ESE535: Electronic Design Automation

Day 4: January 28, 2015
Partitioning
(Intro, KLFM)

Today

• Partitioning
  – why important
    • Can be used as tool at many levels
  – practical attack
  – variations and issues

Motivation (1)

• Cut size (bandwidth) can determine
  – Area, energy
• Minimizing cuts
  – minimize interconnect requirements
  – increases signal locality
• Chip (board) partitioning
  – minimize IO
• Direct basis for placement
  – Particularly for our heterogeneous multicontext computing array

Motivation (2)

• Divide-and-conquer
  – trivial case: decomposition
  – smaller problems easier to solve
    • net win, if super linear
  – Part(n) + 2\cdot T(n/2) < T(n)
  – problems with sparse connections or interactions
  – Exploit structure
    • limited cutsize is a common structural property
    • random graphs would not have as small cuts

Bisection Width

• Partition design into two equal size halves
  – Minimize wires (nets) with ends in both halves
• Number of wires crossing is **bisection width**
• lower \( \text{bw} \) = more locality
Interconnect Area

- Bisection width is lower-bound on IC width
  - When wire dominated, may be tight bound
- (recursively)

Classic Partitioning Problem

- **Given:** netlist of interconnect cells
- **Partition into two (roughly) equal halves** \((A,B)\)
- minimize the number of nets shared by halves
- “Roughly Equal”
  - balance condition: \((0.5-\delta)N \leq |A| \leq (0.5+\delta)N\)

Balanced Partitioning

- NP-complete for general graphs
  - \([ND17: \text{Minimum Cut into Bounded Sets}, \text{Garey and Johnson}]\)
  - Reduce SIMPLE MAX CUT
  - Reduce MAXIMUM 2-SAT to SMC
  - Unbalanced partitioning poly time
- Many heuristics/attacks

KL FM Partitioning Heuristic

- Greedy, iterative
  - pick cell that decreases cut and move it
  - repeat
- small amount of non-greediness:
  - look past moves that make locally worse
  - randomization

Fiduccia-Mattheyses
(Kernighan-Lin refinement)

- Start with two halves (random split?)
- Repeat until no updates
  - Start with all cells free
  - Repeat until no cells free
    - Move cell with largest gain (balance allows)
    - Update costs of neighbors
    - Lock cell in place (record current cost)
  - Pick least cost point in previous sequence and use as next starting position
- Repeat for different random starting points

Efficiency

Tricks to make efficient:
- Expend little work picking move candidate
  - Constant work \(\equiv O(1)\)
  - Means amount of work not dependent on problem size
- Update costs on move cheaply \([O(1)]\)
- Efficient data structure
  - update costs cheap
  - cheap to find next move
Ordering and Cheap Update

- Keep track of Net gain on node \( \delta \) delta net crossings to move a node
  - cut cost after move = cost - gain
- Calculate node gain as \( \Sigma \) net gains for all nets at that node
  - Each node involved in several nets
- Sort nodes by gain
  - Avoid full resort every move

After move node?

- Update cost
  - Newcost=cost-gain
- Also need to update gains
  - on all nets attached to moved node
  - but moves are nodes, so push to
    - all nodes affected by those nets

FM Recompute Cell Gain

- For each net, keep track of number of cells in each partition \([F(net), T(net)]\)
- Move update: (for each net on moved cell)
  - if \( T(net) = 0 \), increment gain on F side of net
    - (think \( -1 \Rightarrow 0 \))
  - if \( T(net) = 1 \), decrement gain on T side of net
    - (think \( 1 \Rightarrow 0 \))

Composability of Net Gains

Gain = Delta in number of nets crossing between partitions
= Sum of net deltas for nets on the node

FM Cell Gains

Gain = Delta in number of nets crossing between partitions
= Sum of net deltas for nets on the node
**FM Recompute Cell Gain**

- Move update: (for each net on moved cell)
  - if $T\text{(net)}=0$, increment gain on $F$ side of net
  - if $T\text{(net)}=1$, decrement gain on $T$ side of net
  - decrement $F\text{(net)}$, increment $T\text{(net)}$

- For each net, keep track of number of cells in each partition [$F\text{(net)}$, $T\text{(net)}$]
  - Move update: (for each net on moved cell)
    - if $T\text{(net)}=0$, increment gain on $F$ side of net
      - (think $-1 \Rightarrow 0$)
    - if $T\text{(net)}=1$, decrement gain on $T$ side of net
      - (think $1 \Rightarrow 0$)
    - decrement $F\text{(net)}$, increment $T\text{(net)}$
    - if $F\text{(net)}=1$, increment gain on $F$ cell
    - if $F\text{(net)}=0$, decrement gain on all cells ($T$)

**FM Recompute (example)**

[Note markings here are deltas...earlier pix were absolutes]
FM Recompute (example)

FM Recompute (example)

FM Recompute (example)

FM Recompute (example)

FM Data Structures

- Partition Counts A,B
- Two gain arrays
  - One per partition
  - Key: constant time cell update
- Cells
  - successors (consumers)
  - inputs
  - locked status

Binned by cost $\rightarrow$ constant time update

Use FM to Partition Preclass Example

- Allow partition of size 5

[diagram of FM data structure and example]
Use FM to Partition Preclass Example

- Initial Partition
- Initial cut size?
- Identify Gains?

Use FM to Partition Preclass Example

- Initial Partition (cut 6)
- Move lists:
  - Left:
    - 2: A
    - 1: E, G
    - 0: B
  - Right:
    - 3: D
    - 1: H
    - 0: F
    - -1: C

Use FM to Partition Preclass Example

- Initial Partition
- Move D
  - Cut: 6-3 = 3
  - Update Gains

Use FM to Partition Preclass Example

- Move lists:
  - Left:
    - 1: G
    - 0: A
    - -1: E
    - -2: B
  - Right:
    - 2:
      - 1: H
      - 0: F
      - -1: C

Use FM to Partition Preclass Example

- Move G
  - Cost: 3-1=2
  - Update Gains?
Use FM to Partition Preclass Example

• Move G
  • Cost: 3-1=2
  • Update Gains?

Use FM to Partition Preclass Example

• Move lists:
  • Left:
    0: A
    -1: E
    -2: B
  • Right:
    2:
    1: H
    -1: C
    -2: F

Use FM to Partition Preclass Example

• Move H
  • Cost: 2-1=1
  • Update Gains?

Use FM to Partition Preclass Example

• Move lists:
  • Left:
    0:
    -1: A
    -2: B
    -3: E
  • Right:
    2:
    -1: C
    -2: F
**FM Optimization Sequence (ex)**

- Gain sequence:
  - +3
  - +3
  - +2
  - +2
  - +1
  - +1
  - 0
  - 0
  - 0
  - 0
  - -1
  - -1
  - -1
  - -1
  - -2
  - -2
  - -2
  - -2
  - -3
  - -3
  - -3
  - -3
  - +3
  - +3
  - 0

**FM Running Time?**

- **Assume:**
  - constant number of passes to converge
  - constant number of random starts
- **N cell updates each round (swap)**
- **Updates K + fanout work (avg. fanout K)**
  - assume at most K inputs to each node
  - For every net attached (K+1)
    - For every node attached to those nets (O(K))
- **Maintain ordered list O(1) per move**
  - every io move up/down by 1
- **Running time:** O(K^N)
  - Algorithm significant for its speed
    - (more than quality)

**FM Starts?**

- 21K random starts, 3K network -- Alpert/Kahng
- So, FM gives a not bad solution quickly
Weaknesses?

- Local, incremental moves only
  - hard to move clusters
  - no lookahead
  - Stuck in local minima?
- Looks only at local structure

Time Permit

Improving FM

- Clustering
- Initial partitions
- Runs
- Partition size freedom

Following comparisons from Hauck and Boriello '96

Clustering

- Group together several leaf cells into cluster
- Run partition on clusters
- Uncluster (keep partitions)
  - iteratively
- Run partition again
  - using prior result as starting point
  - instead of random start

Clustering Benefits

- Catch local connectivity which FM might miss
  - moving one element at a time, hard to see move whole connected groups across partition
- Faster (smaller N)
  - METIS – fastest research partitioner exploits heavily

How Cluster?

- Random
  - cheap, some benefits for speed
- Greedy “connectivity”
  - examine in random order
  - cluster to most highly connected
  - 30% better cut, 16% faster than random
- Spectral (next week)
  - look for clusters in placement
  - (ratio-cut like)
- Brute-force connectivity (can be O(N^2))
Initial Partitions?

- Random
- Pick Random node for one side
  - start imbalanced
  - run FM from there
- Pick random node and Breadth-first search to fill one half
- Pick random node and Depth-first search to fill half
- Start with Spectral partition

Initial Partitions

- If run several times
  - pure random tends to win out
  - more freedom / variety of starts
  - more variation from run to run
  - others trapped in local minima

Number of Runs

- 2 - 10%
- 10 - 18%
- 20 < 20%
- 50 < 22%
- ...but?

Unbalanced Cuts

- Increasing slack in partitions
  - may allow lower cut size

Unbalanced Partitions

Following comparisons from Hauck and Boriello '96
Partitioning Summary

- Decompose problem
- Find locality
- NP-complete problem
- Linear heuristic (KLFM)
- Many ways to tweak
  - Hauck/Boriello, Karypis

Today’s Big Ideas:

- Divide-and-Conquer
- Exploit Structure
  - Look for sparsity/locality of interaction
- Techniques:
  - greedy
  - incremental improvement
  - randomness avoid bad cases, local minima
  - incremental cost updates (time cost)
  - efficient data structures

Admin

- Reading for Monday online
- Assignment 2 due on tomorrow
- Assignment 3 (4, 5, 6) out today