Today

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

Motivation (1)

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

Motivation (2)

- Cut size (bandwidth) can determine
  - Area, energy
- Minimizing cuts
  - minimize interconnect requirements
  - increases signal locality
- Chip (board) partitioning
  - minimize IO
- Direct basis for placement

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
  - \([\text{ND17: Minimum Cut into Bounded Sets, Garey and Johnson}]\)
  - 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 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
FM Cell Gains
Gain = Delta in number of nets crossing between partitions
= Sum of net deltas for nets on the node

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

Composability of Net Gains
\[-1 + 1 - 0 - 1 = -1\]

FM Recompute Cell Gain
• 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 \(1 \Rightarrow 0\))

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 \(T(\text{net})\)
**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 $T(\text{net})$
  - if $F(\text{net}) = 1$, increment gain on $F$ cell
  - if $F(\text{net}) = 0$, decrement gain on all cells ($T$)

---

**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 $T(\text{net})$
  - if $F(\text{net}) = 1$, increment gain on $F$ cell
  - if $F(\text{net}) = 0$, decrement gain on all cells ($T$)

---

**FM Recompute (example)**

- **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)**

- **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 $T(\text{net})$
  - if $F(\text{net}) = 1$, increment gain on $F$ cell
  - if $F(\text{net}) = 0$, decrement gain on all cells ($T$)
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

FM Optimization Sequence (ex)

FM Running Time?

• Randomly partition into two halves
• Repeat until no updates
  – Start with all cells free
  – Repeat until no cells free
    • Move cell with largest gain
    • 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
FM Running Time

- **Claim**: small number of passes to converge
  - Constant passes?
- Small (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^2 N)
  - Algorithm significant for its speed
  - (more than quality)

Weaknesses?

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

Improving FM

- Clustering
- Initial partitions
- Runs
- Partition size freedom
- Replication

Following comparisons from Hauck and Boriello '96

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

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

So, FM gives a *not bad* solution quickly

21K random starts, 3K network -- Alpert/Kahng

FM Starts?

31

32

33

34

35

36
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
- 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

- 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

Small/large is benchmark size not small/large partition IO.

Replication

• Trade some additional logic area for smaller cut size
  – Net win if wire dominated

Replication data from: Enos, Hauck, Sarrafzadeh '97

• 5% ➞ 38% cut size reduction
• 50% ➞ 50+% cut size reduction

What Bisection doesn’t tell us

• Bisection bandwidth purely geometrical
• No constraint for delay
  – i.e. a partition may leave critical path weaving between halves

Critical Path and Bisection

Minimum cut may cross critical path multiple times. Minimizing long wires in critical path ➞ increase cut size.

So...

• Minimizing bisection
  – good for area
  – oblivious to delay/critical path
Partitioning Summary

- Decompose problem
- Find locality
- NP-complete problem
- Linear heuristic (KLFM)
- Many ways to tweak
  - Hauck/Boriello, Karypis
- Even better with replication
- Only address cut size, not critical path delay

Admin

- Reading for Wed. online
- Assignment 2A due on Monday

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