

EDA (CS286.5b)

Day 11
Scheduling
(List, Force, Approximation)

N.B. no class Thursday (FPGA) ...

Today

- Scheduling
 - List-Scheduling
 - Force-Directed
 - Approximation Algorithms
- Assignment #2
 - recursive partitioning for energy minimization

Last Time

- Resources aren't free
- Share to reduce costs
- Schedule operations on resources
- Greedy not optimal
- Optimal solution with Branch-and-Bound
- Lower Bound Pruning

This Time

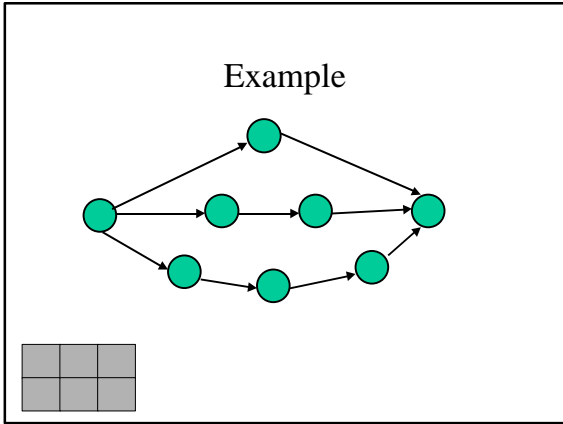
- Heuristic/non-optimal approaches
- Use to solve quickly
- Use to generate upper bounds
 - help pruning in alpha-beta search
- Bounds on “badness” of heuristic

Force-Directed

- **Problem:** how exploit schedule freedom (slack) to minimize instantaneous resources
 - (time constrained)

Force-Directed

- Given a node, can schedule anywhere between ASAP and ALAP schedule time
 - latest schedule predecessor and ALAP
 - ASAP and already scheduled successors
- Scheduling node will limit freedom of nodes in path



Force-Directed

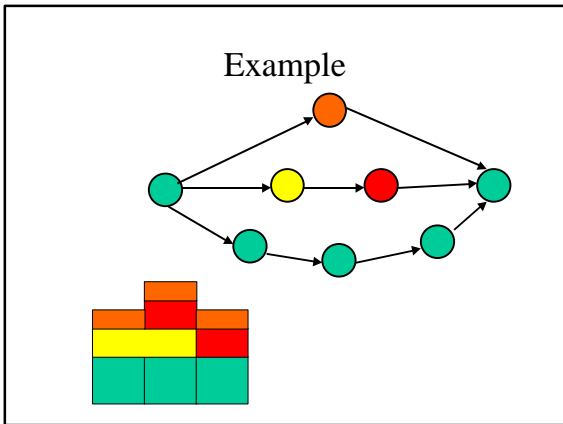
- If everything were scheduled, except for the target node:
 - examine resource usage in all timeslots allowed by precedence
 - place in timeslot which has least increase maximum resources

Force-Directed

- Problem is don't know resource utilization during scheduling
- Strategy: estimate resource utilization

Force-Directed Estimate

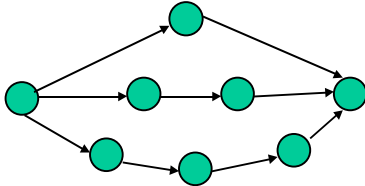
- Assume a node is uniformly distributed within slack region
 - between earliest and latest possible schedule time
- Use this estimate to identify most used timeslots



Force-Directed

- Scheduling a node will shift distribution
 - all of scheduled node's cost goes into one timeslot
 - predecessor/successors may have freedom limited so shift their contributions
- Want to shift distribution to minimize maximum resource utilization

Example



Force-Directed

- ASAP/ALAP schedule to determine range of times for each node
- Compute estimated resource usage
- Pick most constrained node
 - Evaluate effects of placing in feasible time slots (compute forces)
 - Place in minimum cost slot and update estimates
 - Repeat until done

Time

- Evaluate force of putting in timeslot $O(NT)$
- Evaluate all timeslots can put in $O(NT^2)$
- N nodes to place
- $O(N^2T^2)$

List Scheduling

- Keep a ready list of “available” nodes
 - (one whose predecessors have already been scheduled)
- Pick an unscheduled task and schedule on next available resource
- Put any tasks enabled by this one on ready list

List Scheduling

- Greedy heuristic
- How prioritize ready list?
 - Dominant constraint
 - least slack (worst critical path)
 - enables work
 - utilize most precious resource
- Last time:
 - saw that no single priority scheme would be optimal

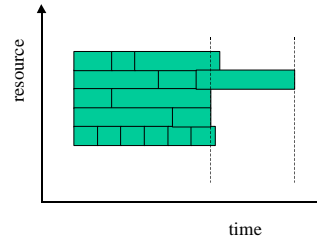
List Scheduling

- Use for
 - resource constrained
 - time-constrained
 - give resource target and search for minimum resource set
- Fast: $O(N) \rightarrow O(N \log(N))$ depending on prioritization
- Simple, general
- How good?

Approximation

- Can we say how close an algorithm comes to achieving the optimal result?

Without Precedence



Observe

- Optimal length L
- No idle time up to start of last job to finish
- start time of last job $\leq L$
- last job length $\leq L$
- Total LS length $\leq 2L$
- Algorithm is within factor of 2 of optimum

Terminology

- Scheduling of identical parallel machines has a 2-approximation
 - *I.e.* we have a polynomial time algorithm which is guaranteed to achieve a result within a factor of two of the optimal solution.
- In fact, for precedence unconstrained there is a $4/3$ -approximation
 - (schedule Longest Processing Time first)

Recover Precedence

- May have idle times, so need to generalize
- Work back from last completed job
 - two cases:
 - entire machine busy
 - some predecessor in critical path is running
- Divide into two sets
 - whole machine busy times
 - critical path chain for this operator

Precedence

- schedule picture

Precedence Constrained

- All busy times $<$ Optimal Length
 - Optimal Length $>$ Resource Bound
 - Resource Bound $>$ All busy
- This path $<$ Optimal Length
 - Optimal Length $>$ Critical Path
 - Critical Path \geq This Path
- List Schedule $\leq 2 \cdot$ (Optimal Length)

Tighten

- LS schedule \in Critical Path + Resource Bound
- LS schedule \in $\text{Min}(\text{CP}, \text{RB}) + \text{Max}(\text{CP}, \text{RB})$
- Optimal schedule \in $3 \cdot \text{Max}(\text{CP}, \text{RB})$
- LS/Opt \in $1 + \text{Min}(\text{CP}, \text{RB}) / \text{Max}(\text{CP}, \text{RB})$

- The more one constraint dominates
 - the closer the approximate solution to optimal

Multiple Resource

- Previous result for homogeneous functional units
- For heterogeneous resources:
 - also a 2-approximation
 - Lenstra+Shmoys+Tardos, Math. Programming v46p259
 - (not online)

Bounds

- No precedence case
 - no polynomial approximation algorithm can achieve better than $4/3$ bound
- Precedence case
 - no polynomial approximation algorithm can achieve better than $3/2$ bound

Other Approaches

- Graph Coloring
- ILP
- Annealing

Summary

- Heuristic approaches to schedule
 - Force-directed
 - List Scheduling
- We can, sometimes, bound the “badness” of a heuristic
 - get a tighter result based on gross properties of the problem
 - approximation algorithms often a viable alternative to finding optimum
 - play role in knowing “goodness” of solution

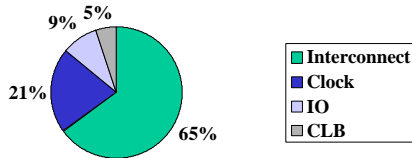
Today's Big Ideas:

- Exploit freedom in problem to reduce costs
 - (slack in schedules)
- Technique: estimation/refinement
- Use dominating effects
 - (constrained resources)
 - the more an effect dominates, the “easier” the problem
- Technique: Approximation

Assignment #2

- Observation:
 - Energy in wires
 - Switching probability varies among nets
- **Hypothesis:** an important part of minimizing energy is to localize highly active notes

Dominant Power



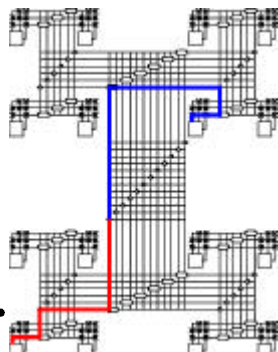
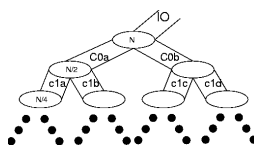
XC4003A data from Eric Kusse (UCB MS 1997)

Assignment #2

- **Goal:** Recursively partition netlist minimizing energy
- Model:
 - $E = \sum (\text{Activity} * (\text{fixed} + F(\text{distance})))$
 - $F(\text{distance}) = \sum C(\text{part_size})$
 - (pull out fixed as unchanging)

Cost Example

$$E = A * (C(L1) + C(L2) + C(L3) + C(L4) + C(L3) + C(L2) + C(L1))$$



Provided

- Parse/represent netlist
- Read activities
- Represent technology (cost at level)
- Represent partitions
- Calculate spectral partitions (io based)
- Calculate cost of partition
- Sample netlists

Your Task

- Devise/implement an algorithm to develop recursive partition
 - class: FM, maxflow, spectral, simulated annealing...

Example

- Show sample use of existing framework...