BENCHMARKING & PERFORMANCE

Sid Chi-Kin Chau

[Lecture 8]

Benchmarking

- Benchmarking
  - Compare the performance of different algorithms and systems
  - Evaluate practical performance with real-world input data
  - Optimize best practice, improve implementation and plan resource allocation
  - Collect and analyze practical performance data
  - Provide assurance and confidence before practical deployment
- Benchmarking trials
  - Construct a suite of independent trials for which the algorithm is executed
  - Trials are executed and milli/nanosecond-level timings are taken before and after the algorithm is executed
  - Eliminate inconsistent measurements

Benchmarking

- Performance measurements may be different in a different time, even with same code and implementation
  - Computer background processes may affect practical performance
  - Eliminate outliner performance data
- In Java, the system garbage collector may affect the performance
  - The system garbage collector is invoked immediately prior to the trial
    - Call `System.gc()`
  - Although this cannot guarantee that the garbage collector does not execute during the trial, it may reduce the impact

Benchmarking in Java

- Example of benchmarking simple summation

```
public class Trial {
    public static void main (String[] args) {
        for (long len = 1000000; len <= 5000000; len += 1000000) {
            for (int i = 0; i < 30; i++) {
                System.gc(); //Invoke garbage collector
                long start = System.currentTimeMillis();

                // Simple summation to be timed
                long sum = 0;
                for (int x = 1; x <= len; x++) { sum += x; }

                long end = System.currentTimeMillis();
                // Output runtime
                System.out.println("trial:" + len + " runtime:" + (end - start));
            }
        }
    }
}
```

Benchmarking in Java

- Instead of millisecond-level timers, nanosecond timers could be used
- In Java, invoke `System.nanoTime()`

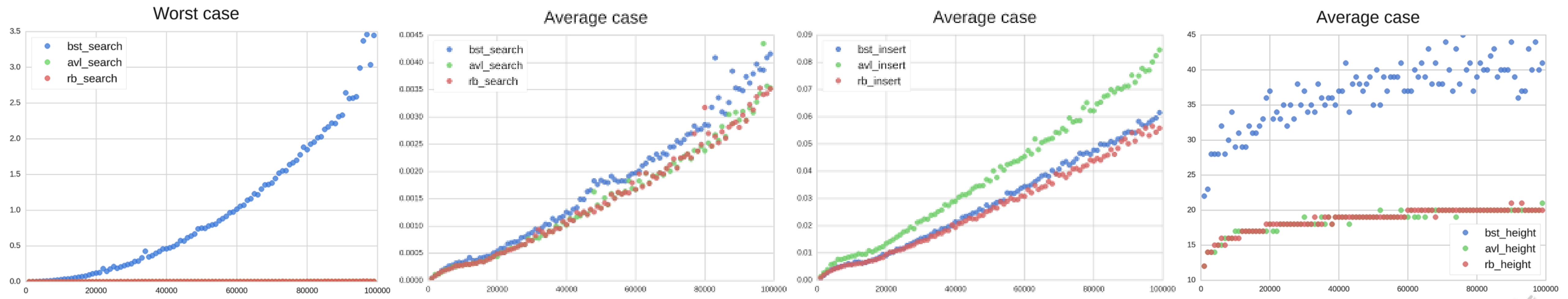
```
for (long len = 1000000; len <= 5000000; len += 1000000) {
    for (int i = 0; i < 30; i++) {
        System.gc(); //Invoke garbage collector
        long start = System.nanoTime(); //Nanosecond timer

        // Simple summation to be timed
        long sum = 0;
        for (int x = 1; x <= len; x++) { sum += x; }

        long end = System.nanoTime();
        // Output runtime
        System.out.println("trial:" + len + " runtime:" + (end - start));
    }
}
```

Benchmarking Data Structures

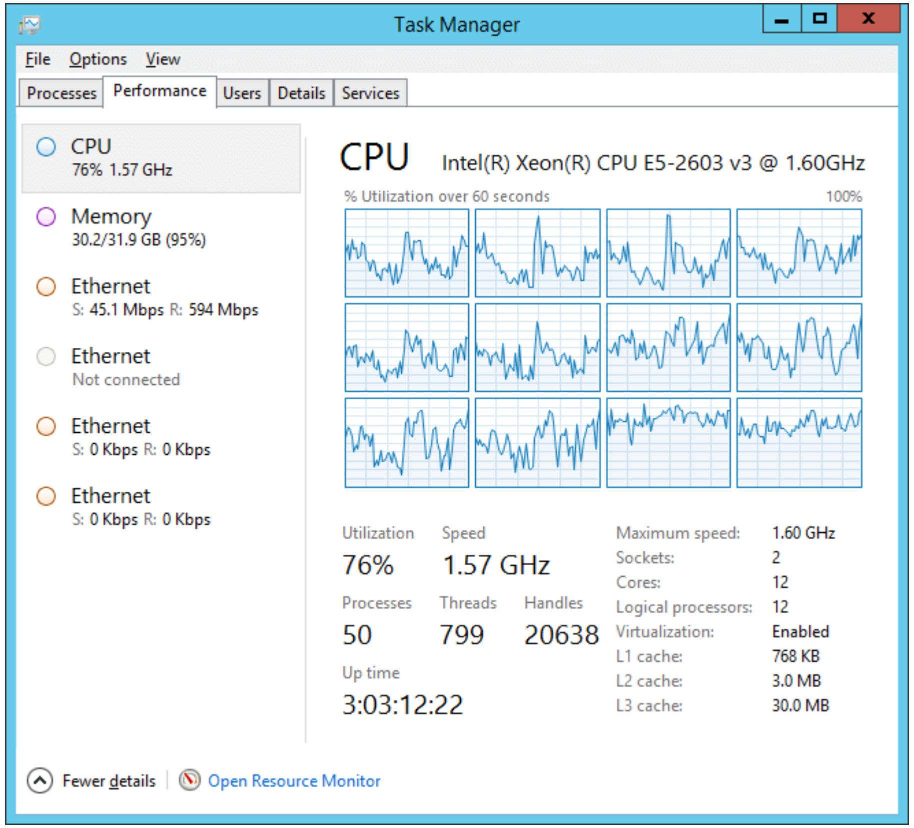
- Benchmark binary search tree, red-black tree and AVL tree
  - Consider worst-case (i.e., highly unbalanced tree) and average-case (i.e., random input sequences)
  - Which one of the tree data structures is the best practically?
    - BST is surprisingly efficient. Why?



Source: https://codedeposit.wordpress.com/2015/10/07/red-black-vs-avl/

Aspects of Performance Analysis

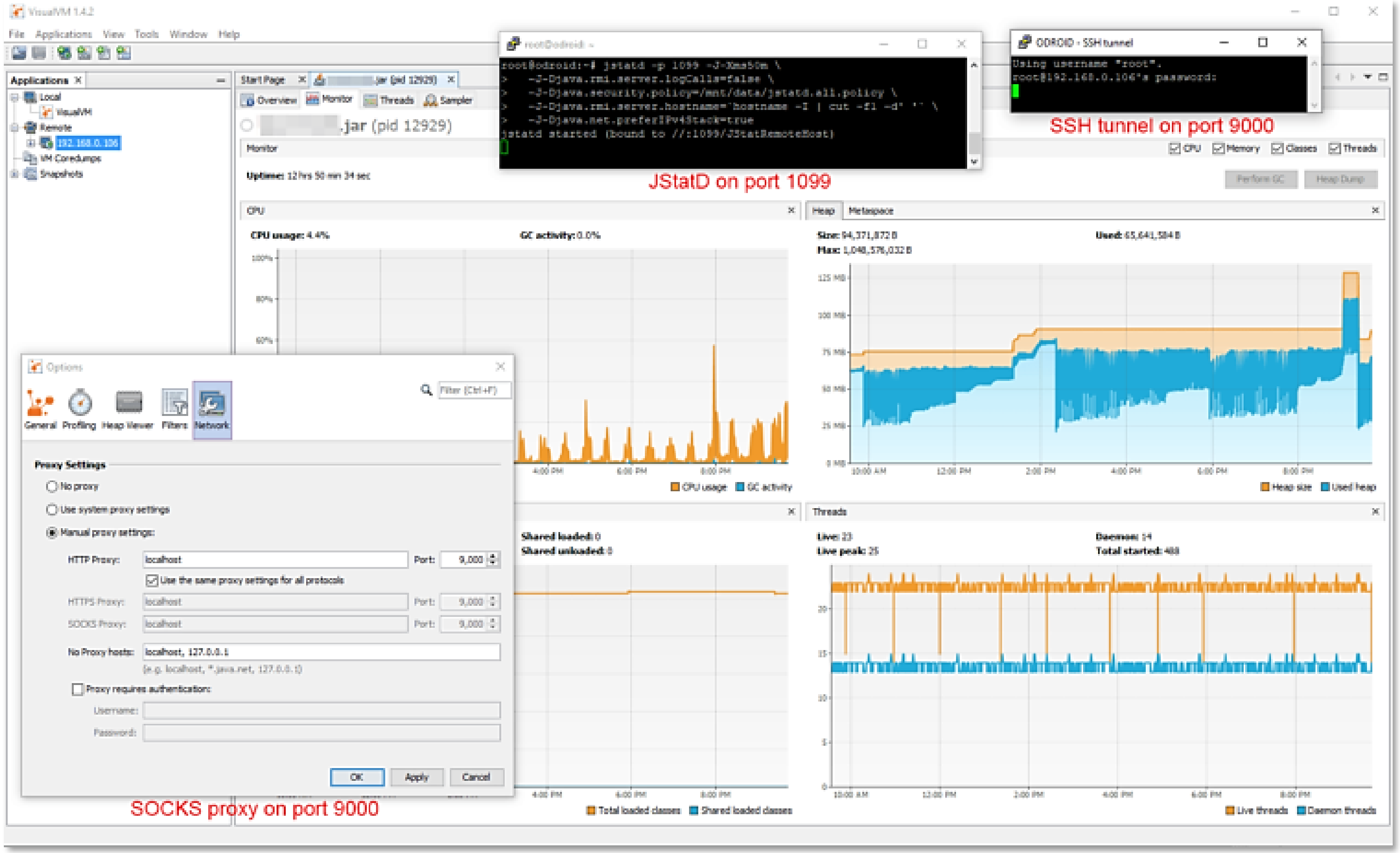
- Does your software perform as what you expect?
  - Does it complete fast?
  - Does it work well with more inputs?
  - Does it break?
- Metrics of performance:
  - Latency, throughput, memory size
  - Network bandwidth
  - Concurrency
- Performance analysis
  - Best case
  - Average case
  - Worst case



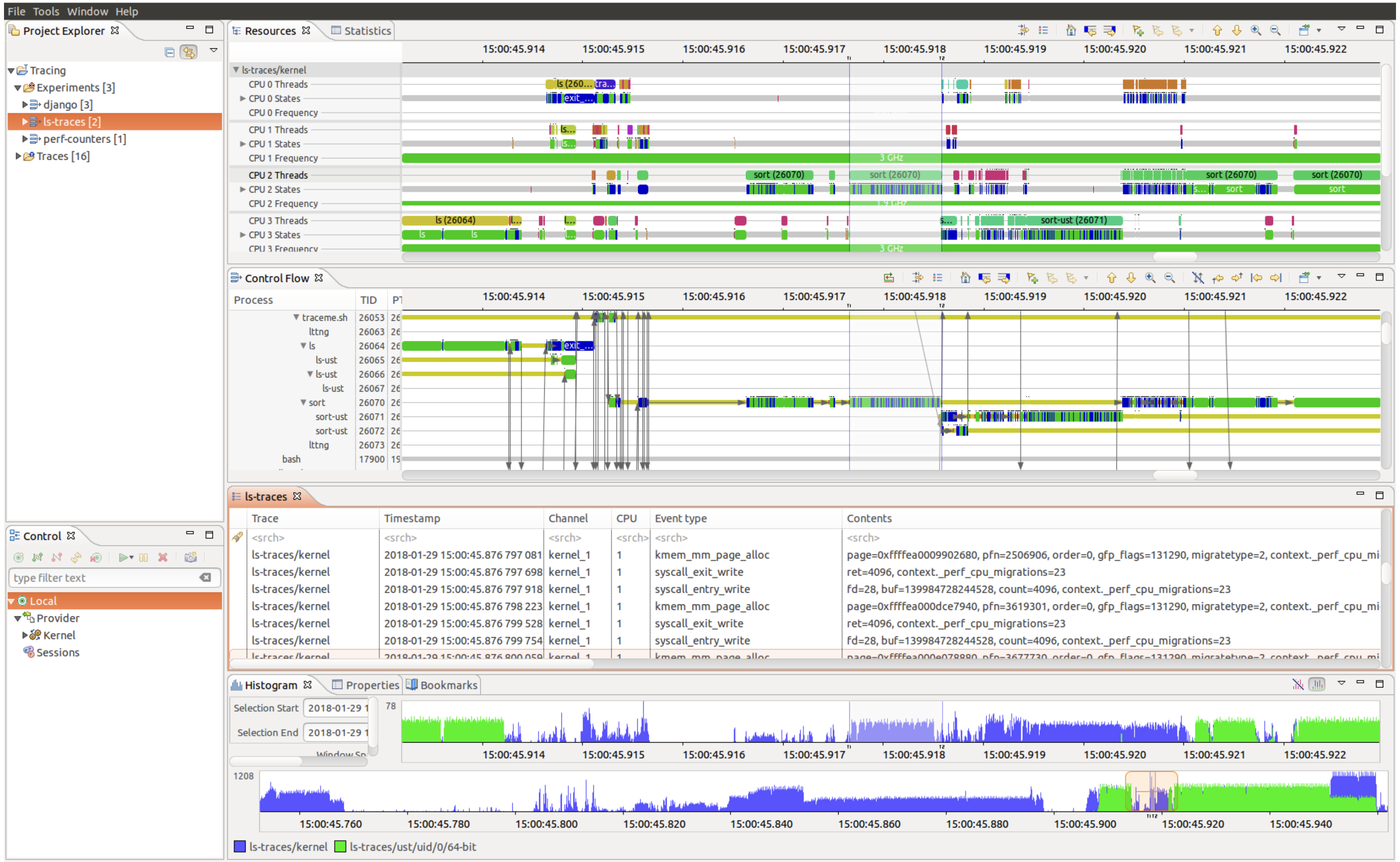
Performance Profiling

- Tools that provide a visual interface for detailed information about the runtime operations of a program
  - Understand how your program utilizes different computing resources
    - e.g. memory, CPU, GPU, hard disk, network
  - Identify the bottleneck of your program
  - Optimize program implementation
  - Locate potential bugs in your program
- Example:
  - JConsole
  - VisualVM
  - Eclipse Trace Compass

VisualVM


SSH tunnel on port 9000
JStatD on port 1099
SOCKS proxy on port 9000

Trace Compass



Performance Evaluation by Simulation

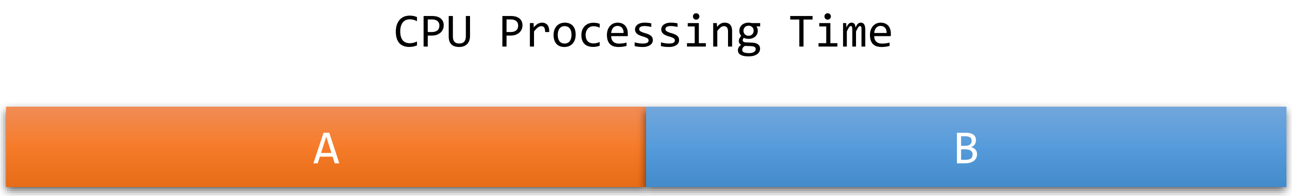
- How to obtain practical performance evaluation of algorithms and systems considering realistic inputs?
  - *Real-world deployment*
    - Setup a small-scale deployment for performance evaluation
    - Expensive or only small-scale; Cannot obtain prior insights
  - *Modeling and analysis*
    - Mathematical reasoning of performance
    - Only apply to simple systems; modeling needs simplifying assumptions
  - *Simulation*
    - Generate artificial inputs to estimate real-world performance
    - A balance between realism and efficiency

Performance Evaluation by Simulation

- Simulation is cost-effective and does not require many simplifying assumptions, which is a viable approach for performance evaluation
- Dynamic systems
  - Systems (which are controlled by certain algorithms) change with time and respond to random inputs, e.g., a system playing Tetris
  - Sample path is the evolution of states in a dynamic system
  - Computational construction of sample paths is a major part of simulation
- Discrete-event simulations
  - Some sample paths are characterized by finite events
  - Construct a random generation model for the discrete events

Motivating Question of Performance

- You have a program X with two component parts A and B
  - Each of which takes 10 minutes. What is the latency of X?
  - Latency is the time from the beginning to the end to complete a job
- Suppose that you can speedup part B by a factor of 5
  - What is the latency now?
  - What is the overall speedup?
  - If A and B are sequential, then Amdahl's Law provides an answer



Amdahl's Law

- How much extra performance can you have if you speed up some part of your program?
- Notations:
  - S is the overall performance gain
  - k is the speed-up factor
  - α is the portion of speed-up

Unimproved part Improved part

- $T_{new} = (1-\alpha)T_{old} + \alpha\frac{T_{old}}{k} = T_{old}((1-\alpha)+\frac{\alpha}{k})$
- $S = \frac{T_{old}}{T_{new}} = \frac{1}{(1-\alpha)+\frac{\alpha}{k}}$

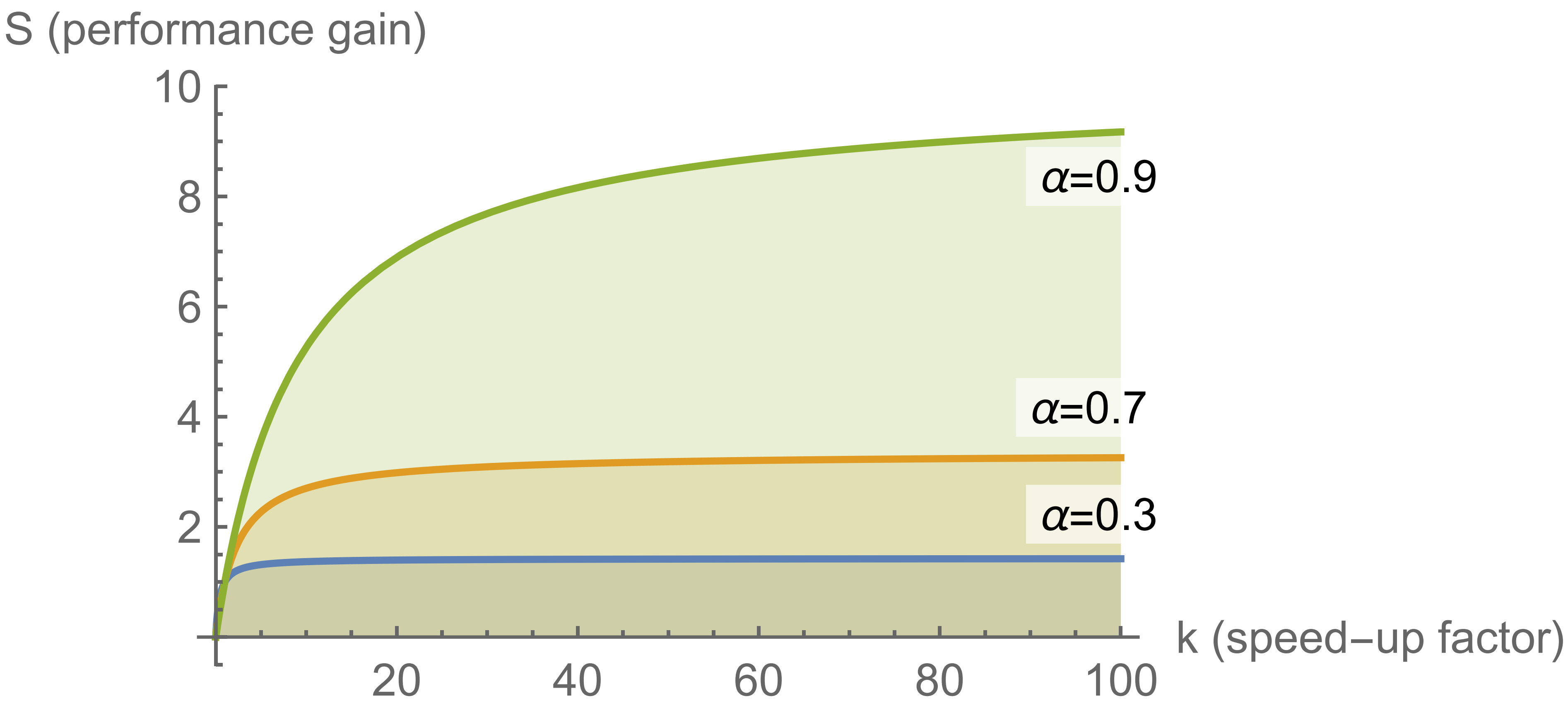
Example

- Your program has one very slow procedure that consumes 70% of the total time. Next, you improve it by a factor of 2
- What is the performance gain in the overall latency?
  - α = 0.7 (70%)
  - k = 2
- $S = \frac{T_{old}}{T_{new}} = \frac{1}{(1-\alpha)+\frac{\alpha}{k}} = \frac{1}{(1-0.7)+\frac{0.7}{2}} = 1.538$

Example

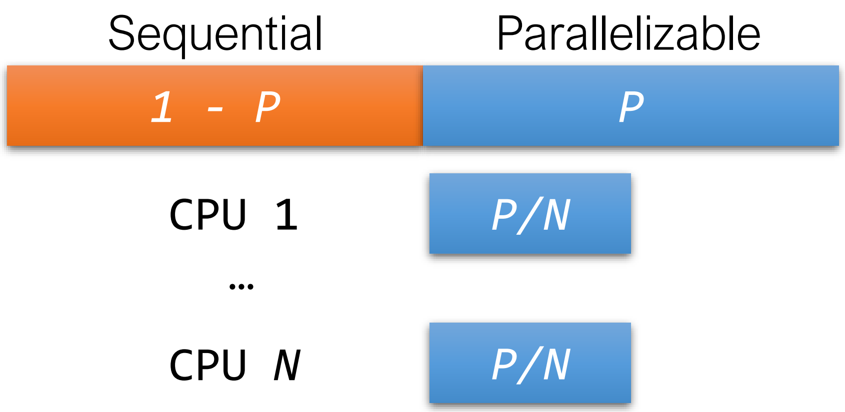
- Floating point instructions could be improved by 2x. Only 15% of instructions are floating point
- What is the performance gain in the overall latency?
  - $\alpha = 0.15$ (15%)
  - $k = 2$
- $S = \frac{T_{old}}{T_{new}} = \frac{1}{(1-\alpha)+\frac{\alpha}{k}} = \frac{1}{(1-0.15)+\frac{0.15}{2}} = 1.081$

Amdahl's Law


S (performance gain)
10
8
6
4
2
20
40
60
80
100
k (speed–up factor)
α=0.9
α=0.7
α=0.3

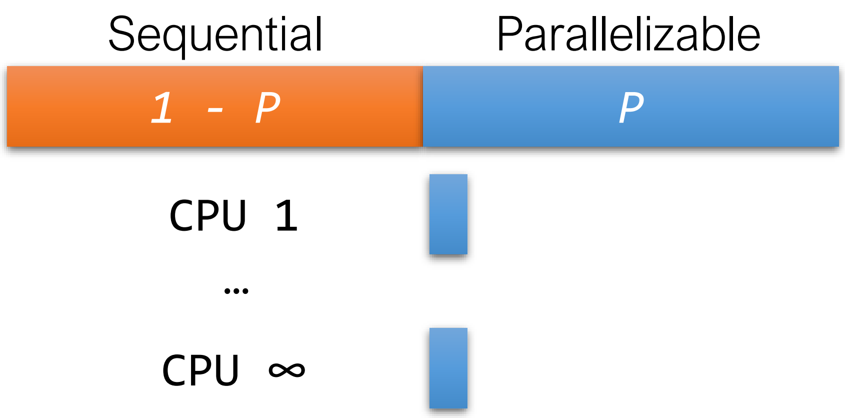
Application to Parallel Processing

- Divide the program into sequential part, 1-*P*, and parallel part, *P*
- Assume there are *N* processors then the improvement of the parallelizable part is *N*
- Based on Amdahl’s law, the performance gain from *N* processors is:
  - $S = \frac{1}{(1-P)+\frac{P}{N}}$



Limit as N → ∞

- The performance gain from a very large number of processors is
- $S = \lim_{N\to\infty} \frac{1}{(1-P)+\frac{P}{N}} = \frac{1}{1-P}$
- Fundamental limitation of performance gain from parallelization (diminishing returns); adding more CPUs may not improve performance
  - Neglects other potential bottlenecks, e.g., memory bandwidth and I/O bandwidth



Reference

- Amdahl's Law paper: "Validity of the single processor approach to achieving large scale computing capabilities"
  - https://inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf

- Java and Parallel Programming
  - https://www.oracle.com/technical-resources/articles/java/fork-join.html