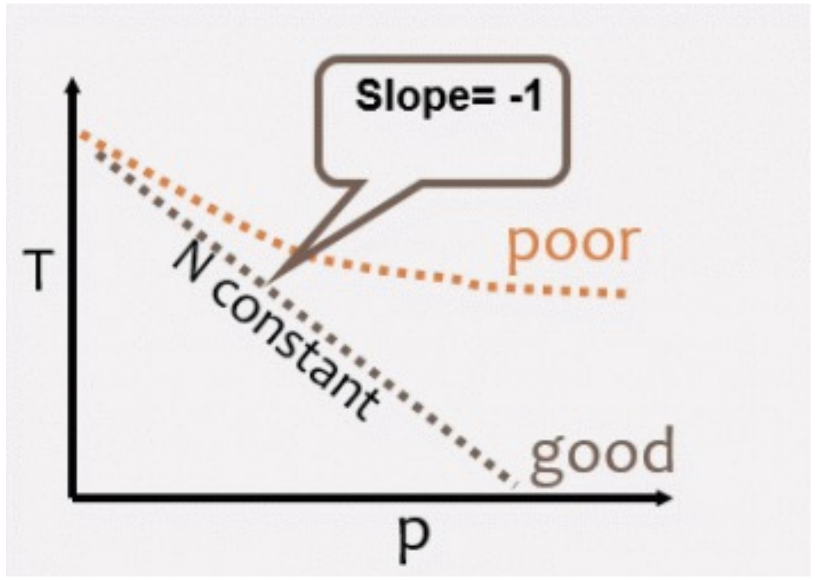
Scaling

- Utilize large computers by increasing n as p increases
- Fix the amount of data per processor: ***weak scaling***
  - Efficiency can remain high if communication does not increase excessively
  - Warning: efficiency improves, but parallel time will increase if SerTime(n) superlinear (ω(n)).
- Amdahl considered ***strong scaling*** : n is fixed
- Linear speedup is difficult
  - Nothing scales to arbitrarily many processors.
- However, for most users, the important question is:
  - Have I achieved acceptable performance on my software/hardware system for a suitable range of data and system sizes?

Weak Scaling



Strong Scaling


Slope= -1
T
poor
N constant
good
p