

EDA (CS286.5b)

Day 10
Scheduling
(Intro Branch-and-Bound)

Today

- Scheduling
 - Basic problem
 - variants
 - branching
 - bounds
 - and pruning

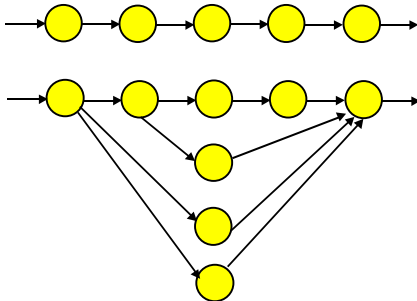
General Problem

- Resources are not free
 - wires, io ports
 - functional units
 - LUTs, ALUs, Multipliers,
 - memory locations
 - memory access ports

Trick/Technique

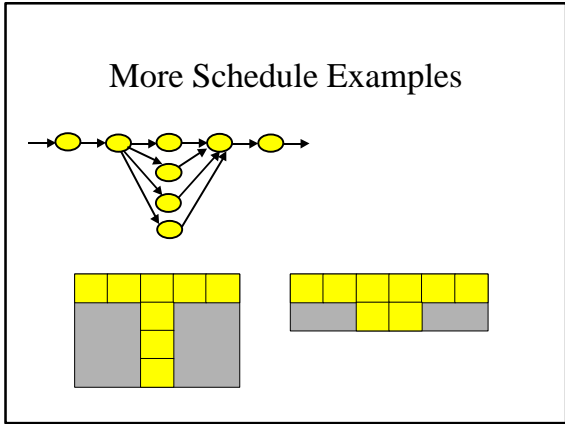
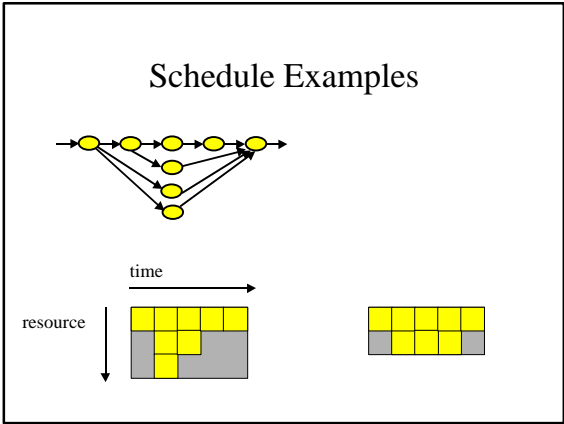
- Resources can be shared (reused) in time
- Sharing resources can reduce
 - instantaneous resource requirements
 - total costs (area)

Example

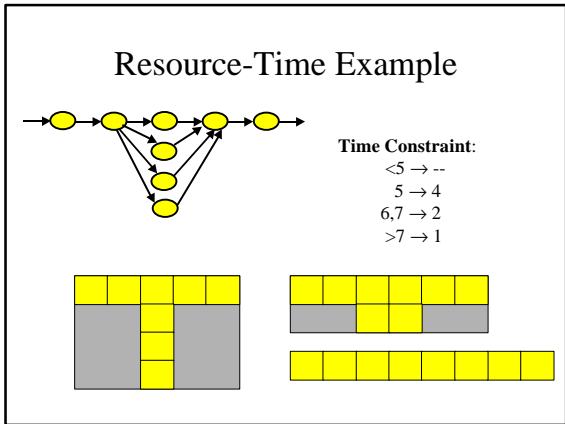


Sharing

- Does not have to increase delay
 - w/ careful time assignment
 - can often reduce peak resource requirements
 - while obtaining original (unshared) delay
- Minimize delay given fixed resources



- ### Scheduling
- **Task:** assign time slots (and resources) to operations
 - **time-constrained:** minimizing peak resource requirements
 - *n.b.* time-constrained, not always constrained to minimum execution time
 - **resource-constrained:** minimizing execution time



- ### Scheduling Use
- Very general problem formulation
 - HDL/Behavioral -> RTL
 - Register/Memory allocation/scheduling
 - Time-Switched Routing
 - TDMA, bus scheduling, static routing
 - Instruction/Functional Unit scheduling
 - Routing (share channel)
 - Processor tasks

- ### Two Types (1)
- **Data independent**
 - graph static
 - resource requirements and execution time
 - independent of data
 - schedule statically
 - maybe bounded-time guarantees
 - typical ECAD problem

Two Types (2)

- **Data Dependent**
 - execution time of operators variable
 - depend on data
 - flow/requirement of operators data dependent
 - if cannot bound range of variation
 - must schedule online/dynamically
 - cannot guarantee bounded-time
 - general case (*i.e.* halting problem)
 - typical “General-Purpose” (non-real-time) OS problem

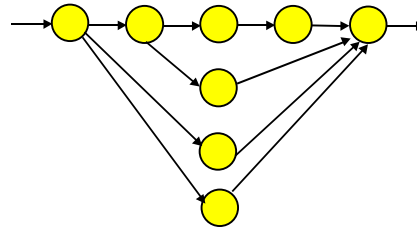
Unbounded Problem

- Easy:
 - compute ASAP schedule
 - *i.e.* schedule everything as soon as predecessors allow
 - will achieve minimum time
 - won't achieve minimum area
 - (meet resource bounds)

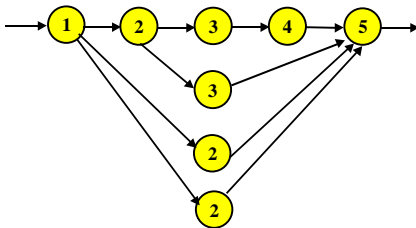
ASAP Schedule

- For each input
 - mark input on successor
 - if successor has all inputs marked, put in visit queue
- While visit queue not empty
 - pick node
 - update time-slot based on latest input
 - mark inputs of all successors, adding to visit queue when all inputs marked

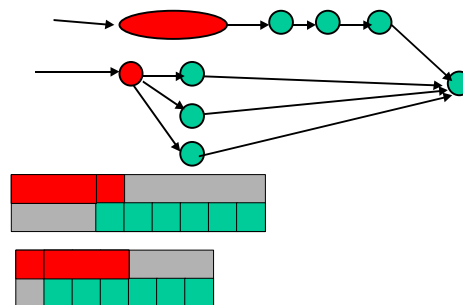
ASAP Example



ASAP Example



Why hard?



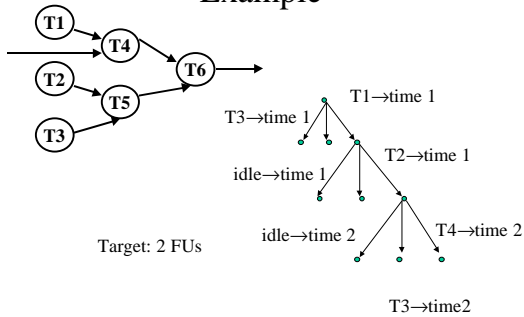
General

- When selecting, don't know
 - need to tackle critical path
 - need to run task to enable work
- Can generalize example to single resource case

Brute-Force

- Try all schedules
- Branching/Backtracking Search
- Start w/ nothing schedule (ready queue)
- At each move (branch) pick:
 - available resource time slot
 - ready task (predecessors completed)
 - schedule task on resource

Example



Branching Search

- Explores entire state space
 - finds optimum schedule
- Exponential work
 - $O(N^{\text{resources} \times \text{time-slots}})$
- Many schedules completely uninteresting

Reducing Work

- Canonicalize “equivalent” schedule configurations
- Identify “dominating” schedule configurations
- Prune partial configurations which will lead to worse (or unacceptable results)

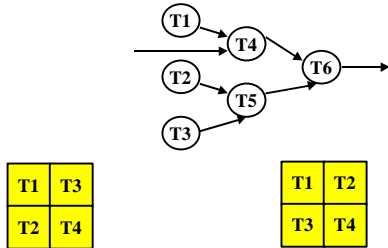
“Equivalent” Schedules

- If multiple resources of same type
 - assignment of task to particular resource at a particular timeslot is not distinguishing

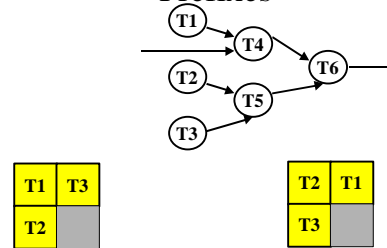


Keep track of resource usage by capacity at time-slot.

“Equivalent” Schedule Prefixes



“Non-Equivalent” Schedule Prefixes

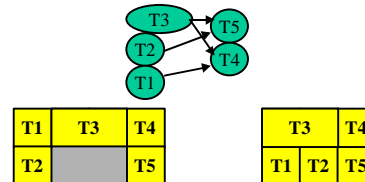


Pruning Prefixes?

- Not sure there is an efficient way (general?)
- Keep track of schedule set
 - walk through state-graph of scheduled prefixes
 - unfortunately, set is power-set so 2^N

Dominant Schedules

- A strictly shorter schedule
 - scheduling the same or more tasks
 - will always be superior to the longer schedule



Pruning

- If can establish a particular schedule path will be worse than one we’ve already seen
 - we can discard it w/out further exploration
- In particular:
 - current schedule time + lower_bound_estimate
 - if greater than existing solution, prune

Pruning Techniques

- Establish Lower Bound on schedule time
- Critical Path (ASAP schedule)
- Aggregate Resource Requirements
- Critical Chain

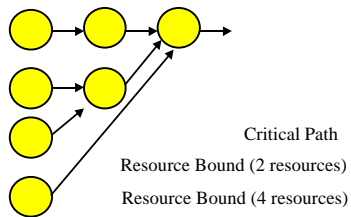
Critical Path Lower Bound

- ASAP schedule ignoring resource constraints
 - (look at length of remaining critical path)
- Certainly cannot finish any faster than that

Resource Capacity Lower Bound

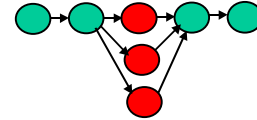
- Sum up all capacity required per resource
- Divide by total resource (for type)
- Lower bound on remaining schedule time
 - (best can do is pack all use densely)

Example



“Critical Chain” Lower Bound

- Bottleneck resource present coupled resource and latency bound



Single red resource

Lower Bound Pruning

- Typically
 - make move (schedule)
 - recalculate lower bound
 - if best case completion greater than current best
 - prune branch

Search Ordering

- Performance sensitive to ordering
 - *e.g.* if find best path first, easy to prune
 - paths in order of decreasing length, cannot
- Depth first
 - get complete time for bound
 - may find long paths first
- Breadth first
 - don't have complete path for bound
 - lots of intermediate data to keep
 - ultimately finds short paths before long

Search Ordering

- Mixture of depth and breadth
- Use heuristics to select nodes for expansion
- If heuristics help find good solutions
 - helps prune
 - focus attention in search

Alpha-Beta Search

- Generalization
 - keep both upper and lower bound estimates on partial schedule
 - expand most promising paths
 - (least upper bound, least lower bound)
 - prune based on lower bounds exceeding known upper bound
 - (technique typically used in games/Chess)

Alpha-Beta

- Each scheduling decision will tighten
 - lower/upper bound estimates
- Can choose to expand
 - least current time (breadth first)
 - least lower bound remaining (depth first)
 - least lower bound estimate
 - least upper bound estimate
- can control greediness
 - weighting lower/upper bound
 - selecting “most promising

Upper Bounds

- Determine upper bounds by heuristics/approximations
 - talk about next time

Note

- Aggressive pruning and ordering
 - can sometimes make polynomial time in practice
 - often cannot *prove* will be polynomial time
 - usually represents problem structure we still need to understand
- Not sure we’ve covered enough techniques today to make scheduling (in general) polynomial time in practice

Adapt Time Constrained

- Vary resources and run as-is
- Binary search for minimum resources achieving time target
- Use lower bounds to define region of interest to search
- Use lower bounds to prune/exit search early if won’t meet target

Multiple Resources

- Works for multiple resource case
- Computing lower-bounds per resource
 - resource constrained
- Sometimes deal with resource coupling
 - *e.g.* must have 1 A and 1 B simultaneously or in fixed time slot relation
 - *e.g.* bus and memory port

Summary

- Resource sharing saves area
 - allows us to fit in fixed area
- Requires that we schedule tasks onto resources
- General kind of problem arises
- Branch-and-bound search
 - “equivalent” states
 - dominators
 - estimates/pruning

Today’s Big Ideas:

- Exploit freedom in problem to reduce costs
 - (slack in schedules)
- Use dominating effects
 - (constrained resources)
- Use dominators to reduce work
- Technique: Search
 - Branch-and-Bound
 - Alpha-Beta