Lecture 3: Multiprocessor Scheduling

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu
1. finish { (F1)
2. async (WV) { Watch COMP 322 video for topic 1.2 by 1pm on Wednesday
3. Watch COMP 322 video for topic 1.3 by 1pm on Wednesday
4. }
5. async (MB) Make your bed
6. async (SF) { Clean out your fridge
7. Buy food supplies and store them in fridge }
8. finish (F2) { async Run load 1 in washer (LW1)
9. Run load 2 in washer (LW2) }
10. async Run load 1 in dryer (LD1)
11. async Run load 2 in dryer (LD2)
12. async Call your family (CF)
13. }
14. Post on Facebook that you’re done with all your tasks! (PF)
Computation Graph Exercise
Observations:
• Any node with out-degree > 1 must be an async (must have an outgoing spawn edge)
• Any node with in-degree > 1 must be an end-finish (must have an incoming join edge)
• Adding or removing transitive edges does not impact ordering constraints

1. A();
2. finish { // F1
3.   async D();
4.   B();
5.   E();
6.   finish { // F2
7.     async H();
8.     F();
9.   } // F2
10.  G();
11. } // F1
12.  C();
Ordering Constraints and Transitive Edges in a Computation Graph

- The primary purpose of a computation graph is to determine if an ordering constraint exists between two steps (nodes).
  - Observation: Node A must be performed before node B if there is a path of directed edges from A and B.

- An edge, $X \rightarrow Y$, in a computation graph is said to be transitive if there exists a path of directed edges from $X$ to $Y$ that does not include the $X \rightarrow Y$ edge.
  - Observation: Adding or removing a transitive edge does not change the ordering constraints in a computation graph.
Ideal Parallelism (Recap)

- Define ideal parallelism of Computation G Graph as the ratio, $\text{WORK}(G)/\text{CPL}(G)$

- Ideal Parallelism only depends on the computation graph, and is the speedup that you can obtain with an unbounded number of processors

Example:
$\text{WORK}(G) = 26$
$\text{CPL}(G) = 11$
Ideal Parallelism = $\text{WORK}(G)/\text{CPL}(G) = 26/11 \approx 2.36$
What is the critical path length of this parallel computation?

1. `finish { // F1`
2. `async A; // Boil water & pasta (10)`
3. `finish { // F2`
4. `async B1; // Chop veggies (5)`
5. `async B2; // Brown meat (10)`
6. `} // F2`
7. `B3; // Make pasta sauce (5)`
8. `} // F1`
Scheduling of a Computation Graph on a fixed number of processors

Node label = time(N), for all nodes N in the graph

Node label = time(N), for all nodes N in the graph

NOTE: this schedule achieved a completion time of 11. Can we do better?
Scheduling of a Computation Graph on a fixed number of processors

• Assume that node N takes TIME(N) regardless of which processor it executes on, and that there is no overhead for creating parallel tasks.

• A schedule specifies the following for each node:
  — $\text{START}(N) =$ start time
  — $\text{PROC}(N) =$ index of processor in range 1...P

such that:
  — $\text{START}(i) + \text{TIME}(i) \leq \text{START}(j)$, for all CG edges from i to j (Precedence constraint)
  — A node occupies consecutive time slots in a processor (Non-preemption constraint)
  — All nodes assigned to the same processor occupy distinct time slots (Resource constraint)
Greedy Schedule

• A greedy schedule is one that never forces a processor to be idle when one or more nodes are ready for execution

• A node is ready for execution if all its predecessors have been executed

• Observations
  — $T_1 = \text{WORK}(G)$, for all greedy schedules
  — $T_\infty = \text{CPL}(G)$, for all greedy schedules

• $T_P(S) = \text{execution time of schedule } S \text{ for computation graph } G \text{ on } P \text{ processors}$
Lower Bounds on Execution Time of Schedules

• Let $T_P$ = execution time of a schedule for computation graph $G$ on $P$ processors
  — $T_P$ can be different for different schedules, for same values of $G$ and $P$

• Lower bounds for all greedy schedules
  — Capacity bound: $T_P \geq \text{WORK}(G)/P$
  — Critical path bound: $T_P \geq \text{CPL}(G)$

• Putting them together
  — $T_P \geq \max(\text{WORK}(G)/P, \text{CPL}(G))$
Upper Bound on Execution Time of Greedy Schedules

Theorem [Graham '66]. Any greedy scheduler achieves

\[ T_P \leq \frac{\text{WORK}(G)}{P} + \text{CPL}(G) \]

**Proof sketch:**
Define a time step to be **complete** if P processors are scheduled at that time, or **incomplete** otherwise

# complete time steps \( \leq \frac{\text{WORK}(G)}{P} \)

# incomplete time steps \( \leq \text{CPL}(G) \)
Bounding the Performance of Greedy Schedulers

Combine lower and upper bounds to get
\[ \max(\text{WORK}(G)/P, \text{CPL}(G)) \leq T_P \leq \text{WORK}(G)/P + \text{CPL}(G) \]

Corollary: Any greedy scheduler achieves execution time \( T_P \) that is within a factor of 2 of the optimal time (since \( \max(a,b) \) and \( (a+b) \) are within a factor of 2 of each other, for any \( a \geq 0, b \geq 0 \)).