Today

- Scheduling
  - Basic problem
  - Variants
  - List scheduling approximation

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)
- Pattern: scheduled operator sharing

Example

Sharing

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

Resource-Time Example

<table>
<thead>
<tr>
<th>Time Constraint:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;5 \rightarrow 5$</td>
</tr>
<tr>
<td>$5 \rightarrow 4$</td>
</tr>
<tr>
<td>$6.7 \rightarrow 2$</td>
</tr>
<tr>
<td>$&gt;7 \rightarrow 1$</td>
</tr>
</tbody>
</table>

Scheduling Use

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

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
- Used for timing analysis (Day 6)

Also Useful to Define ALAP

- As Late As Possible
- Work backward from outputs of DAG
- Also achieve minimum time with unbounded resources
ALAP Example

ALAP and ASAP
- Difference in labeling between ASAP and ALAP is slack of node
  - Freedom to select timeslot
- If ASAP = ALAP, no freedom to schedule

ASAP, ALAP, Difference

Why hard?
- Start with Critical Path?
- Schedule on: 1 Red Resource
  1 Green Resource

General
- When selecting, don’t know
  - need to tackle critical path
  - need to run task to enable work (parallelism)
- Can generalize example to single resource case

Single Resource Hard (1)
General:
Why Hard

- When selecting, don’t know
  – need to tackle critical path
  – need to run task to enable work (parallelism)

Two Bounds
Bounds

• Useful to have bounds on solution
• Two:
  – CP: Critical Path
  – RB: Resource Bound

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)
  – Ignores schedule constraints

Example

Greedy Algorithm → Approximation
List Scheduling (basic algorithm flow)

- 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
- **Key Question:** How prioritize ready list?
  - What is dominant constraint?
    - least slack (worst critical path)
    - enables work
    - utilize most precious resource
- So far:
  - seen 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?
- Technically:
  - If can show \(\text{Heuristic(Prob)}/\text{Optimal(Prob)} \leq \alpha \quad \forall \text{prob}\)
  - Then the Heuristic is an \(\alpha\)-approximation

Scheduled Example Without Precedence

- \(\exists\) 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

Observe
Results

• 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
  – i.e. schedule Longest Processing Time first

Recover Precedence

• With precedence we 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

Precedence Constrained

• Optimal Length > All busy times
  – Optimal Length ≥ Resource Bound
  – Resource Bound ≥ All busy

• Optimal Length > This Path
  – Optimal Length ≥ Critical Path
  – Critical Path ≥ This Path

• List Schedule = This path + All busy times
• List Schedule ≤ 2 * (Optimal Length)

Conclude

• Scheduling of identical parallel machines with precedence constraints has a 2-approximation.

Tighten

• LS schedule ≤ Critical Path + Resource Bound
• LS schedule ≤ Min(CP,RB) + Max(CP,RB)
• Optimal schedule ≥ Max(CP,RB)
• LS/Opt ≤ 1 + Min(CP,RB)/Max(CP,RB)

• The more one constraint dominates
  ➔ the closer the approximate solution to optimal
  % (EEs think about 3dB point in frequency response)
Tightening

- Example of
  - More information about problem
  - More internal variables
  - ...allow us to state a tighter result
- 2-approx for any graph
  - Since CP may = RB
- Tighter approx as CP and RB diverge

Multiple Resource

- Previous result for homogeneous functional units
- For heterogeneous resources:
  - also a 2-approximation
    - Lenstra+Shmoys+Tardos, Math. Programming v46p259
    - (not online, no precedence constraints)

Bounds

- Precedence case, Identical machines
  - no polynomial approximation algorithm can achieve better than 4/3 bound
    - (unless P=NP)
- Heterogeneous machines (no precedence)
  - no polynomial approximation algorithm can achieve better than 3/2 bound

Summary

- Resource sharing saves area
  - allows us to fit in fixed area
- Requires that we schedule tasks onto resources
- General kind of problem arises
- 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

Admin

- Schedule reorg.
  - Deal with recent slip
  - No class on Monday 4/14

Big Ideas:

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