ESE535: Electronic Design Automation

Day 20: April 7, 2008
Scheduling
Variants and Approaches

Today

- Scheduling
  - Force-Directed
  - SAT/ILP
  - Branch-and-Bound

Last Time

- Resources aren't free
- Share to reduce costs
- Schedule operations on resources
- Greedy approximation algorithm

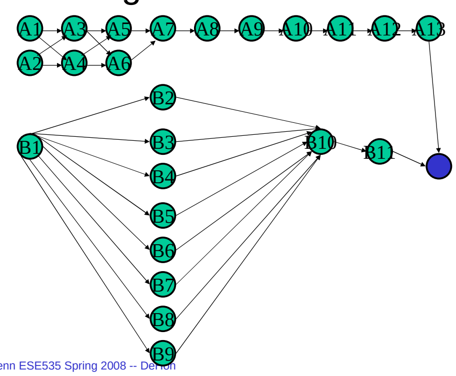
Force-Directed

- **Problem**: how exploit schedule freedom (slack) to minimize instantaneous resources
  - Directly solve time constrained
    - (last time only solved indirectly)
  - Trying to minimize resources

Force-Directed

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

Single Resource Challenge



Force-Directed

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

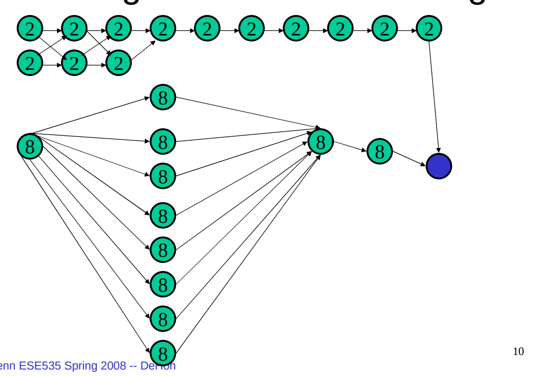
Force-Directed

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

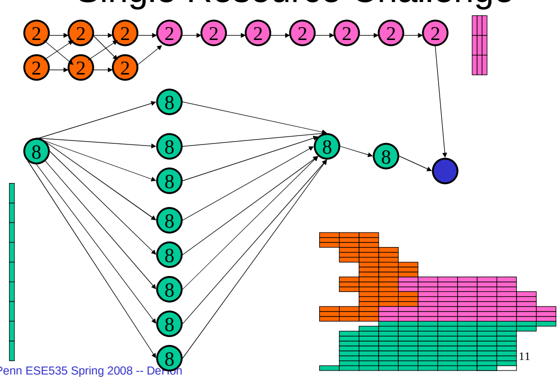
Force-Directed Estimate

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

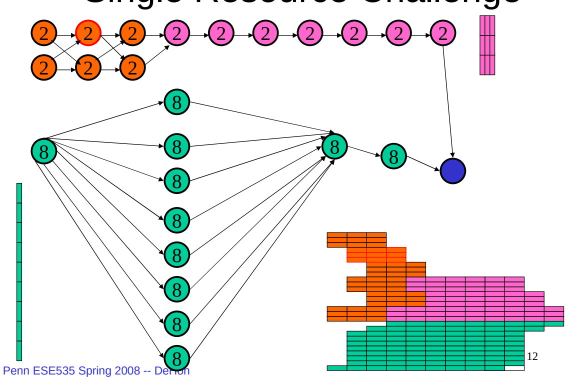
Single Resource Challenge



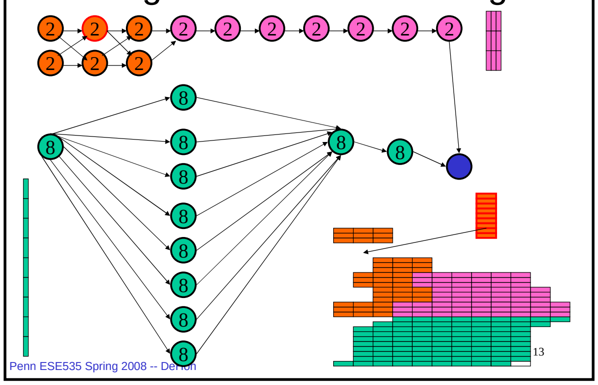
Single Resource Challenge



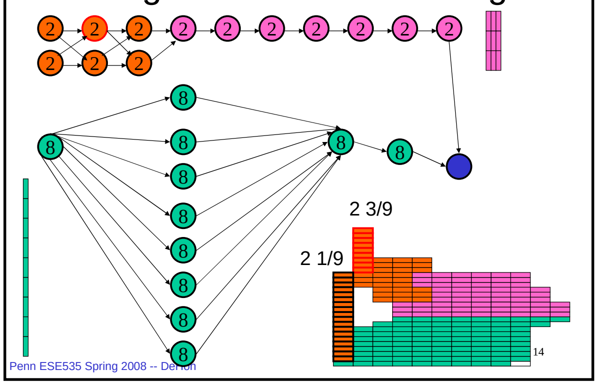
Single Resource Challenge



Single Resource Challenge



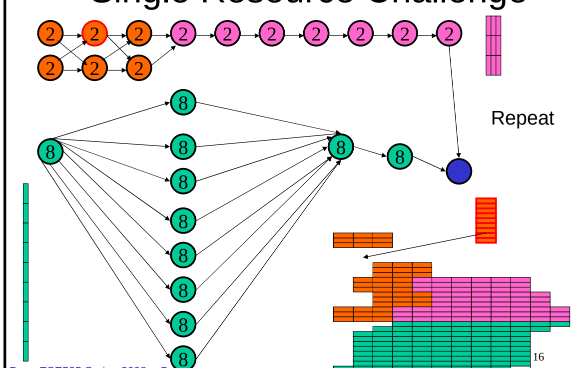
Single Resource Challenge



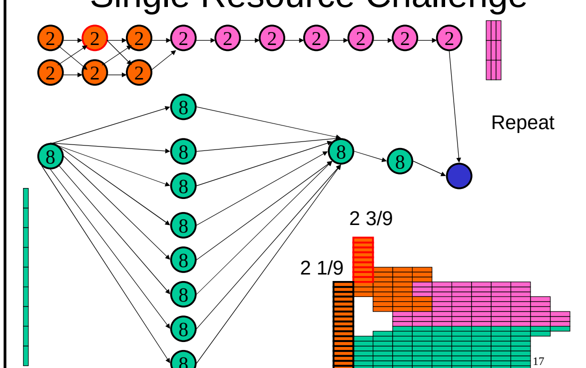
Force-Directed

- 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 (estimate)

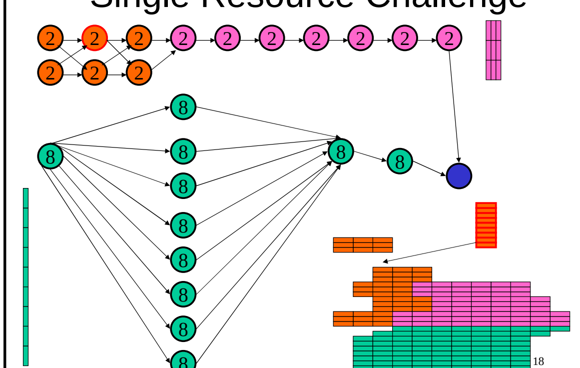
Single Resource Challenge



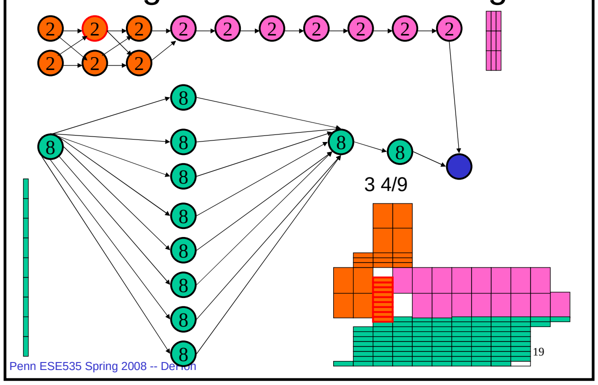
Single Resource Challenge



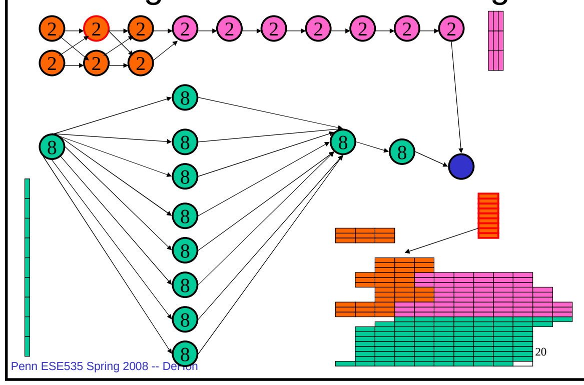
Single Resource Challenge



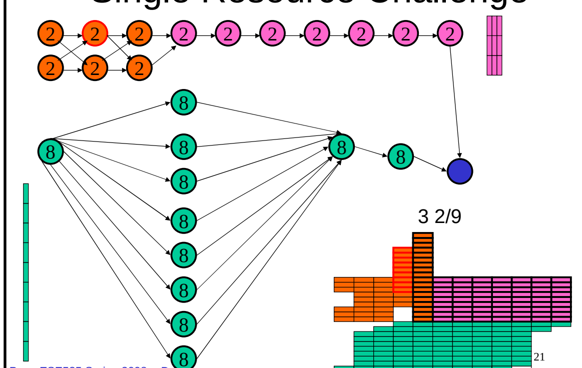
Single Resource Challenge



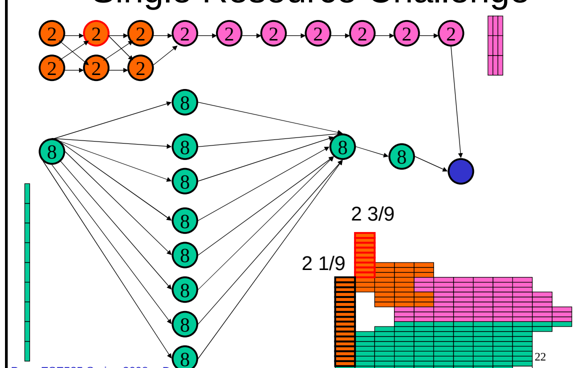
Single Resource Challenge



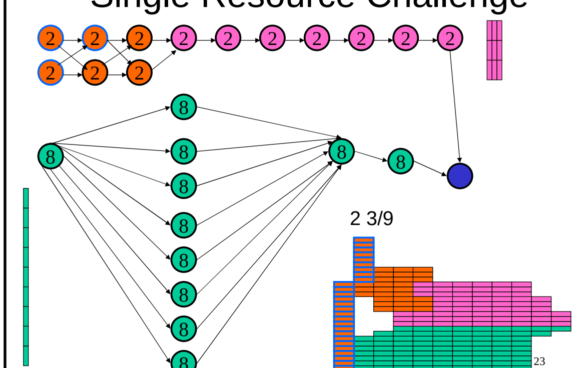
Single Resource Challenge



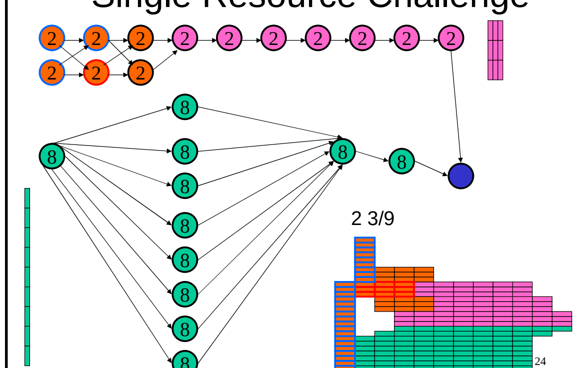
Single Resource Challenge



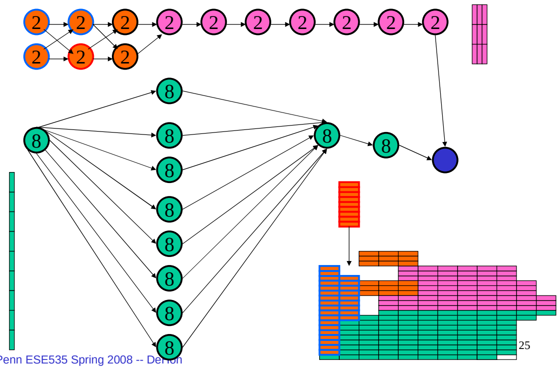
Single Resource Challenge



Single Resource Challenge

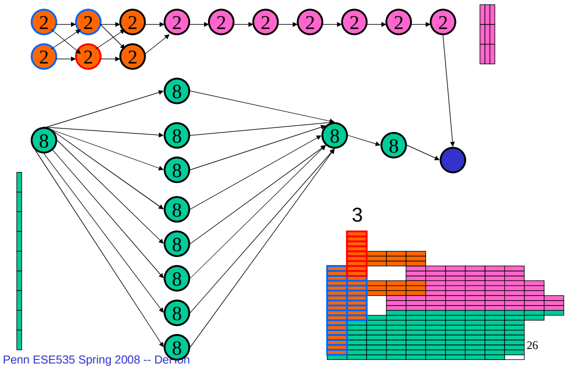


Single Resource Challenge



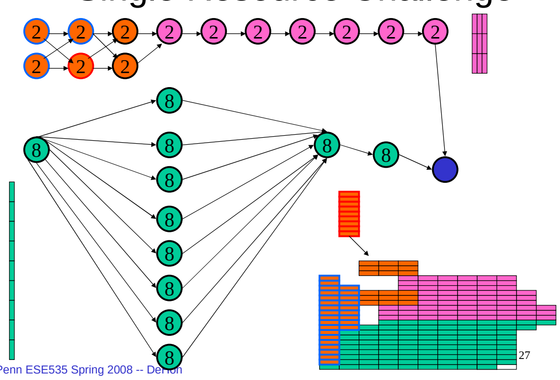
Penn ESE535 Spring 2008 -- DeHon 25

Single Resource Challenge



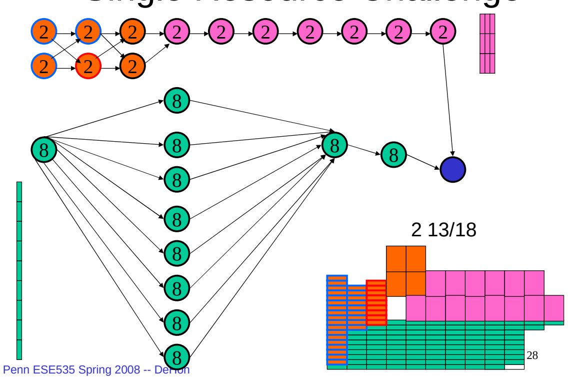
Penn ESE535 Spring 2008 -- DeHon 26

Single Resource Challenge



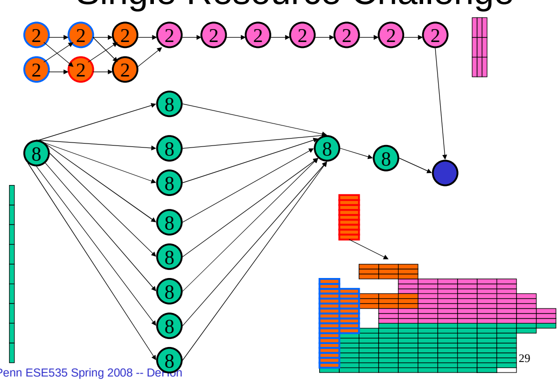
Penn ESE535 Spring 2008 -- DeHon 27

Single Resource Challenge



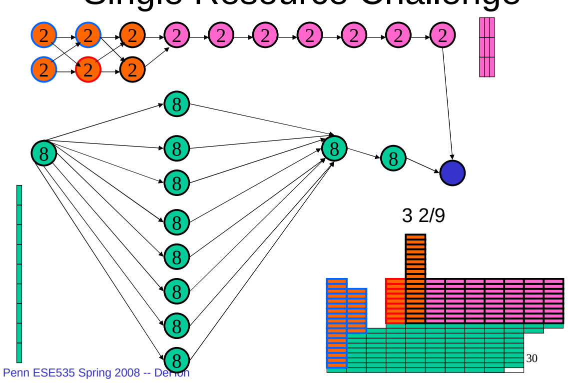
Penn ESE535 Spring 2008 -- DeHon 28

Single Resource Challenge



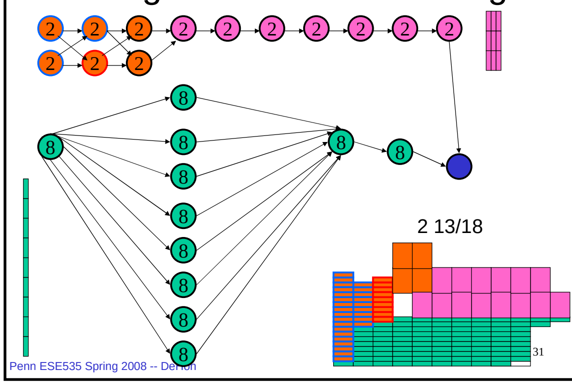
Penn ESE535 Spring 2008 -- DeHon 29

Single Resource Challenge

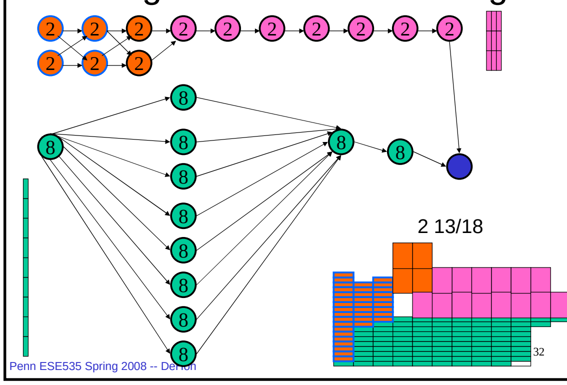


Penn ESE535 Spring 2008 -- DeHon 30

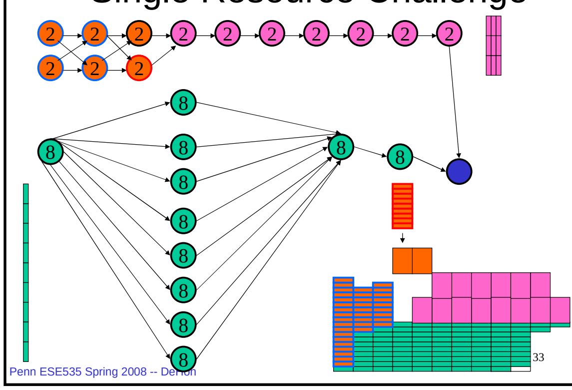
Single Resource Challenge



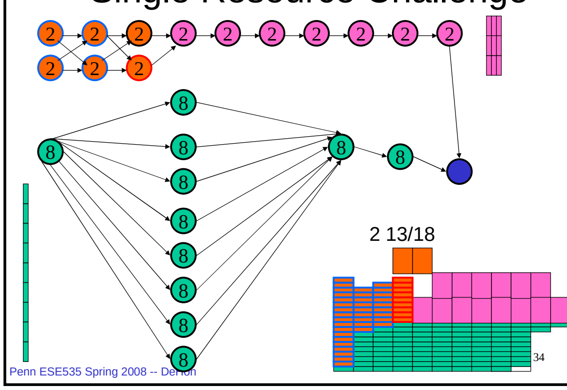
Single Resource Challenge



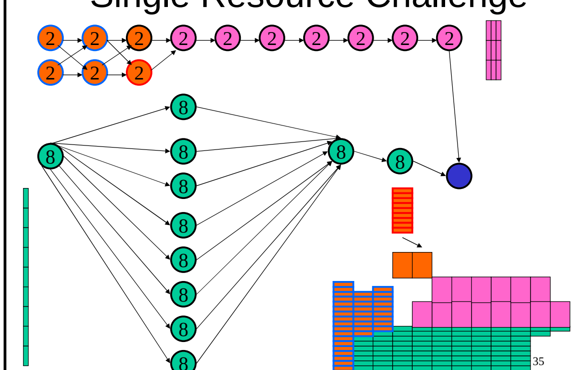
Single Resource Challenge



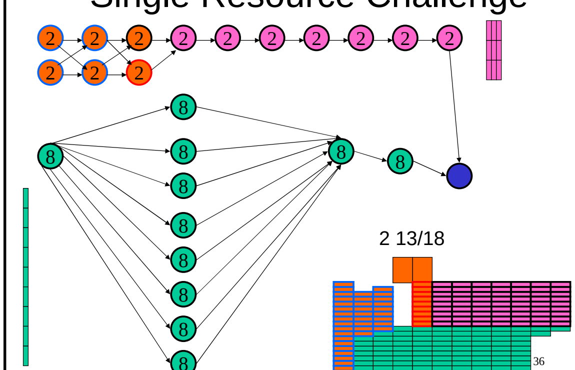
Single Resource Challenge



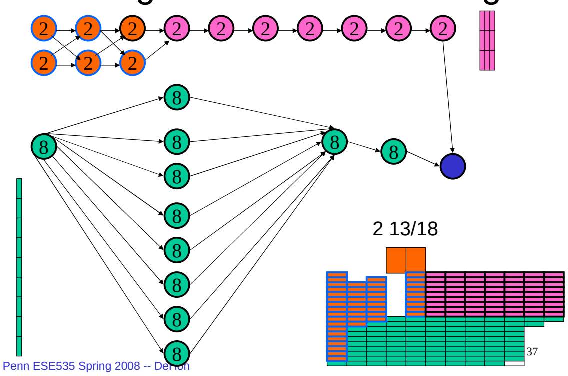
Single Resource Challenge



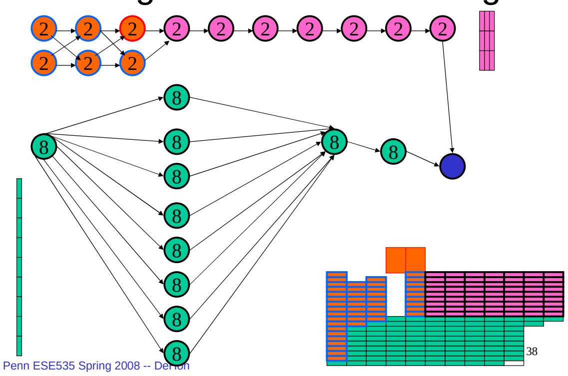
Single Resource Challenge



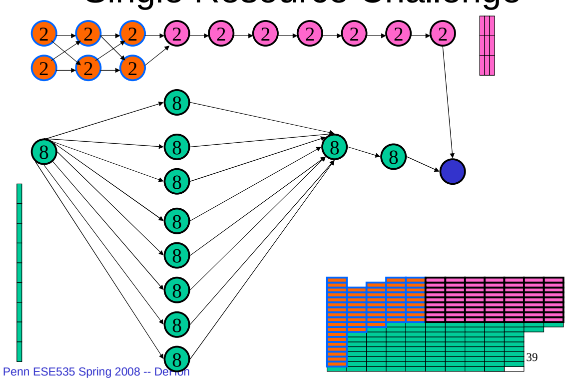
Single Resource Challenge



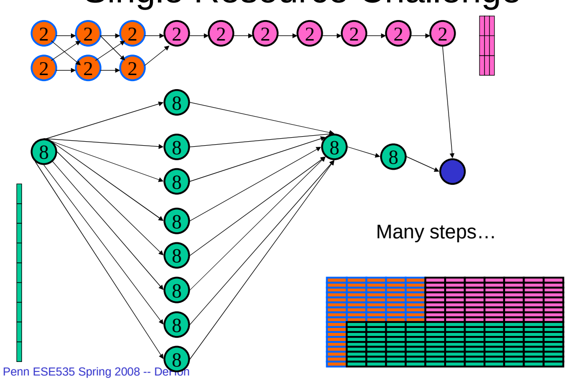
Single Resource Challenge



Single Resource Challenge



Single Resource Challenge



Force-Directed Algorithm

1. ASAP/ALAP schedule to determine range of times for each node
2. Compute estimated resource usage
3. Pick most constrained node (in largest time slot…)
   – 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(N)
  – Potentially perturbing slack on net prefix/postfix for this node → N
- Each node potentially in T slots: ×T
- N nodes to place: ×N
- $O(N^2T)$
  – Loose bound--don't get both T slots and N perturbations

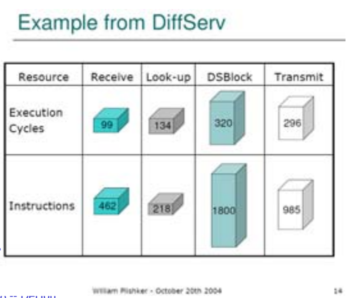
SAT/ILP
(Integer-Linear Programming)

Two Constraint Challenge

- Processing elements have limited memory
  – Instruction memory (data memory)
- Tasks have different requirements for compute and instruction memory
  – *i.e.* Run length not correlated to code length

Task

- **Task:** schedule tasks onto PEs obeying both memory and compute capacity limits



Example and ILP solution From Plishker et al. NSCD2004

Task

- **Task:** schedule tasks onto PEs obeying both memory and compute capacities
- ➔ two capacity partitioning problem
  – …actually, didn't say anything about communication…
- ➔ two capacity bin packing problem
- Task: i <C_i,I_i>

SAT Packing

Variables:

- $A_{i,j}$ – task i assigned to resource j

Constraints

- Coverage constraints
- Uniqueness constraints
- Cardinality constraints
  - PE compute
  - PE memory

$$U_i = \sum_j A_{i,j} = 1$$

$$\sum_i (A_{i,j} \times C_i) \le PE.cap(j)$$

Allow Code Sharing

- Two tasks of same type can share code
- Instead of memory capacity
  – Vector of memory usage
- Compute PE Imem vector
  – As OR of task vectors assigned to it
- Compute mem space as sum of non-zero vector entries

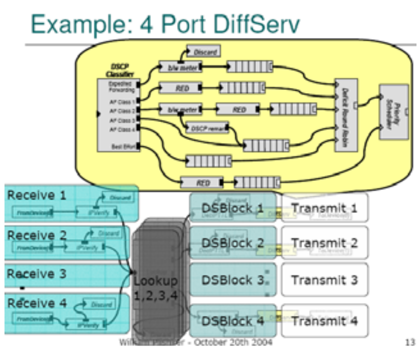
Allow Code Sharing

- Two tasks of same type can share code
- Task has vector of memory uage
  - Task i needs set of instructions k: $T_{i,k}$
- Compute PE Imem vector
  - OR (all i): PE.Imem$_{j,k}$+=$A_{i,j}$ * $T_{i,k}$
- PE Mem space
  - PE.Total_Imem$_j$= Σ(PE.Imem$_{j,k}$*Instrs(k))

Symmetries

- Many symmetries
- Speedup with symmetry breaking
  - Tasks in same class are equivalent
  - PEs indistinguishable
  - Total ordering on tasks and PEs
  - Add constraints to force tasks to be assigned to PEs by ordering
  - Plishker claims "significant runtime speedup"
  - Using GALENA [DAC 2003] psuedo-Boolean SAT solver

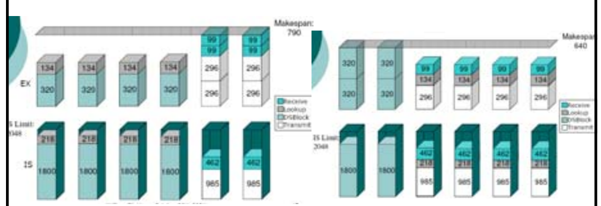
Plishker Task Example



Results

Greedy (first-fit) binpack

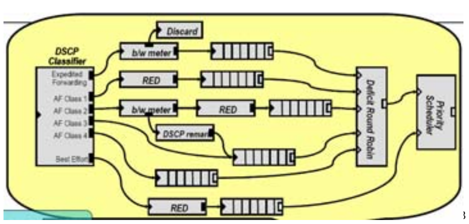
SAT/ILP Solve



Solutions in < 1 second

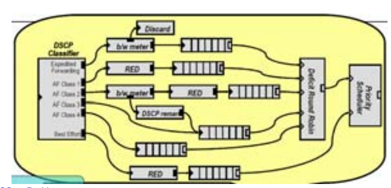
Why can they do this?

- Ignore precedence?
- Ignore Interconnect?



Why can they do this?

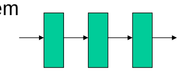
- Ignore precedence?
  - feed forward, buffered
- Ignore Interconnect?
  - Through shared memory, not dominant?



Interconnect Buffers

- Allow "Software Pipelining"

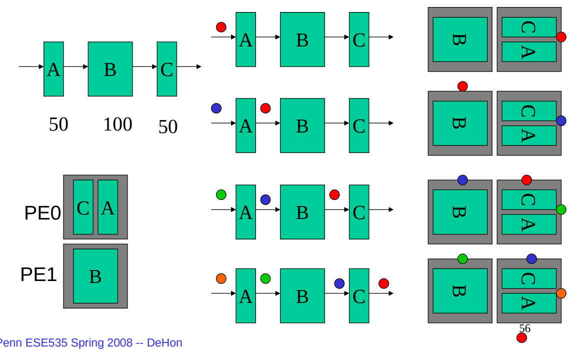
Each data item



Spatial we would pipeline, running all three at once

Think of each schedule instance as one timestep in spatial pipeline.

Interconnect Buffer



Add Precedence to SAT/ILP?

- Assign start time to each task
- **Precedence:** constrain start of each task to be greater than start+run of each predecessor
- **Time Exclusivity:** constrain non-overlap of start→start+run-1 on nodes on same PE
  - Maybe formulate as order on PE
  - And make PE order predecessor like a task predecessor?

Untested conjecture

Memory Schedule Variants

- **Persistent:** holds memory whole time
  - *E.g.* task state, instructions
- **Task temporary:** only uses memory space while task running
- **Intra-Task:** use memory between point of production and consumption
  - *E.g.* Def-Use chains

Memory Schedule Variants

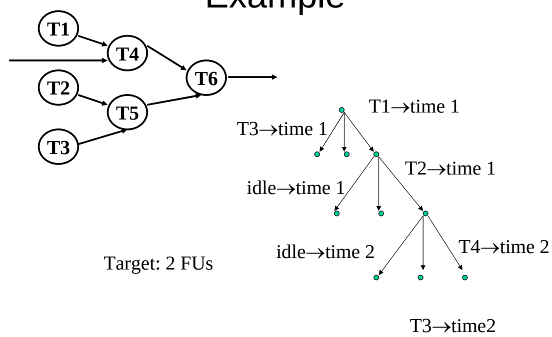
- **Persistent:**
  - Binpacking in memory
- **Task temporary:**
  - Co-schedule memory slot with execution
- **Intra-Task:**
  - Lifetime in memory depends on scheduling **def** and last **use**
  - Phase Ordered: Register coloring

Branch-and-Bound

Brute-Force

- Try all schedules
- Branching/Backtracking Search
- Start w/ nothing scheduled (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 $(N^{(resources*time\text{-}slots)})$
- Many schedules completely uninteresting

Reducing Work

1. Canonicalize "**equivalent**" schedule configurations
2. Identify "**dominating**" schedule configurations
3. **Prune** partial configurations which will lead to worse (or unacceptable results)

"Equivalent" Schedules

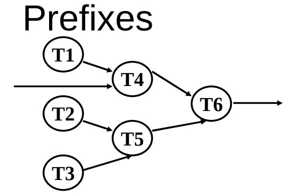
- If multiple resources of same type
  - assignment of task to particular resource at a particular timeslot is not distinguishing

T1	T3
T2	

T2	T3
T1	

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

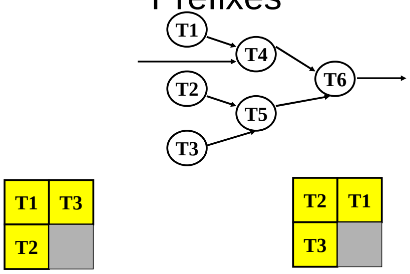
"Equivalent" Schedule Prefixes



T1	T3
T2	T4

T1	T2
T3	T4

“Non-Equivalent” Schedule Prefixes

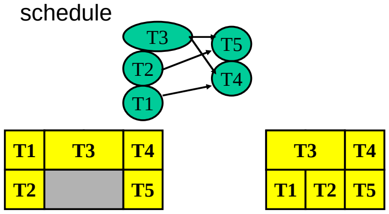


Pruning Prefixes?

- I’m 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
  - …but not all feasible, so shape of graph may simplify

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:
  - LB=current schedule time + lower_bound_estimate
  - if LB greater than existing solution, prune

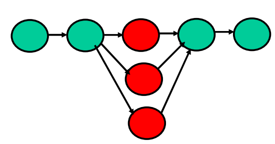
Pruning Techniques

Establish Lower Bound on schedule time

- Critical Path (ASAP schedule)
- Resource Bound
- Critical Chain

“Critical Chain” Lower Bound

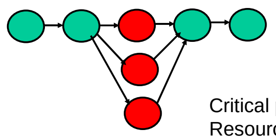
- Bottleneck resource present coupled resource and latency bound



Single red resource

“Critical Chain” Lower Bound

- Bottleneck resource present coupled resource and latency bound



Critical path 5
Resource Bound (1,1) 4
Critical Chain (1,1) 7

Single red resource

73

Alpha-Beta Search

- Generalization
  - keep both upper and lower bound estimates on partial schedule
    - Lower bounds from CP, RB, CC
    - Upper bounds with List Scheduling
  - 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)

74

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”

75

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

76

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

77

Summary

- Resource estimates and Refinement
- SAT/ILP Schedule
- Software Pipelining
- Branch-and-bound search
  - “equivalent” states
  - dominators
  - estimates/pruning

78

Admin

- Reading
- Assignment 6

Big Ideas:

- Estimate Resource Usage
- Use dominators to reduce work
- Techniques:
  - Force-Directed
  - SAT/ILP
  - Coloring
  - Search
    - Branch-and-Bound
    - Alpha-Beta