Measuring Effectiveness

Important to answer this:

How Scalable is a Parallel Program?

Speedup and Efficiency

- See pp. 1013 -1020 (section 12.6) in your book
  - pp. 1049 – 1056 of international version
- Speedup:

$S_p = T_s / T_p$

T_s = time for sequential program
Time of sequential version
Gives us true speedup
We could use time of parallel version running on one core
Gives us relative speedup
T_p = time for parallel program

Speedup and Efficiency

Speedup, $S_p = T_s / T_p$

Efficiency helps us measure the overhead

$E_p = S_p / p = T_s / (p * T_p)$

Efficiency near 1, or 100% is ideal; usually above 75% is fairly good

p is number of processes, or threads used in parallel program

“Trapezoidal Rule example” activity linked on workshop program site

- Direct that for loops should split their work
  - omp parallel for
- Indicate what data is shared by all threads, what is private to each thread (each thread has its own copy)
  - shared()
  - private()
- Use provided function to set threads:
  - omp_set_num_threads(threadct);

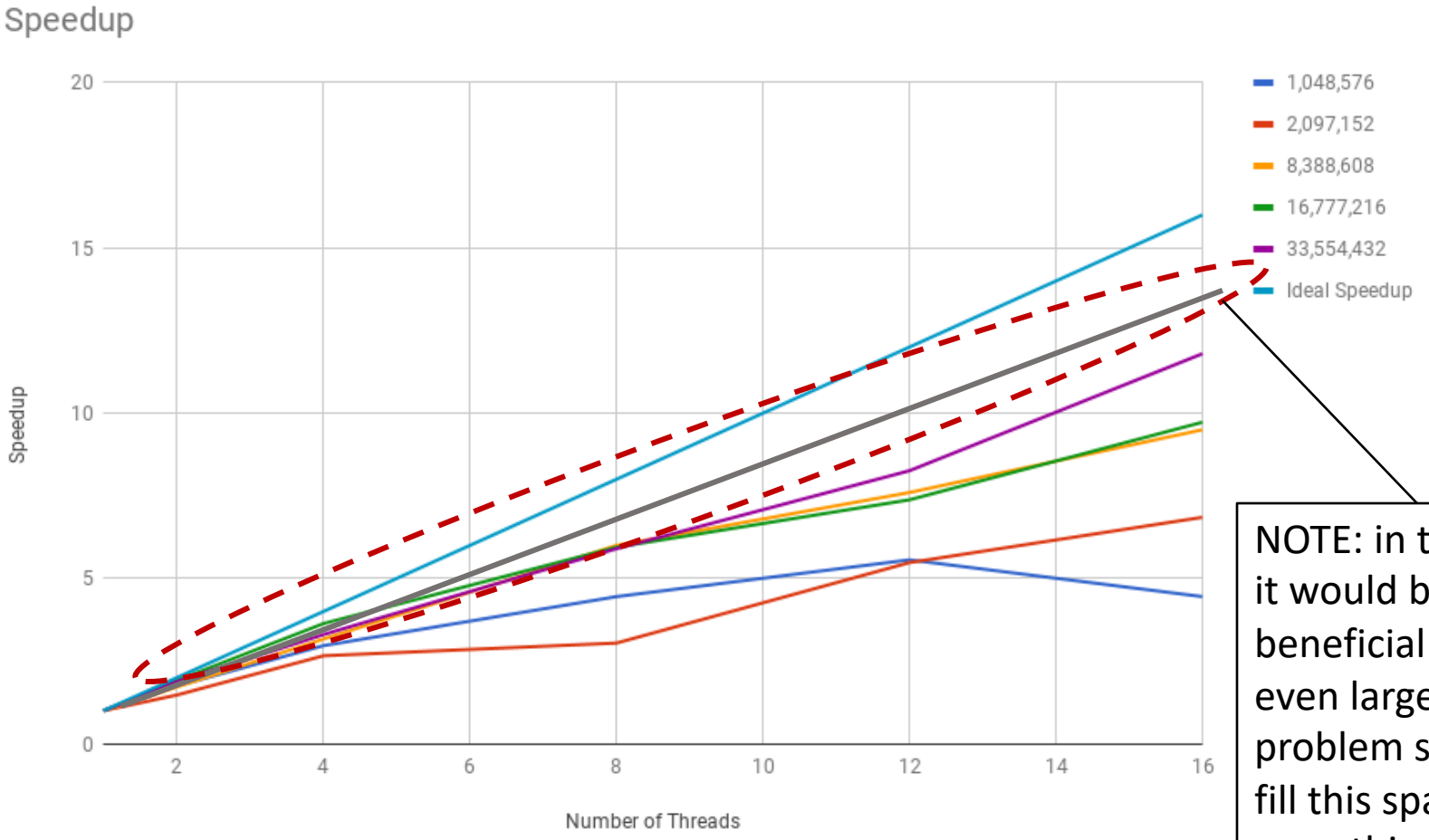
```
#pragma omp parallel for default(none) \
     shared (…) private(…) reduction(…)
      for(i = 1; i < n; i++) {
        integral += f(a+i*h);
      }
```

Need to finish this
integral cannot be shared!
Why?

Experimental process

- Use ‘shell scripts’ to execute code several times
  - duplicate experiments
  - run several different conditions
- Transfer data to spreadsheet
  - analyze using different measurements
    - speedup and efficiency
      - for strong scalability
    - run times as problem size and number of threads are increased proportionally
      - for weak scalability

Trapezoidal rule code speedup

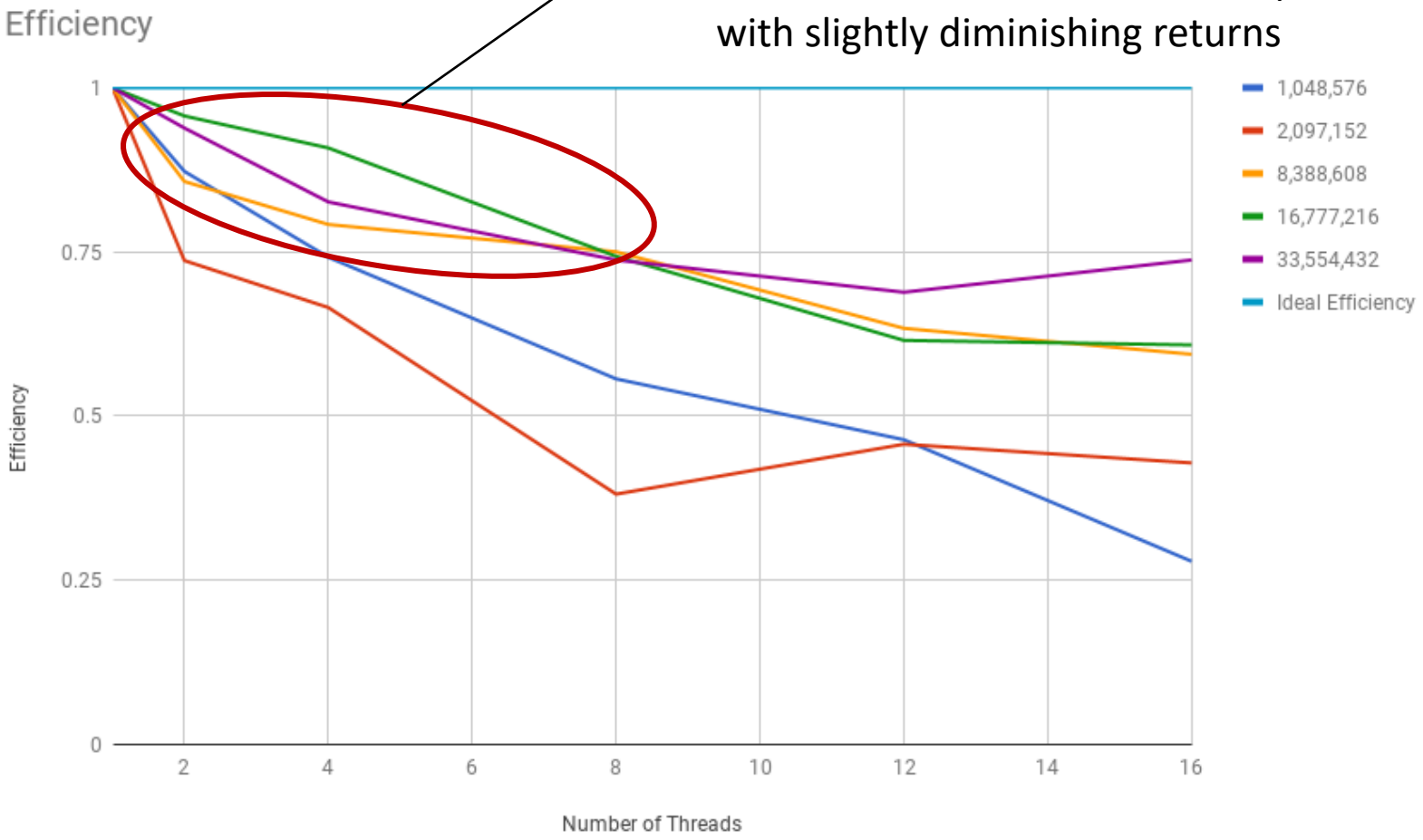


NOTE: in this case, it would be beneficial to try even larger problem sizes to fill this space with something like this black line.

You should have results similar to this
HOWEVER, on the cloud VM, we don't see quite the same performance

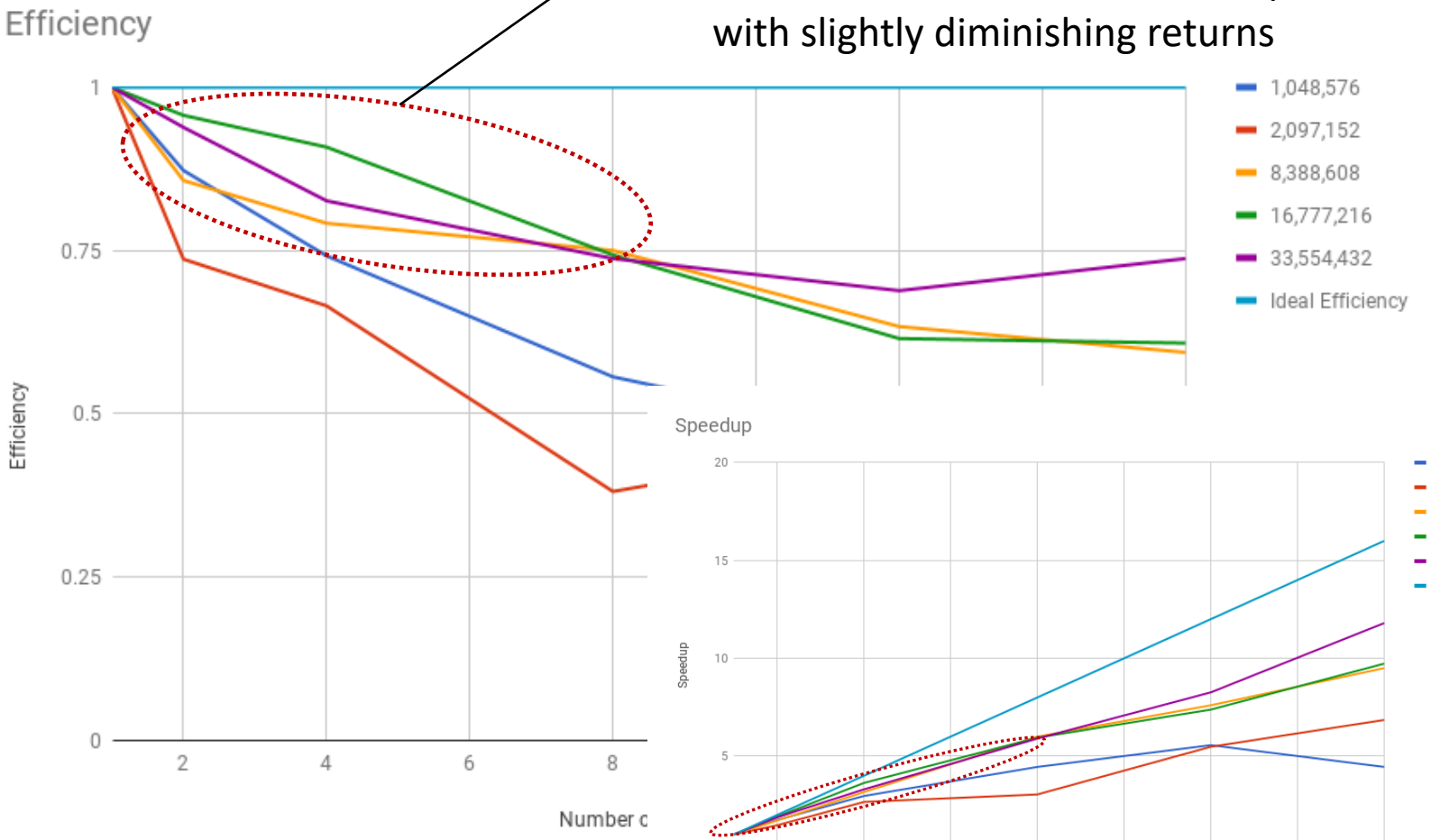
Trapezoidal rule code efficiency

This area of performance is good and is showing fairly strong scalability.
Note how it matches the ideal speed line with slightly diminishing returns



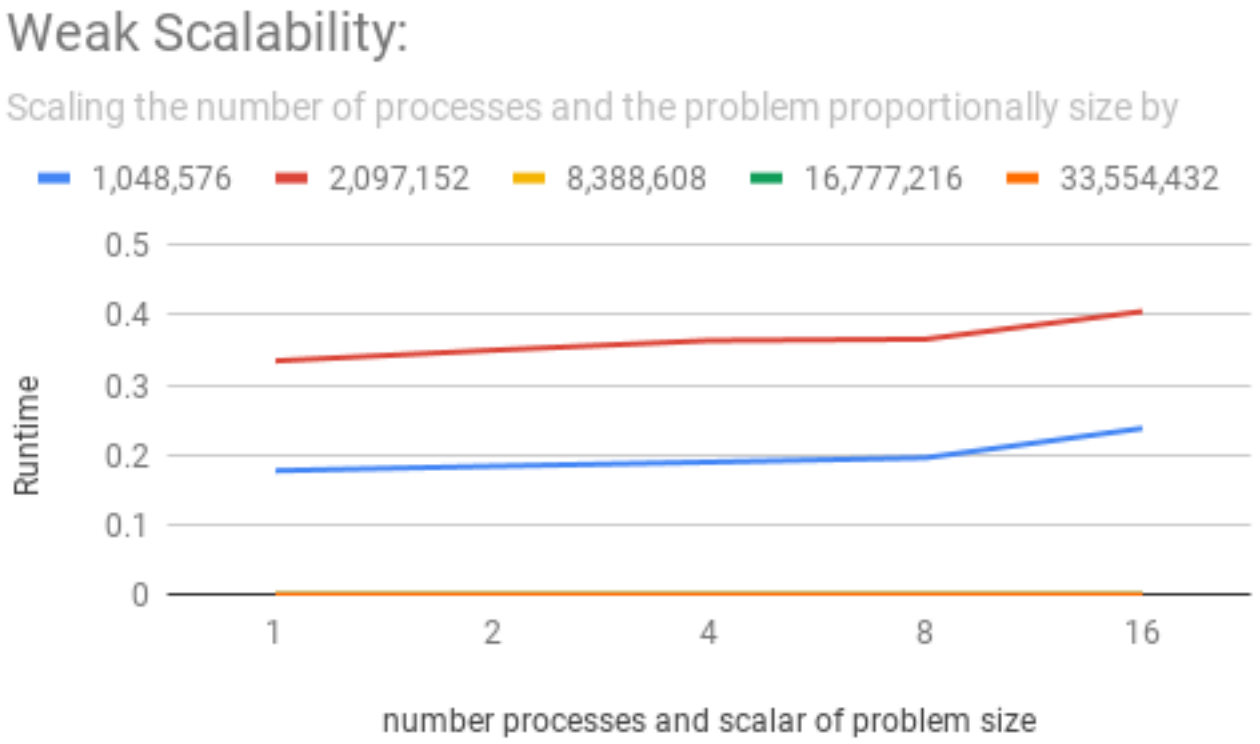
Trapezoidal rule code efficiency

This area of performance is good and is showing fairly strong scalability.
Note how it matches the ideal speed line with slightly diminishing returns



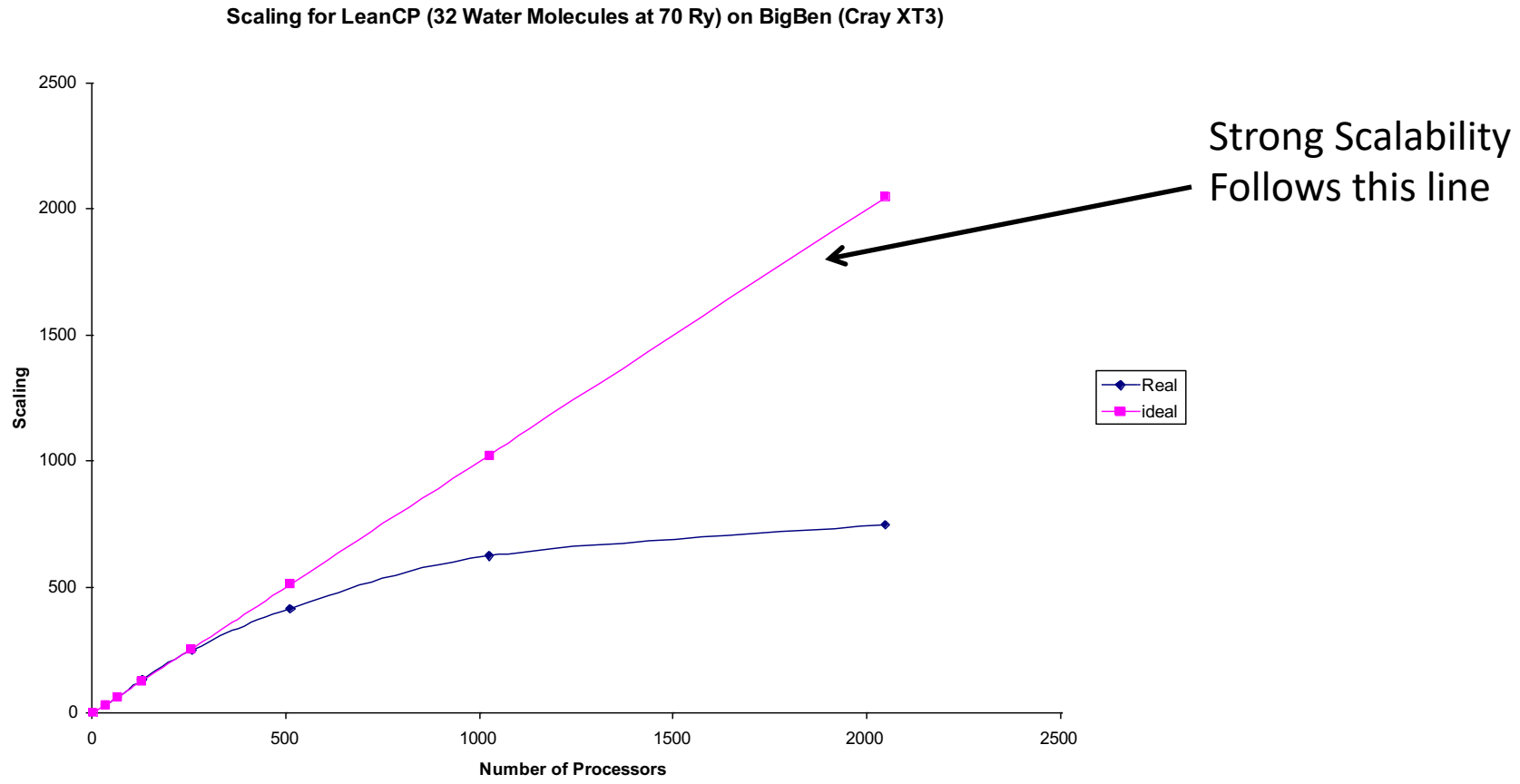
Weak Scalability is also sometimes desirable

- When increasing the problem size by some factor, increasing the the number of processors proportionately keeps the running time nearly constant
  - Also the efficiency therefore remains the same



Speedup and scaling: real-world

- Speedup = T_s / T_p
- For a given (usually large) problem size, measure this for various numbers of processors. An example of a real-world chemistry modeling problem:



More details

Amdahl's "law"

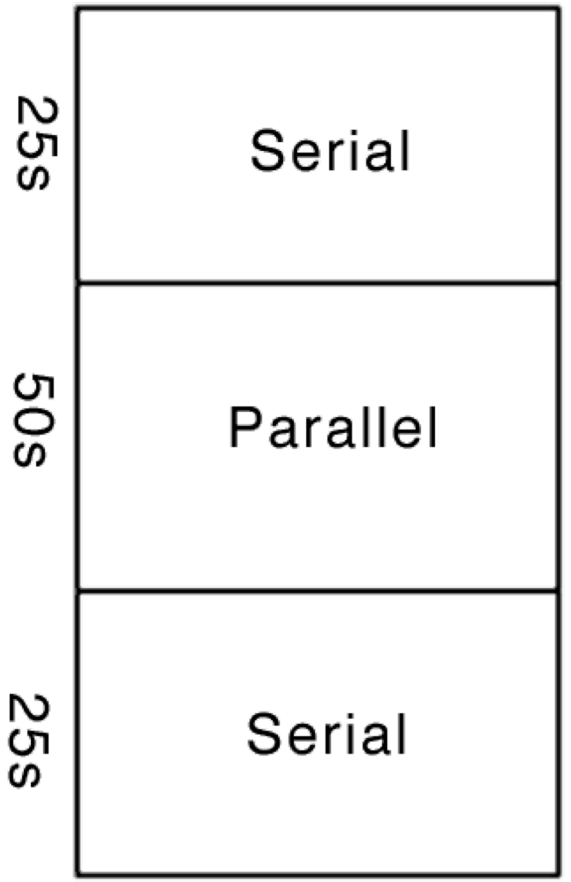
Parts of most code are ‘sequential’

- Let’s observe our simple trapezoidal code example. Sequential parts:
  - Setup was needed
  - ‘reduction’ needed to sum up the final integral
  - Post-processing to report/record results
- Many codes follow this kind of pattern
  - Data-parallel in nature
- Task-parallel services might not (e.g. web server that spawns threads for each request)

Your Scaling Enemy: Amdahl's Law

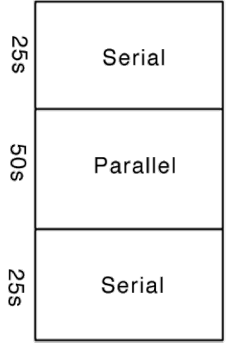
How many processors can we really use?

Let's say we have a legacy code such that is it only feasible to convert half of the heavily used routines to parallel:

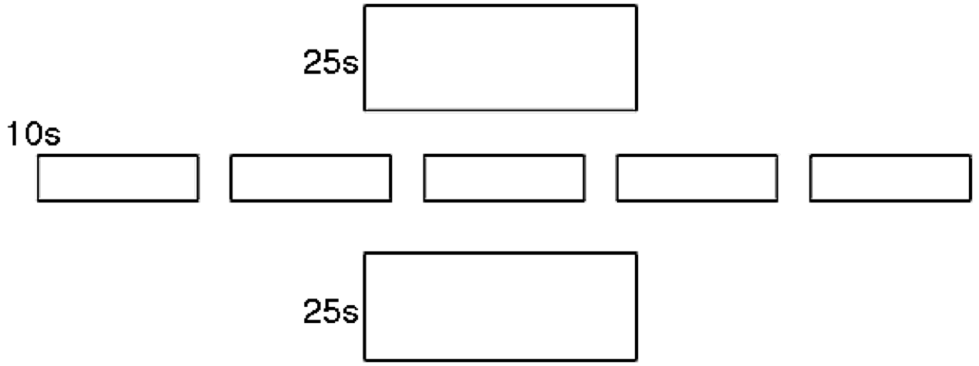


Amdahl's Law

Original:

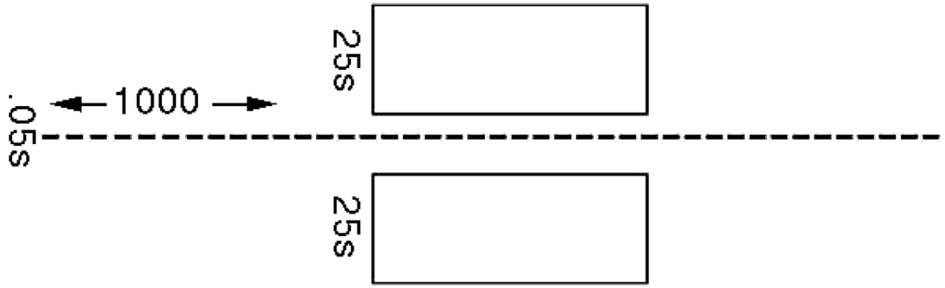


If we run this on a parallel machine with five processors:



Our code now takes about 60s. We have sped it up by about 40%.

Let's say we use a thousand processors:



We have now sped our code by about a factor of two. Is this a big enough win?

Amdahl's Law

- If there is x% of serial component, speedup cannot be better than 100/x.
- If you decompose a problem into many parts, then the parallel time cannot be less than the largest of the parts.
- If the critical path through a computation is T, you cannot complete in less time than T, no matter how many processors you use.