Class Overview:

1. Concepts of parallel structures:
   Dependencies
   Granularity
   Locality
2. Performance metrics:
   Execution Time (latency)
   Speed Up
   Efficiency
3. Reading discussion
4. Task versus data parallelism
   Unlimited / Fixed / Scalable Parallelism
5. Working groups discussion

Concepts of Parallel Structures:
3 topics on how to formally limit overhead.

1. Dependency: ordering relationship between computations.
   Examples:
   • Message passing system: must wait to use data
   • Protect critical region (count 3s example)

   Define a taxonomy by looking at the ordering of read and write operations:
   Data Dependencies:
   • Read after write: flow dependency (“true” dependency)
   • Write after read: anti dependency (“false” dependency)
   • Write after write: output dependency (“false” dependency)
• Read after read: no constraint

Consider the code sequence:
(i) sum = a + 1;
(ii) first_term = sum * scale1;
(iii) sum = b + 1;
(iv) second_term = sum * scale2;
(v) return (first_term + second_term);

Where are the data dependencies:
Flow dependency, sum, from (i) to (ii)
Anti dependency, sum, from (ii) to (iii)
Flow dependency, sum, from (iii) to (iv)
Flow dependency, first_term, from (ii) to (v)
Flow dependency, second_term, from (iv) to (v)

Can we eliminate any of the dependencies?
• There was a proposal to address the flow dependency by replacing the sum variables, for example:
  first_term = (a + 1) * scale1;
  second_term = (b + 1) * scale2;
However, this does not eliminate the “true” dependency. Flow dependencies are part of the logic of the program and typically cannot be removed, although they can be introduced unnecessarily.
• Eliminate the anti dependency (and add a little memory) by changing (iii):
  sum2 = b + 1;
As an example: the SETI@home project has no dependencies - all independent computations.

Dependencies that cross thread or process boundaries can cause problems.

2. Granularity: Measure as the frequency of interactions between units of execution.
   • Coarse grain
     For distributed systems, make the granularity as coarse as possible. For example, the SETI@Home project is very coarse.
   • Fine grain

3. Locality:
   Temporal locality: references to memory clustered in time.
   For example: in count3s accessing the private count.
   Spatial locality: references to memory clustered by address.
   For example: in count3s accessing the assigned chunk array.

Performance Metrics

1. Execution Time (Latency)
   FLOPS (Throughput)

2. SpeedUp = Ts / Tp;
   Where Ts is the sequential time. If the parallel algorithm is significantly different Ts can be replaced with T1, the execution time when executing with a single thread.
Achieving super linear speedup (or speedup better than $T_s / P$):
  • Eliminate memory limitations.
  • Search: depth first search or “branch and bound”, where the thread can stop when it finds an acceptable solution.

3. Efficiency = SpeedUp / $P$

Compute Efficiency Exercise:
Solution: 26 processors
Algorithm:
(1) Split the list of words and send appropriate sublist to each processor
(2) Order each list
(3) Merge each list

(1) and (3) are overhead due to parallelism, 20% of the time, does not change with number of processors.

$T_2 = 0.2 * T_s + T_s / 2 = 0.7 * T_s$

Efficiency = SpeedUp / 2 = ($T_s / T_2$) / 2 = $T_s / (2 * (0.7 * T_s)) = 0.71$