ESE535: Electronic Design Automation

Day 8: February 11, 2015
Scheduling Introduction

Today

• Scheduling
  – Basic problem
  – Variants
  – List scheduling approximation

General Problem
• Resources are not free
  – Wires, io ports
  – Functional units
    • LUTs, ALUs, Multipliers, ....
  – Memory access ports
  – State elements
    • memory locations
    • Registers
      – Flip-flop
      – loadable master-slave latch
    – Multiplexers (mux)

Trick/Technique
• Resources can be shared (reused) in time
• Sharing resources can reduce
  – instantaneous resource requirements
  – total costs (area)
• Pattern: scheduled operator sharing

Example
Assume unit delay operators.
How many operators do I need to evaluate this computation in ~5 time units?

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

Scheduling Use

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

Resource-Time Example

- Time Constraint:
  - <5 → --
  - 5 → 4
  - 6, 7 → 2
  - >7 → 1

Preclass 2

- Schedule onto two adders
  - Does the number of cycles depend on i[7], i[6], ... i[0]?
  - How many cycles?
Preclass 3

- Schedule onto:
  - 2 adders (+)
  - 2 increminter (++)
  - 2 comparator (>)
- Does the number of cycles depend on i[7], i[6], ... i[0]?
- How many cycles?

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 Resource 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

As Soon As Possible (ASAP)

- 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
    - Time-slot = max(time-slot-of-inputs)+1
  - mark inputs of all successors, adding to visit queue when all inputs marked

ASAP Example
Also Useful to Define ALAP

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

ALAP and ASAP

• Difference in labeling between ASAP and ALAP is slack of node
  – Freedom to select timeslot
  – Class theme: exploit freedom to reduce costs
• If ASAP=ALAP, no freedom to schedule

ASAP, ALAP, Difference

Two Bounds
Bounds

- Useful to have bounds on solution
- Two:
  - CP: Critical Path
    - Sometimes call it “Latency Bound”
  - RB: Resource Bound
    - Sometimes call it “Throughput Bound” or “Compute 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

- Critical Path
- Resource Bound (2 resources)
  - $7/2 = 4$
- Resource Bound (4 resources)
  - $7/4 = 2$

Example

- Critical Path
- Resource Bound (2 resources)
- Resource Bound (4 resources)

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)**

```
A1  A2  A3  A4  A5  A6  A7  A8  A9  A10  A11  A12  A13
B1  B2  B3  B4  B5  B6  B7  B8  B9  B10  B11
```

**Crit. Path:**

- A1
- A2
- A3
- A4
- A5
- A6
- A7
- B1
- A8
- B2
- A9
- B3
- A10
- B4
- A11
- B5
- A12
- B6
- A13
- B7
- B8
- B9
- B10
- B11

---

**Single Resource Hard (2)**

```
A1  A2  A3  A4  A5  A6  A7  A8  A9  A10  A11  A12  A13
B1  B2  B3  B4  B5  B6  B7  B8  B9  B10  B11
```

**PFirst**

- A1
- B1
- A2
- B2
- A3
- B3
- A4
- B4
- A5
- B5
- A6
- B6
- A7
- B7
- A8
- B8
- A9
- B9
- A10
- B10
- B11

---

**Single Resource Hard (3)**

```
A1  A2  A3  A4  A5  A6  A7  A8  A9  A10  A11  A12  A13
B1  B2  B3  B4  B5  B6  B7  B8  B9  B10  B11
```

**Balance1**

- A1
- B1
- A2
- B2
- A3
- B3
- A4
- B4
- A5
- B5
- A6
- B6
- A7
- B7
- A8
- B8
- A9
- B9
- A10
- B10
- B11
- A12
- A13

---

**List Scheduling**

**Greedy Algorithm → Approximation**

---

**List Scheduling (basic algorithm flow)**

- Keep a ready list of “available” nodes
  - (one whose predecessors have already been scheduled)
  - Like ASAP queue
    - But won’t necessary process in FIFO order

- While there are unscheduled tasks
  - Pick an unscheduled task and schedule on first available resource after its predecessors
  - 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) \(\Rightarrow\) LPT
  - LPT = Longest Processing Time first
    - enables work
    - utilize most precious (limited) resource
- So far:
  - seen that no single priority scheme would be optimal

**LPT Schedule**

- 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
- Good for upper bound – results is achievable
- Not always optimal
- How good?

**Approximation**

- Can we say how close an algorithm comes to achieving the optimal result?
- Technically:
  - **If** can show
    - \(\text{Heuristic(Prob)}/\text{Optimal(Prov)} \leq \alpha\) \(\forall\) Prob
  - **Then** the Heuristic is an \(\alpha\)-approximation

**Scheduled Example**

Without Precedence
Observe

- \( \exists \) 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 \( \leq 2L \)
- What can say about optimality?
  - Algorithm is within factor of 2 of optimum

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 Constrained

- Optimal Length > All busy times
  - Optimal Length \( \geq \) Resource Bound
  - Resource Bound \( \geq \) All busy
- Optimal Length>This Path
  - Optimal Length \( \geq \) Critical Path
  - Critical Path \( \geq \) This Path
- List Schedule = This path + All busy times
- List Schedule \( \leq 2 \times (\text{Optimal Length}) \)

Conclude

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

• How could we do better?

• What is particularly pessimistic about the previous cases?
  – List Schedule = This path + All busy times
  – List Schedule ≤ 2 * (Optimal Length)

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
Relate HMC

• How does this relate to our mapping for Heterogeneous multicontext computing array?

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

Admin

• Reading on web for Monday
  – Same reading for today and Monday
• Assignment 4 Due Thursday