Measuring Performance

- Given a problem with serial execution time $T$
- What is the ideal execution time with $P$ processors?
Performance

Why might we not achieve a P-fold Speedup?
Performance

Why might we not achieve a P-fold Speedup?

1. Overhead which the sequential computation does not need to pay
2. Non-parallelizable computation
3. Contention for resources
4. Idle processors
1. Overhead

Overhead – any cost that is incurred in the parallel solution but not in the serial solution

4 Sources of Overhead
- Communication
- Synchronization
- Computation
- Memory
2. Non-Parallelizable Code

The Obvious: Inherently sequential code cannot be sped up by increasing the number of processors.

Amdahl’s Law

- Given that $1/S$ of a computation is inherently sequential.
- Given that $Ts$ is the time it takes to solve the whole problem sequentially.
- Then for $P$ processors we have a running time $Tp$, so $Tp = \frac{1}{S} \cdot Ts + \frac{P - 1}{S} \cdot Ts / P$.
- So as $P \to \infty$, $Tp = \frac{1}{S} \cdot T$, i.e. the maximum performance improvement is limited to a factor of $S$. 
2. Non-Parallelizable Code

- **Amdahl’s Law**
  \[ T_p = \frac{1}{S} \cdot T_s + \left(1 - \frac{1}{S}\right) \cdot \frac{T_s}{P} \]

- But it is worse, because parallelized code usually has additional instructions

- So it proves that applying large number of processors to a problem will have limited success.

- Or does it?
2. Non-Parallelizable Code

Amdahl’s Law

\[ T_p = \frac{1}{S} \cdot T_s + (1 - \frac{1}{S}) \cdot \frac{T_s}{P} \]

Amdahl’s Law is stated for a FIXED instance of a problem (keep N constant, increase P), but most parallel computations fix the parallelism and expand the size of the instances (increase N, keep P constant).

Amdahl’s Law reminds us that to achieve parallel performance we must be concerned with the entire program.
3. Contention

- Degradation of system performance caused by competition for a shared resource
- Lock contention
- False sharing – e.g. two local caches contending for the same resource, and the effects are seen by all processors in the form of increased bus traffic
4. Idle time

You can’t fool all the people all the time, and unfortunately, it is difficult to make all the processors work all the time. Often synchronization and communication introduce idle time

- Load imbalance
- Memory-bound computations
  - Bandwidth
  - Latency
Parallel Structure

Understand where the dependences are

Determine the granularity that will match a computation to the underlying hardware

And aim to increase the locality of reference
Dependences

- Provide a general way to describe the limits of parallelism – provide a way to reason about potential sources of performance loss

- Data dependence – an ordering on a pair of memory operations (reads/writes) that must be preserved to maintain correctness
Granularity

The granularity of parallelism is determined by the frequency of interactions among threads or processes.

- Coarse-grained parallelism refers to thread and processes that only infrequently depend on data or events in other processes.

- No fixed granularity is best for all situations.
Locality

- Temporal locality – memory references that are clustered in time
- Spatial locality – memory references that are clustered by address
- In the parallel context, locality has the added benefit of minimizing dependences among threads or processes
Performance Trade-Offs

- Communication vs Computation
  - Overlapping communication and computation
  - Redundant computation

- Memory vs Parallelism
  - Privatization
  - Padding
Performance Trade-Offs

Overhead vs Parallelism

Try to parallelize overhead (like parallelizing accumulation of count of 3s)

Load balance vs Overhead – over-decompose problem into lots of of small problems to ensure balanced load, but also introduce overhead

Granularity trade-offs

Batching is often a good idea

In general, for systems with high communication and synchronization latency, it pays to create course-grained computation that avoids as many interactions as possible while fully exploiting available parallelism