ESE535: Electronic Design Automation

Day 7: February 9, 2009
Scheduling
Variants and Approaches

Previously

- Introduced Scheduling
- Greedy Schedule Approximation
- Optimal ILP Formulation
- SAT solve

Today

- SAT
  - Learning
- Scheduling
  - Force-Directed
  - SAT/ILP

Learning SAT

SAT Reminder

- Avoid full exponential with pruning search
  - Constrain propagation
  - Contradictions
- Variable ordering important
  - Use statistics on variable/clause activity

Learning

- When encounter a conflict
  - Determine variable assignment contributing to conflict
  - Add new clause to database
- New clause allows pruning

Davis-Putnam w/ Learning

```
while (true) {
  if (!decide()) // no unassigned vars
     return(satisfiable);
  while ( !bcp()) { // constraint propagation
     analyzeConflicts(); // learning
     if (!resolveConflict()) // backtrack
        return(not satisfiable);
  }
}
```


Implication Graph

- As perform bcp propagation
  - When set variable, insert back link to previous variable set forcing this variable set
  - Graph captures what this implication depends upon
- When encounter a conflict
  - Identify what variable values caused

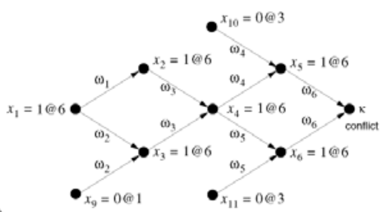
Example

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$

Current Decision Assignment: $\{x_1 = 1@6\}$

$\omega_1 = (\neg x_1 + x_2)$
$\omega_2 = (\neg x_1 + x_3 + x_9)$
$\omega_3 = (\neg x_2 + \neg x_3 + x_4)$
$\omega_4 = (\neg x_4 + x_5 + x_{10})$
$\omega_5 = (\neg x_4 + x_6 + x_{11})$
$\omega_6 = (\neg x_5 + \neg x_6)$
$\omega_7 = (x_1 + x_7 + \neg x_{12})$
$\omega_8 = (x_1 + x_8)$
$\omega_9 = (\neg x_7 + \neg x_8 + \neg x_{13})$
...

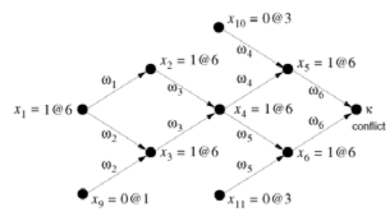
Clause Database



Implication Graph for Current Decision Assignment

Marques-Silva/Sakallah TRCOMP v48n5p506 1999

Conflict Resolution



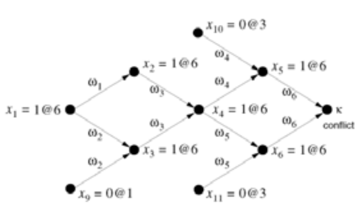
- x1 & /x9 & /x10 & /x11 lead to conflict
- /(x1 & /x9 & /x10 & /x11)
- /x1+x9+x10+x11 ← new clause for DB

New Clause

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$

Current Decision Assignment: $\{x_1 = 1@6\}$

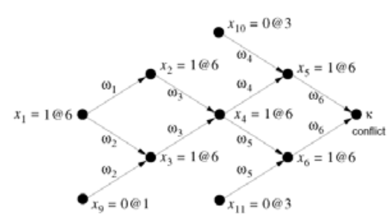
•New clause does not include x12, x13
•May encounter this case again



Implication Graph for Current Decision Assignment

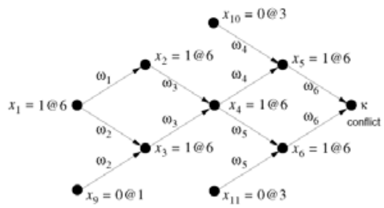
/x1+x9+x10+x11 ← new clause for DB

More Implications



- x4 & /x10 & /x11 lead to conflict
- /x4+x10+x11 ← new clause for DB
- Also (/x1+x9+x4) since x1*/x9 → x4

Unique Implication Point

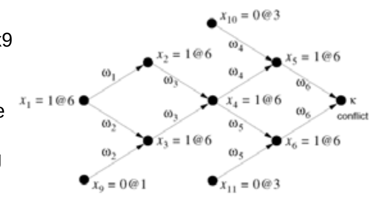


- UIP = vetext that dominates verticies leading to conflict
  - x1 is UIP (decision variable causing is always a UIP)
  - x4 is UIP

New Clauses

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$
Current Decision Assignment: $\{x_1 = 1@6\}$

- /x4+x10+x11
  - Doesn't depend on x9
- (/x1+x9+x4)
  - x4 not in decision tree
- Will be useful for later pruning



Implication Graph for Current Decision Assignment

Clause Tradeoff

- Adding clauses facilitates implications
  - Increases pruning
  - Must make less decisions
- Adding clauses increases size of clause database
  - Increases memory
  - Could add exponential clauses
  - Forces more work to push implications

Learned Clauses

- Runtime = Decisions * ImplicationTime
  - Decisions decreasing
  - Implication Time increasing
- Starting from 0 learned clauses,
  - Net decrease in runtime
- Eventually, Implication Time too large and slows down
- Optimum with limited number of learned clauses

Limiting Learned Clauses

- Filter out dominated clauses
- Keep smaller clauses (fewer literals)
  - Have most relevance
- zChaff study suggest inserting only UIP closest to conflict [Zhang et al., ICCAD2001]
- Treat like cache and evict learned clauses
  - Use activity statistics as with variables so keep most useful clauses [minisat 1.2]

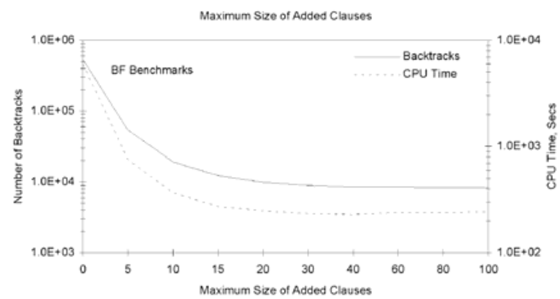
(Recall) Restarts

- Periodically restart
  - Clearing the state of all variables
    - i.e. clear decision stack
  - Leave clauses in clause database
  - State of clause database drives variable ordering
    - Benefit: new variable ordering based on lessons of previous search

Impact of Learning

- zChaff [ICCAD2001] showed 2x improvement based on tuning the learning scheme
- Learning can be orders of magnitude benefit

Impact of Learning



Marques-Silva/Sakallah TRCOMP v48n5p506 1999

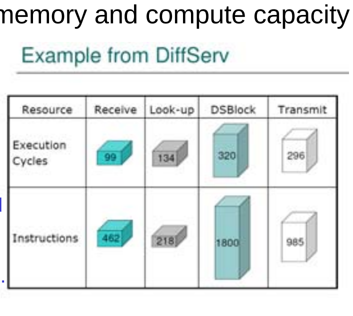
SAT/ILP Scheduling Variant

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

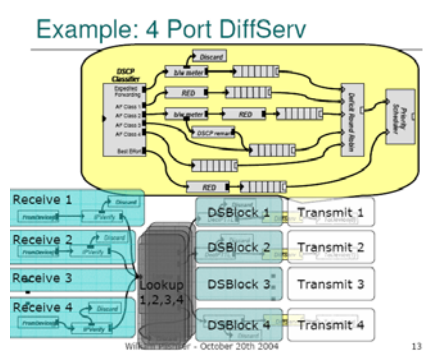
Task

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



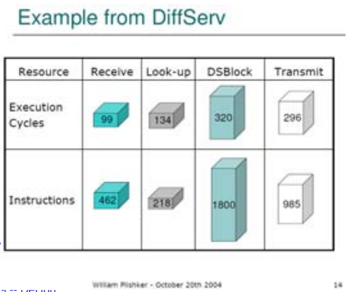
Example and ILP solution From Plishker et al. NSCD2004

Plishker Task Example



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

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

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

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 entry weights (dot product)

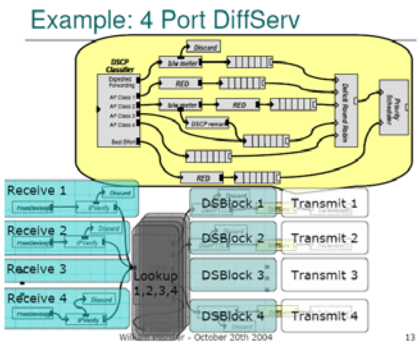
Allow Code Sharing

- 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

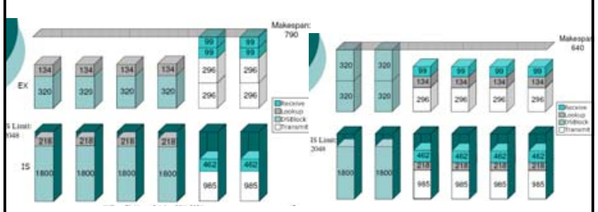
Plishker Task Example



Results

Greedy (first-fit) binpack

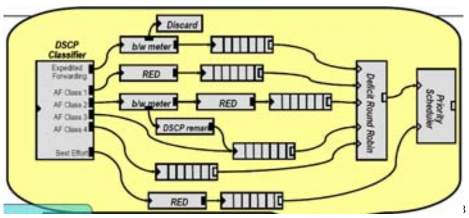
SAT/ILP Solve



Solutions in < 1 second

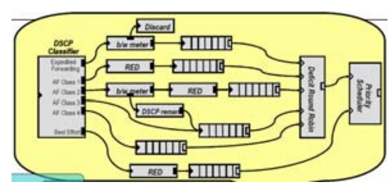
Why can they do this?

- Ignore precedence?
- Ignore Interconnect?



Why can they do this?

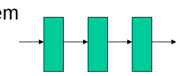
- Ignore precedence?
  - feed forward, buffered
- Ignore Interconnect?
  - Through shared memory, not dominant?



Interconnect Buffers

- Allow "Software Pipelining"

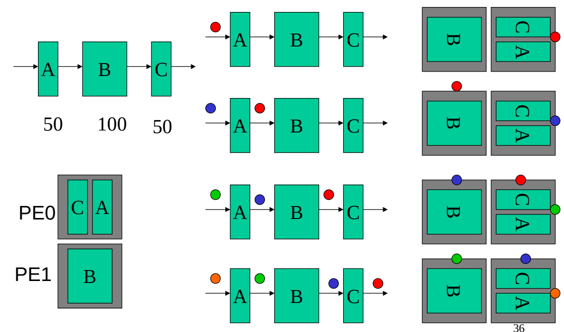
Each data item



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?

- Saw that formulation on Day 5

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

Force Directed

Previously

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

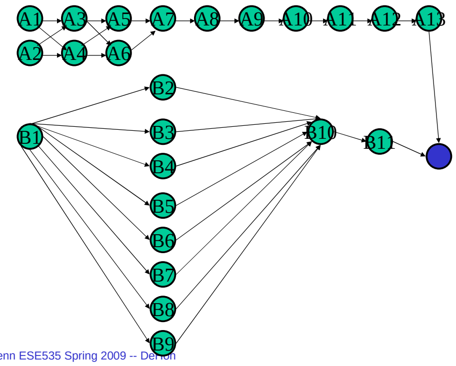
Force-Directed

- **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

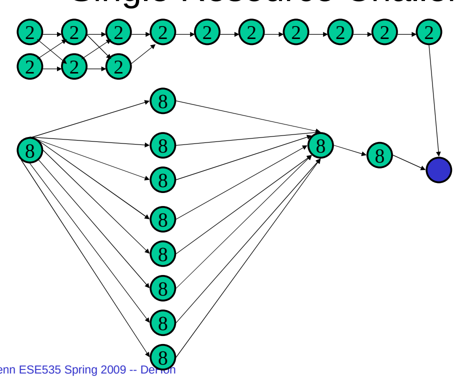
Force-Directed

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

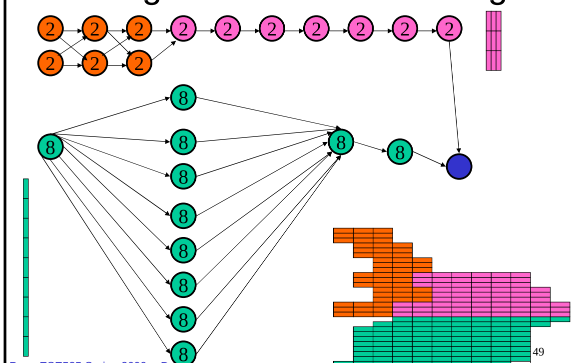
Force-Directed Estimate

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

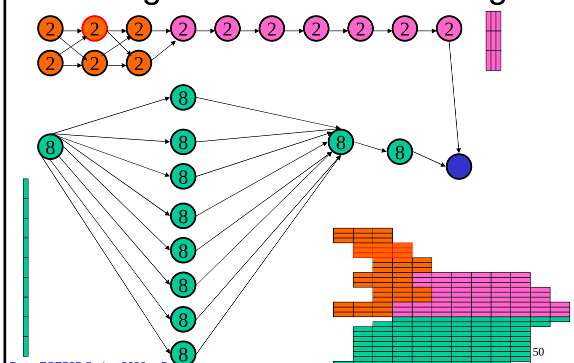
Single Resource Challenge



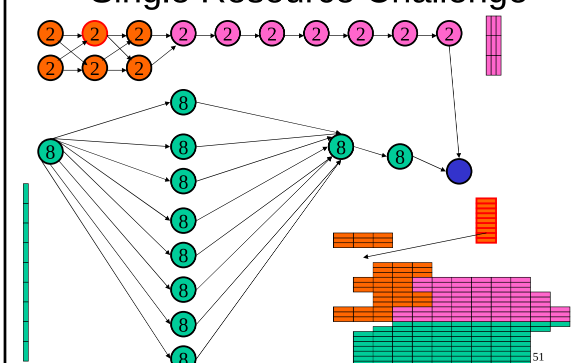
Single Resource Challenge



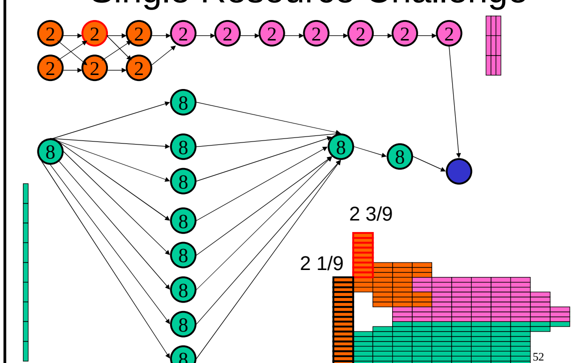
Single Resource Challenge



Single Resource Challenge



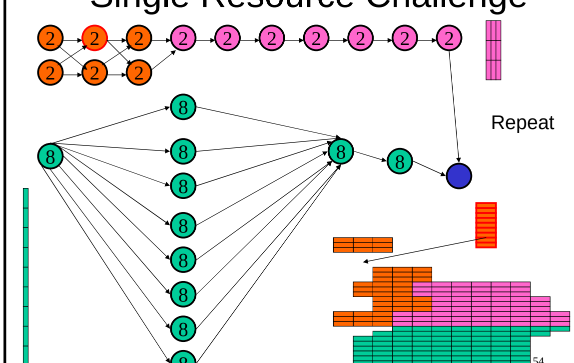
Single Resource Challenge



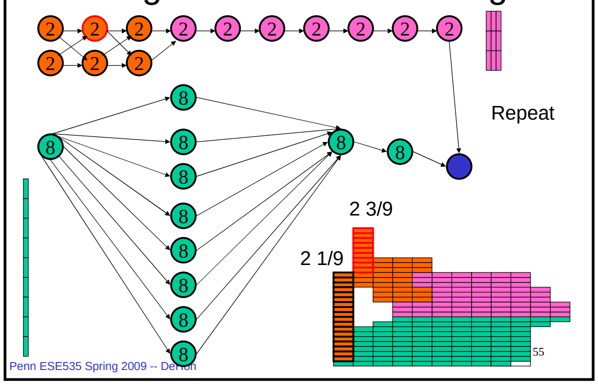
Force-Directed

- 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)

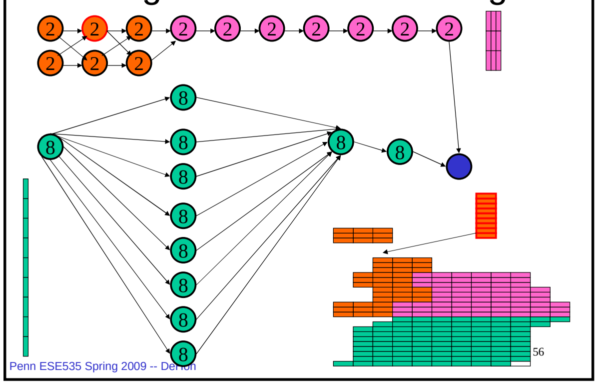
Single Resource Challenge



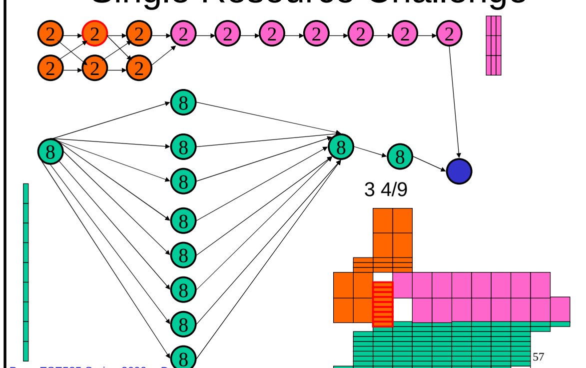
Single Resource Challenge



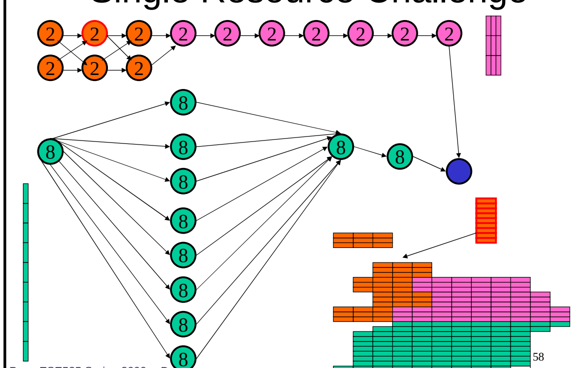
Single Resource Challenge



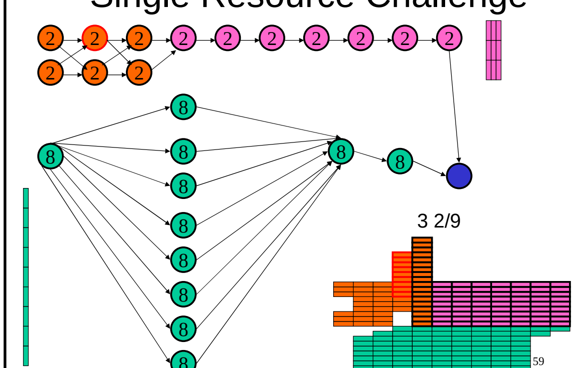
Single Resource Challenge



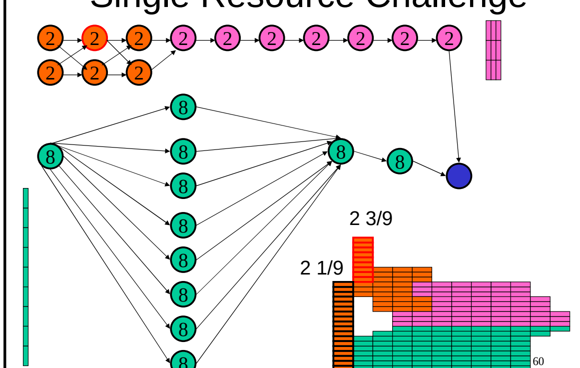
Single Resource Challenge



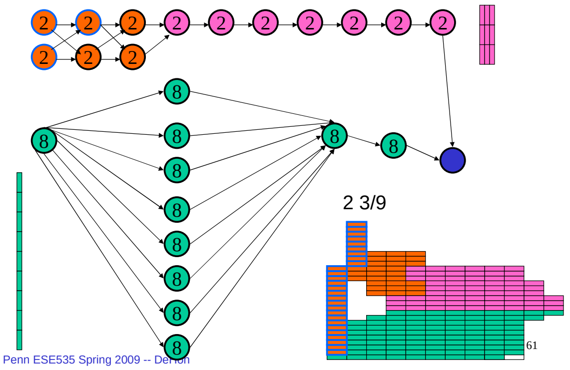
Single Resource Challenge



Single Resource Challenge



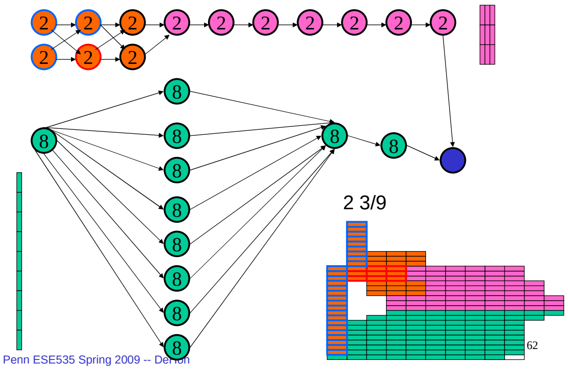
Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

61

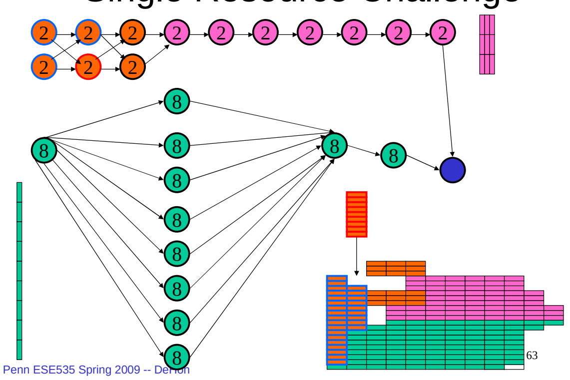
Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

62

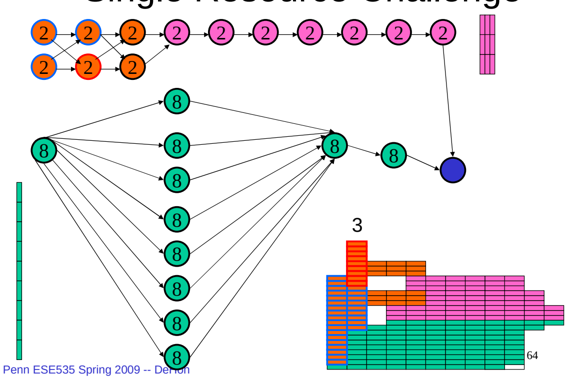
Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

63

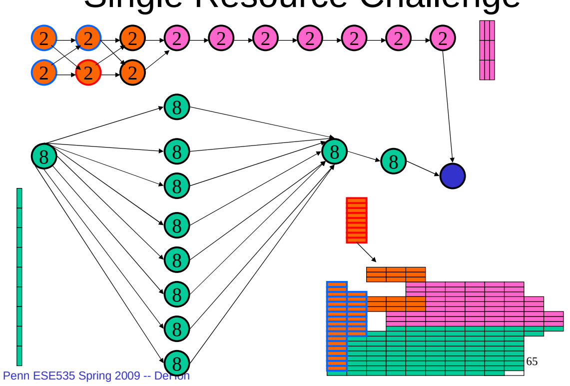
Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

64

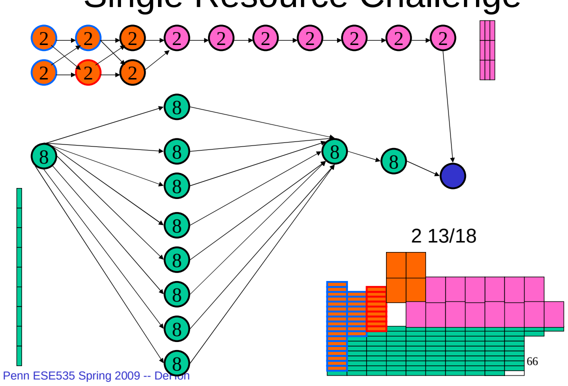
Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

65

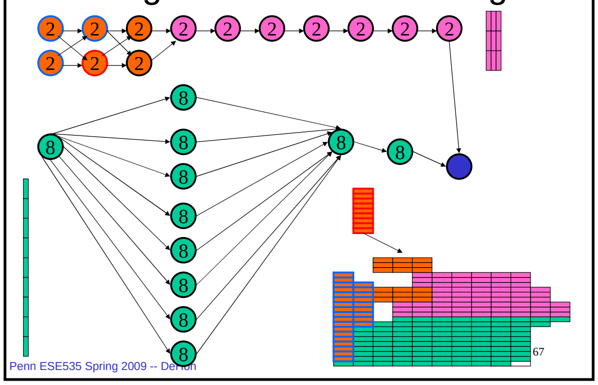
Single Resource Challenge



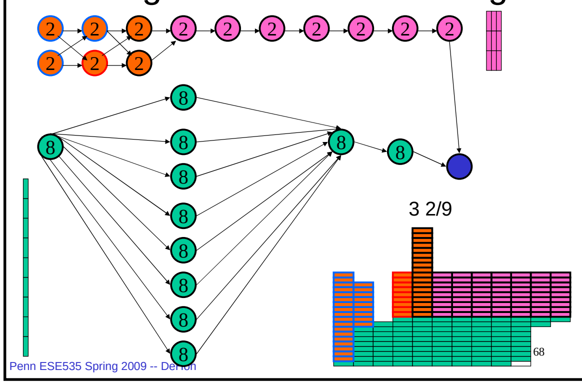
Penn ESE535 Spring 2009 -- DeHon

66

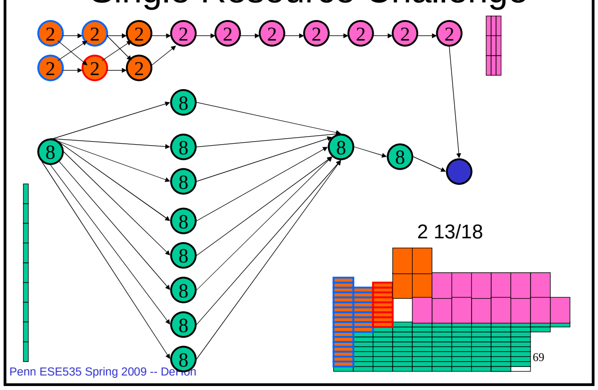
Single Resource Challenge



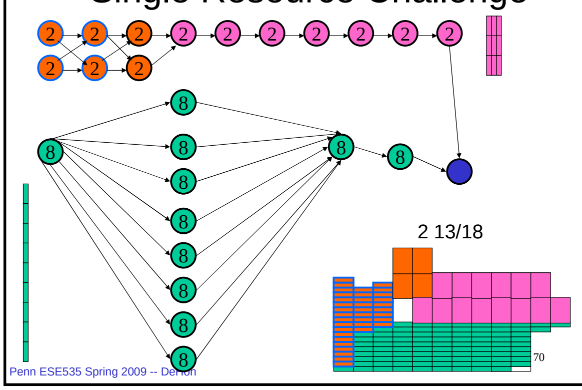
Single Resource Challenge



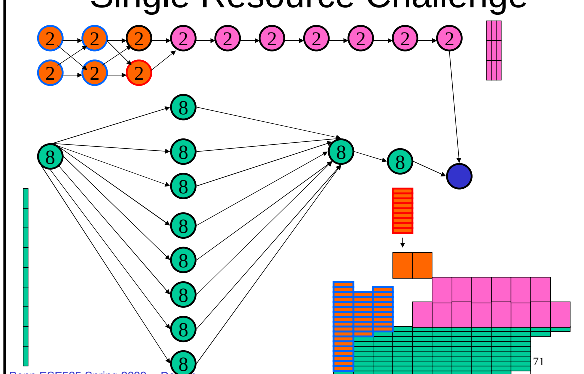
Single Resource Challenge



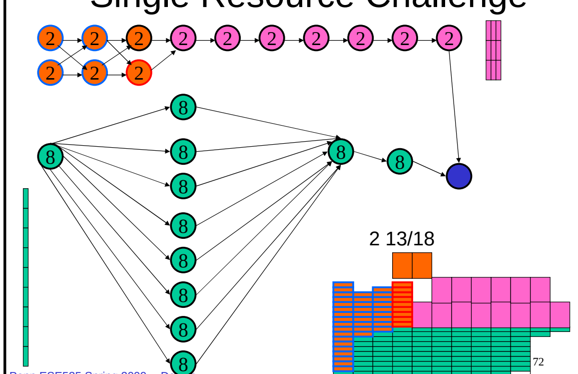
Single Resource Challenge



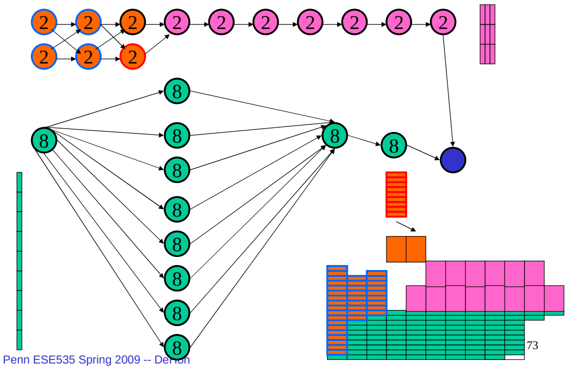
Single Resource Challenge



Single Resource Challenge

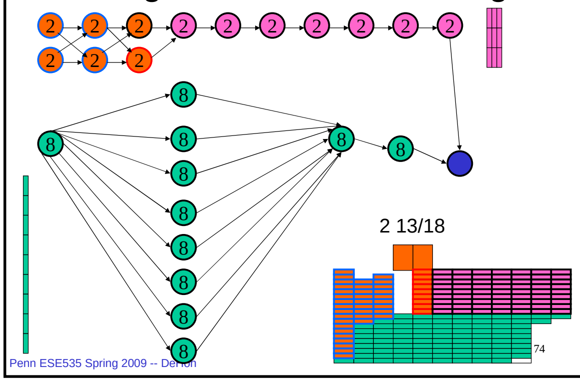


Single Resource Challenge



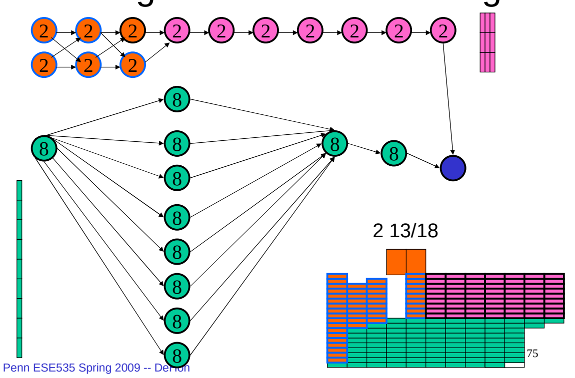
Penn ESE535 Spring 2009 -- DeHon

Single Resource Challenge



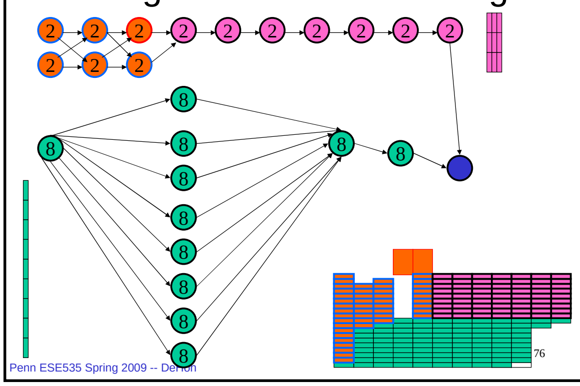
Penn ESE535 Spring 2009 -- DeHon

Single Resource Challenge



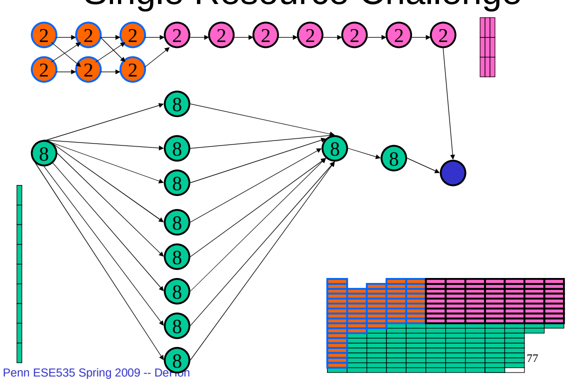
Penn ESE535 Spring 2009 -- DeHon

Single Resource Challenge



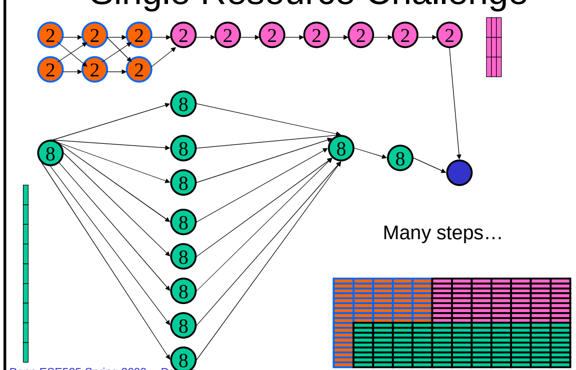
Penn ESE535 Spring 2009 -- DeHon

Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

Single Resource Challenge



Penn ESE535 Spring 2009 -- DeHon

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

Summary

- Learning
  – Understand structure
  – Enhance pruning
- SAT/ILP Schedule
- Resource estimates and refinement
- Software Pipelining

Admin

- Reading
  – Wednesday online

Big Ideas:

- Estimate Resource Usage
- Exploit Structure
  – Learning (discover structure)
- Techniques:
  – Force-Directed
  – SAT/ILP
  – Coloring