

# EDA (CS286.5b)

Day 5  
Partitioning:  
Intro + KLFM

# Today

- Partitioning
  - why important
  - practical attack
  - variations and issues

## Motivation (1)

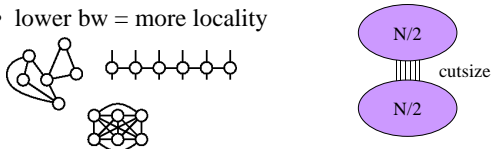
- Divide-and-conquer
  - trivial case: decomposition
  - smaller problems easier to solve
    - net win, if super linear
  - problems with sparse connections or interactions
  - Exploit structure
    - limited cutsize is a common structural property
    - random graphs would **not** have as small cuts

## Motivation (2)

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

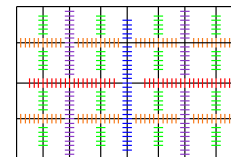
## Bisection Bandwidth

- Partition design into two equal size halves
- Minimize wires (nets) with ends in both halves
- Number of wires crossing is **bisection bandwidth**
- lower bw = more locality



## Interconnect Area

- Bisection is lower-bound on IC width
- (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:  $(0.5-\delta)N \leq |A| \leq (0.5+\delta)N$

## Partitioning

- NP-complete for general graphs
- Many heuristics/attacks

## KL FM Partitioning Heuristic

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

## Fiduccia-Mattheyses (Kernighan-Lin refinement)

- Randomly partition into two halves
- 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

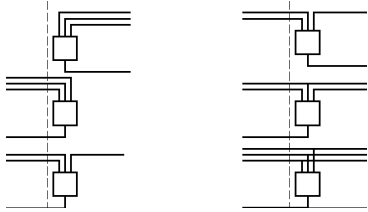
- Pick move candidate with little work
- Update costs on move cheaply
- Efficient data structure
  - update costs cheap
  - cheap to find next move

## 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
- Sort by gain

## FM Cell Gains

Gain = Delta in number of nets crossing between partitions



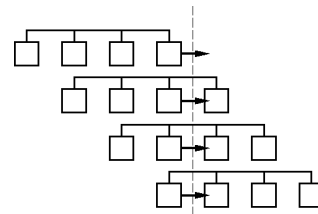
## After move node?

- Update cost each
  - cost-gain
- Also need to update gains
  - on all nets attached to moved node
  - roll up to all nodes affected by those nets

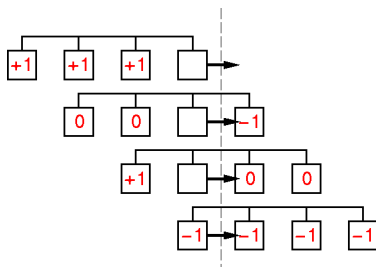
## FM Recompute Cell Gain

- 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 => 0)
  - if T(net)==1, decrement gain on T side of net
    - (think 1=>0)
  - decrement F(net), increment T(net)
  - if F(net)==1, increment gain on F cell
  - if F(net)==0, decrement gain on all cells (T)

## FM Recompute (example)

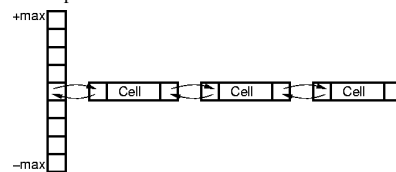


## FM Recompute (example)



## FM Data Structures

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



## FM Optimization Sequence (ex)

+3	+2	-1
+3	+1	-1
+2	0	-1
+2	-1	0
+1	0	+1
0	-1	-1
+1	-1	-1
0	-2	-2
0	-2	-2
-1	-1	-2
+1	0	-1
-1	-1	-2
-2	-2	-1
-2	-3	-3
-3	-3	-3
-3	-3	-3
+12	+3	0

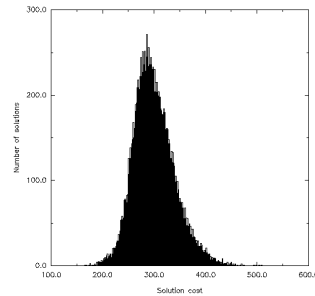
## FM Running Time?

- 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 (constant?) to converge
- Small (constant?) number of random starts
- N cell updates
- Updates K + fanout work (avg. fanout K)
  - assume K-LUTs
- Maintain ordered list O(1) per move
  - every io move up/down by 1
- Running time: O(KN)

## FM Starts?



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

## Weaknesses?

- Local, incremental moves only
  - hard to move clusters
  - no lookahead
  - [show example]
- Looks only at local structure

## Improving FM

- Clustering
- technology mapping
- initial partitions
- runs
- partition size freedom
- replication

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 partitioners exploits heavily
  - FM work better w/ larger nodes (???)

## 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 time)
  - look for clusters in placement
  - (ratio-cut like)
- Brute-force connectivity (can be  $O(N^2)$ )

## LUT Mapped?

- Better to partition before LUT mapping.



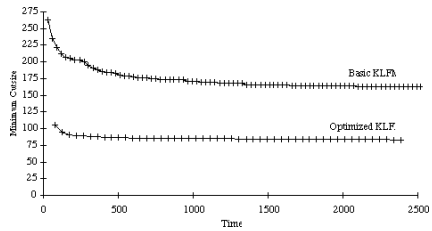
## Initial Partitions?

- 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

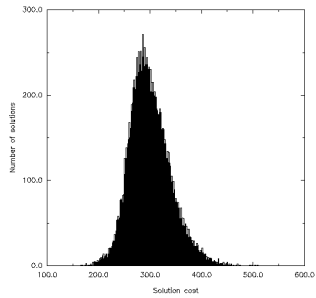
## Number of Runs



## Number of Runs

- 2 - 10%
- 10 - 18%
- 20 < 20% (2% better than 10)
- 50 (4% better than 10)
- ...but?

## FM Starts?

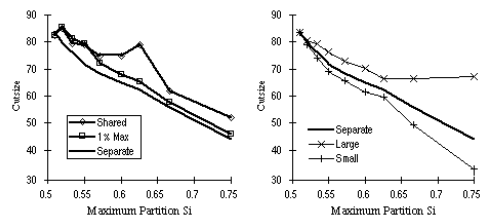


21K random starts, 3K network -- Alpert/Kahng

## Unbalanced Cuts

- Increasing slack in partitions
  - may allow lower cut size
  - [show contrived example]

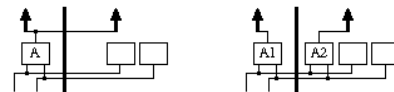
## Unbalanced Partitions



Following comparisons from Hauck and Boriello '96

## Replication

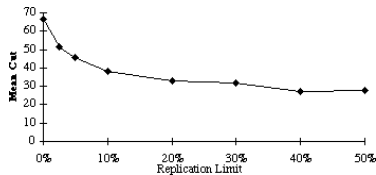
- Trade some additional logic area for smaller cut size



Replication data from: Enos, Hauck, Sarrafzadeh '97

## Replication

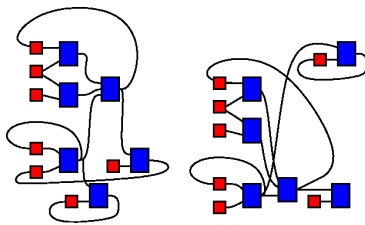
- 5% => 38% cut size reduction
- 50% => 50+% cut size reduction



## 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

## 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