COMP 322: Fundamentals of Parallel Programming

Lecture 13: Parallel Speedup and Amdahl's Law

Mack Joyner and Zoran Budimlić
{mjoyner, zoran}@rice.edu

http://comp322.rice.edu

Preventing Hw #2 GUI Freeze

Put inside button.addActionListener() lambda body:

```
new Thread(() -> {
    launchHabaneroApp(() -> {
      … loadContributorsPar(…) …
    });
}).start();
```

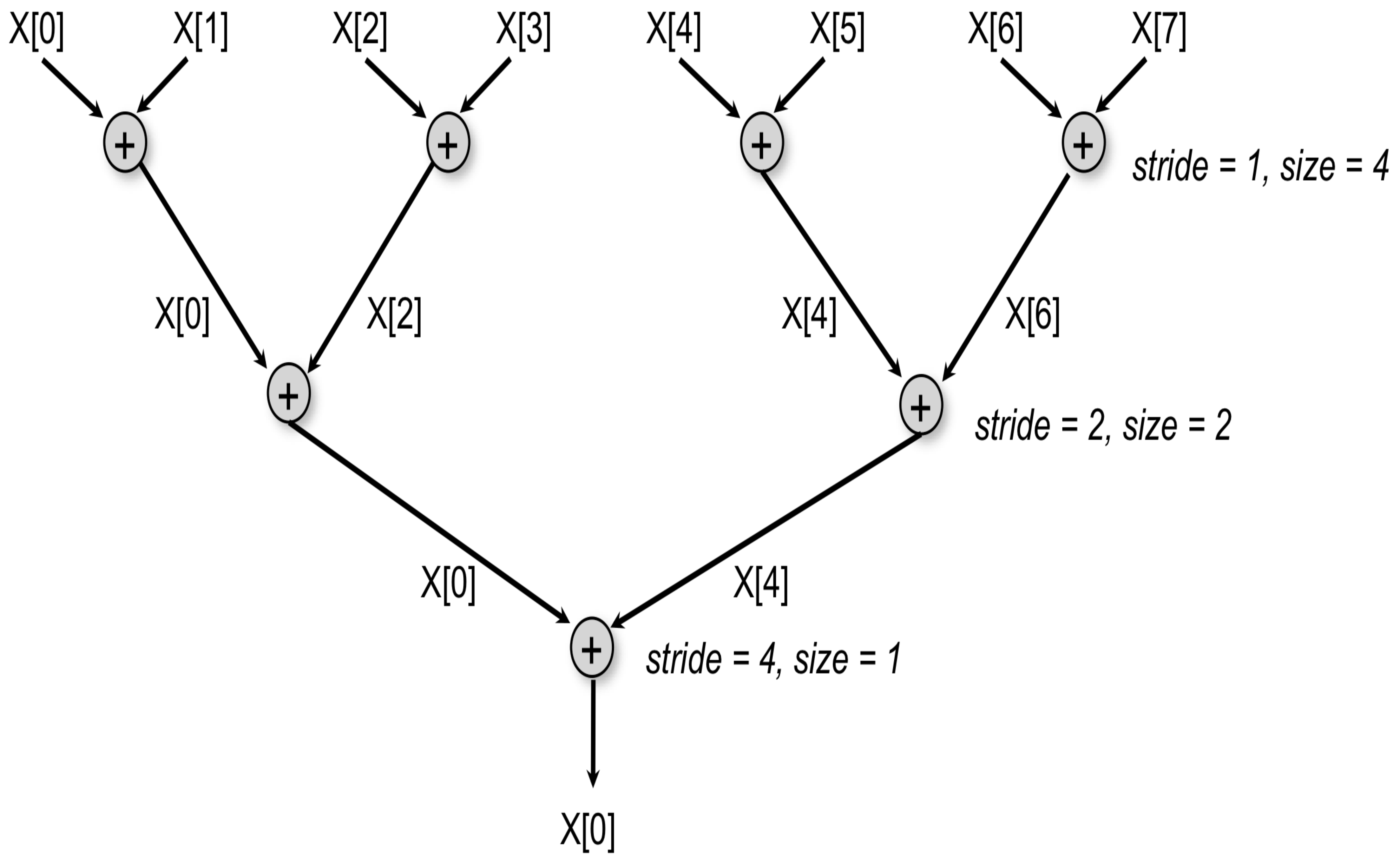
Parallel Speedup

- Define Speedup(P) = T_1 / T_P
  - —Factor by which P processors speeds up execution time relative to 1 processor, for fixed input size
  - —For ideal executions without overhead, 1 <= Speedup(P) <= P
    - —You see this with abstract metrics, but bounds may not hold when measuring real execution times with real overheads
  - —Linear speedup
    - – When Speedup(P) = k*P, for some constant k, 0 < k < 1

- Ideal Parallelism = WORK / CPL = T_1 / T_∞

  = Parallel Speedup on an unbounded (infinite) number of processors

Computation Graph for Recursive Tree approach to computing Array Sum in parallel



Assume greedy schedule, input array size S is a power of 2, each add takes 1 time unit

- WORK(G) = S-1, and CPL(G) = log2(S)
- Define T(S,P) = parallel execution time for Array Sum with size S on P processors
- Use upper bound T(S,P) <= WORK(G)/P + CPL(G) as a worst-case estimate

T(S,P) = WORK(G)/P + CPL(G) = (S-1)/P + log2(S) ⇒ Speedup(S,P) = T(S,1)/T(S,P) = (S-1)/((S-1)/P + log2(S))

How many processors should we use?

Define Efficiency(P) = Speedup(P)/ P = $T_1/(P * T_P)$

- —Processor efficiency --- figure of merit that indicates how well a parallel program uses available processors
- —For ideal executions without overhead, 1/P <= Efficiency(P) <= 1
- —Efficiency(P) = 1 (100%) is the best we can hope for

How many processors should we use?

What should be the minimum efficiency to determine how many processors we should use?

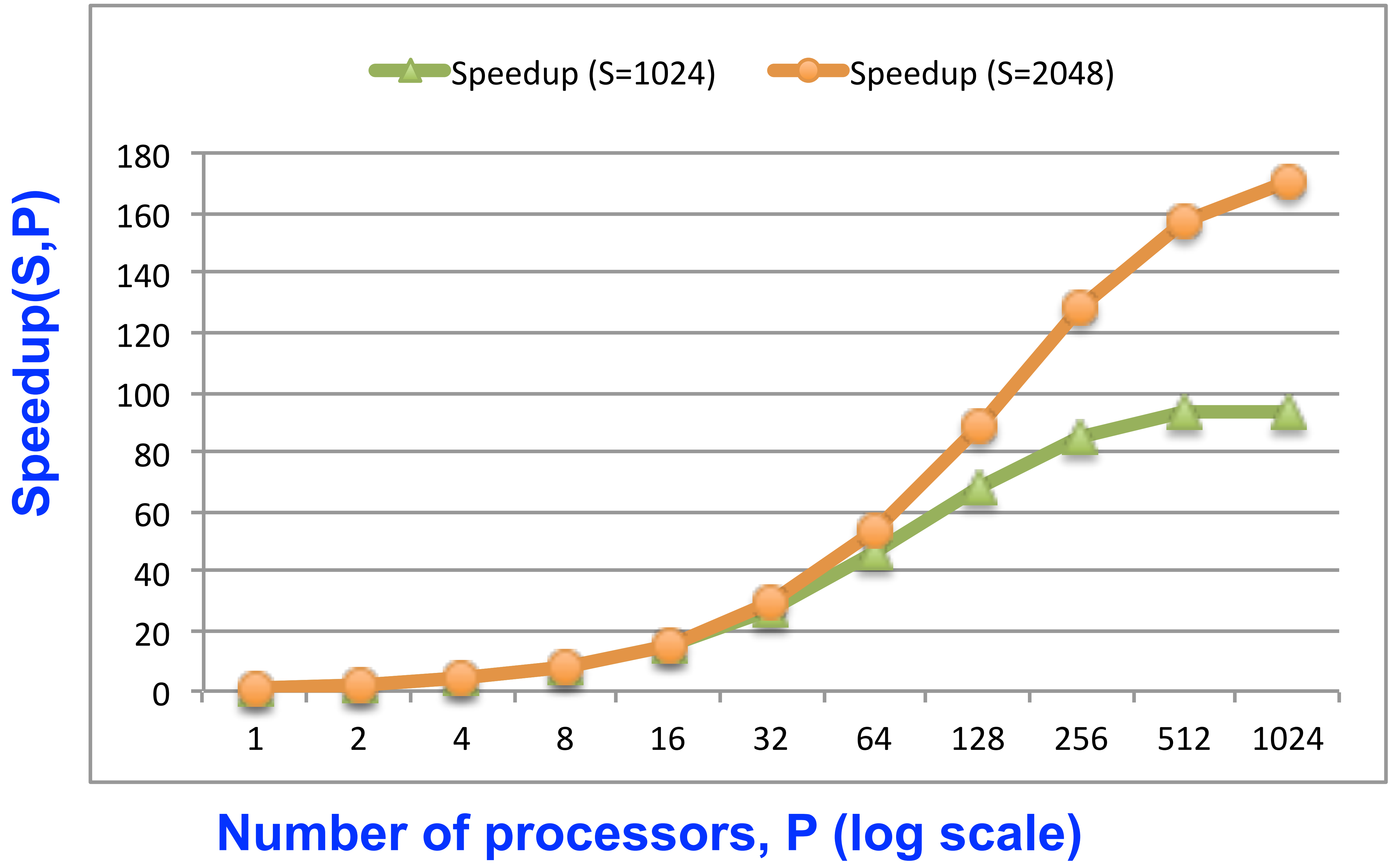
How many processors should we use?

- Common goal: choose number P for a given input size, S, so that efficiency is at least 0.5 (50%)

- Half-performance metric
  - —$S_{1/2}$ = input size that achieves Efficiency(P) = 0.5 for a given P
  - —Figure of merit that indicates how large an input size is needed to obtain efficient parallelism
  - —A larger value of $S_{1/2}$ indicates that the problem is harder to parallelize efficiently

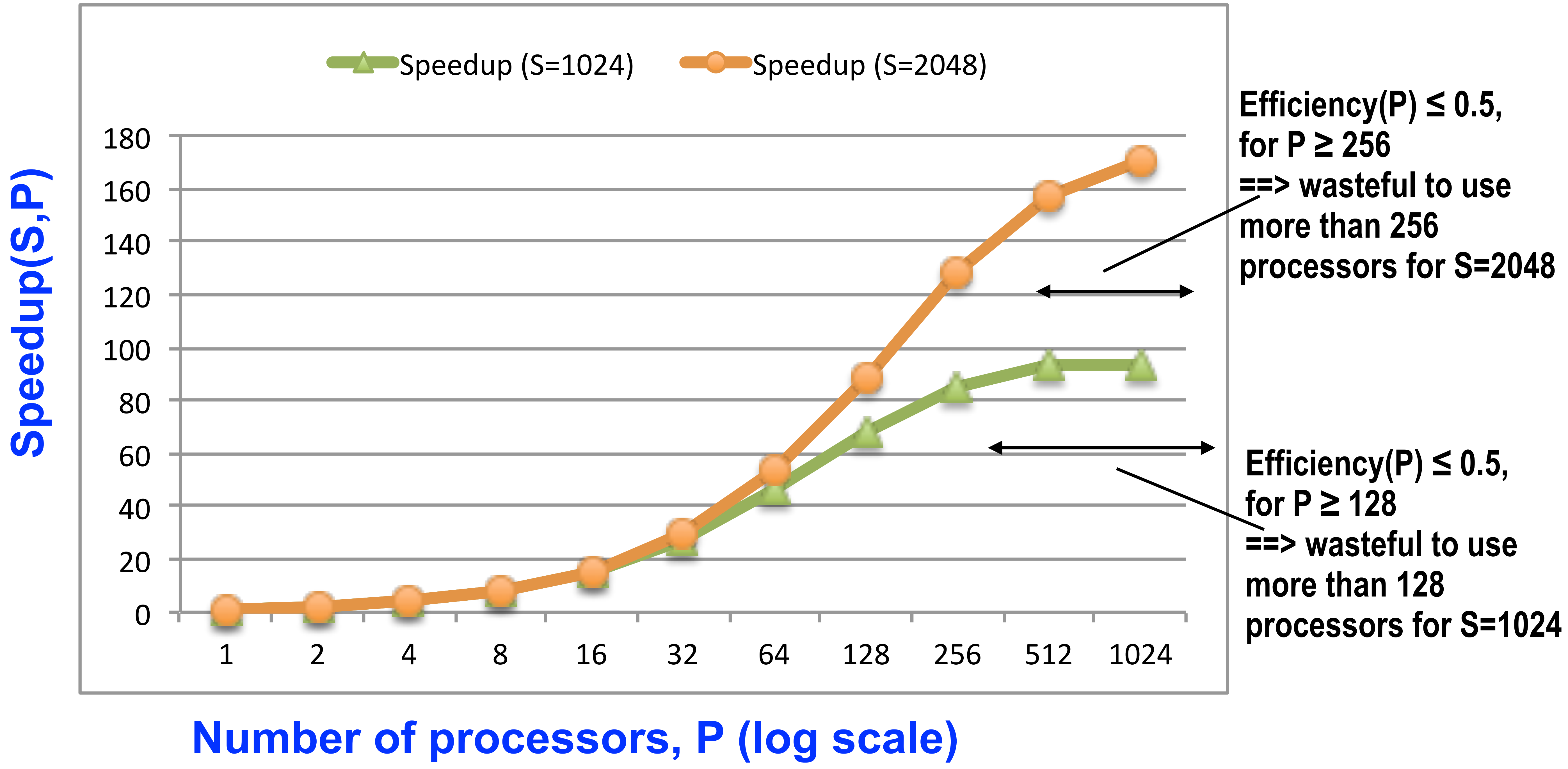
Array Sum: Speedup as a function of array size S and number of processors P

- Speedup(S,P) = T(S,1)/T(S,P) = (S-1)/((S-1)/P + $\log_2$(S))
- Asymptotically, Speedup(S,P) →(S-1)/$\log_2$S, as P → infinity



Array Sum: Speedup as a function of array size S and number of processors P

- Speedup(S,P) = T(S,1)/T(S,P) = (S-1)/((S-1)/P + $\log_2$(S))
- Asymptotically, Speedup(S,P) → (S-1)/$\log_2$S, as P → infinity



Amdahl's Law

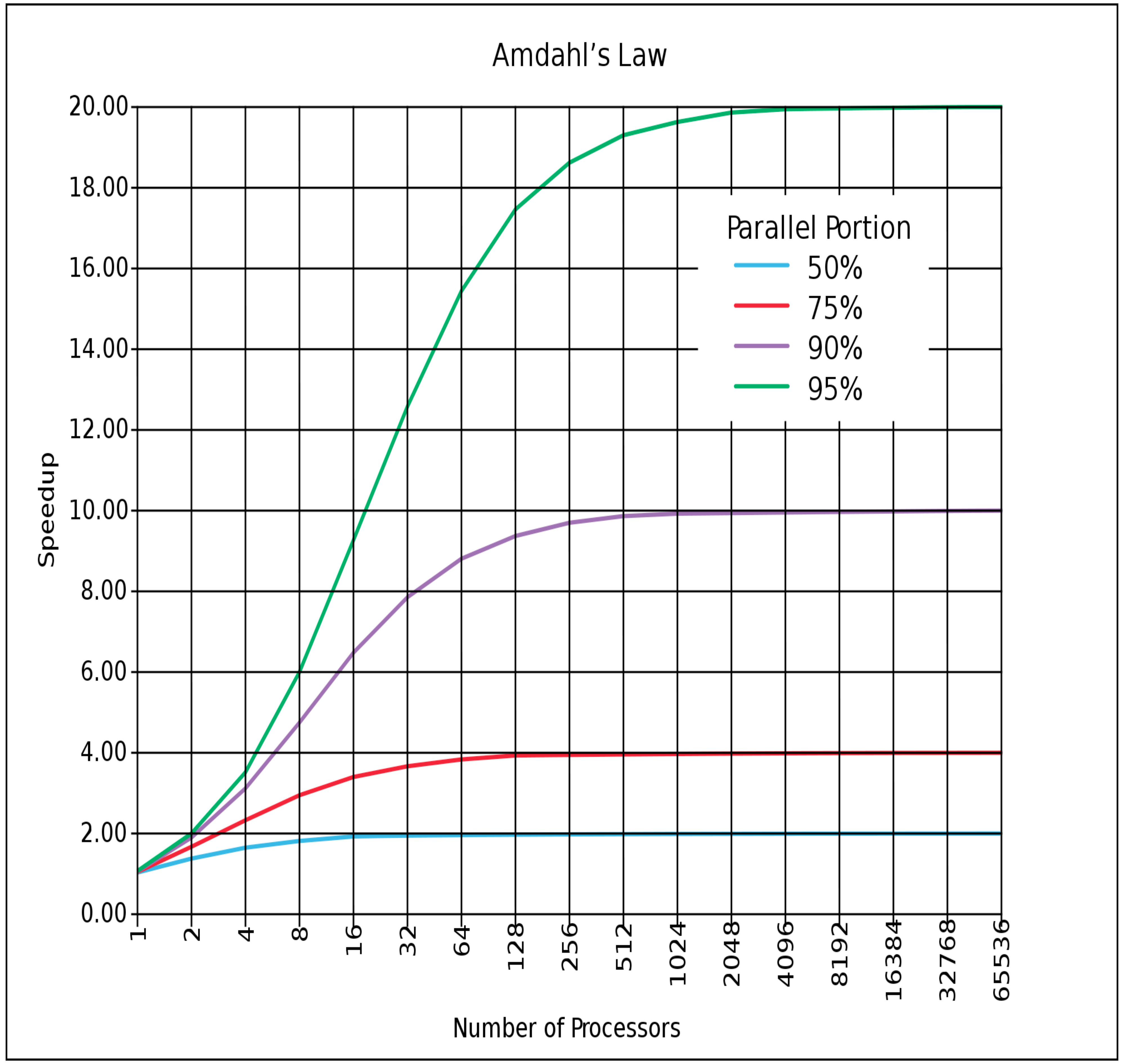
If $q \leq 1$ is the fraction of WORK in a parallel program that must be executed sequentially for a given input size S, then the best speedup that can be obtained for that program is $Speedup(S,P) \leq 1/q$.

Amdahl's Law

- Observation follows directly from critical path length lower bound on parallel execution time
  - CPL >= q * T(S,1)
  - T(S,P) >= q * T(S,1)
  - Speedup(S,P) = T(S,1)/T(S,P) <= 1/q

- Upper bound on speedup simplistically assumes that work can be divided into sequential and parallel portions
  - Sequential portion of WORK = q
    - also denoted as f_S (fraction of sequential work)
  - Parallel portion of WORK = 1-q
    - also denoted as f_p (fraction of parallel work)

Illustration of Amdahl's Law: Best Case Speedup as function of Parallel Portion



Announcements & Reminders

- Quiz #3 is due Tuesday, Feb. 15th at 11:59pm