Performance Metrics

Speedup = \frac{\text{(Time on 1 CPU)}}{\text{(Time on } p \text{ CPUs)}}

Efficiency = \frac{\text{speedup}}{p}

What do we compare against?
- parallel program on 1 CPU?
- equivalent sequential program?
- best sequential program?
Example: sorting

• Suppose a parallel bubble sort program obtains 100% efficiency

• Efficiency relative to quicksort:
  – Efficiency = 0 for large N
  – Parallelizing an inefficient sequential algorithm won’t help
How to Cheat with Speedups…

Program with hardware floating point support

Sequential program: 100 sec
Parallel program on 10 CPUs:
  computation: 10 sec
  communication: 10 sec
Speedup = \( \frac{100}{10 + 10} = 5 \)

Without hardware floating point support

Sequential program: 400 sec
Parallel program on 10 CPUs:
  computation: 40 sec
  communication: 10 sec
Speedup = \( \frac{400}{40 + 10} = 8 \)
Absolute performance

• User is interested in how many instructions can be processed per cycle

• Common measure:
  – Floating point instructions per second (FLOPs)

• Compare FLOPS achieved by parallel program against maximum FLOPs for architecture
  – As in TOP-500 for Linpack
Can Speedup be Super linear?

Super linear speedup: speedup > p

Negative search overhead (for search problems)
  Applies to certain input problems, not to average case
  Sequential program could simulate parallel program to get same behavior

Better caching behavior on parallel systems
  Single machine may run out of cache/memory space
  Parallel machine has more cache and memory
Amdahl's Law

Let \( f \) = fraction of code that must be performed sequentially

\[
\text{Speedup} = \frac{T(1)}{T(p)} \leq \frac{T}{(f + (1-f)/p) \times T} = \frac{1}{(f + (1-f)/p) < 1/f}
\]

Example \( f = 10\% \Rightarrow \text{speedup} \leq 10 \), even if \( p \) is infinite!

So much for parallel computing??
Amdahl's Law - the True Story

In practice, $f$ depends on the problem size (Gustafson’s law)

Goal of parallel computing is to solve *large* problems fast

If problem size increases, maximum speedup increases
Weak and strong scaling

• Strong scalability
  – Variation of run time with the number of processors for a fixed problem size
  – More processors -> less work per processor

• Weak scalability
  – Variation of run time with the number of processors for a fixed problem size per processor
  – Increase total problem size with number of processors
How to Measure Performance

Use wall clock time

```c
int start, stop;
start = get_time();
do computations ... 
stop = get_time();
print(stop-start);
```

Exclude initialization

Measures I/O performance

Avoid debugging statements
Performance Debugging

• Try to understand performance behavior
  – Just attributing poor performance to 'communication overhead' is unacceptable

• Determine where the time is spent
  – Measure number of messages per second, data volume per second, idle time

• Optimize the program, for example:
  – Message aggregation: send fewer (larger) messages
  – Latency hiding: overlap communication and computation
  – Load balancing (statically or dynamically)