CSE248
Spring 2011
Partitioning
Many-Core Placement (discussion #3)

**Main parts**

- **Testcase generation** – up to 50M cells
  - Tiling existing benchmarks – 25 copies of bblue4 (don’t like this)
  - “Netlist generators” – gnl (tell me what you decided to NOT use – Darnauer-Dai, Hutton et al., etc.)
  - Agree on netlist parameters (e.g., macro cells, % of buffers/inverters, Rent parameters, FOD, …), sanity checks (degree, pathlengths, etc.)
  - Implementation work (SN) – must be tested and available two weeks before final experiments done

- **GP ends with a “global placement, but not a legal placement”**
  - Phase 1 Option1: parallel clustering (LJW – later)
  - Phase 1 Option2: {GWTW, multi-start top-down partitioning with constrained depth} (Capo, MLPart) (SN - now)
  - **Phase 2 (perfectly parallelizable) (based on APlace / SimPL) {PLP, WLR}**

- **DP has two phases** (LJW – now)
  - Legalization: revise your pseudocode and make it clear how it is parallelized, which rows are fixed while others are being adjusted, how to avoid cycling
  - Detailed Placement: keep simple (no independent sets, sorting, … or other non-decomposable steps) – start with Mongrel (multi-row DP) or other “powerful” local operator
  - *** Think through WHY you’re “innovating” (to escape local minima of previous works, to ensure perfect parallelizability, …)

- (Platform testing – gcc4.6 install, …)
Planar Bounded-Skew Tree (discussion #2)

- Started implementation of Charikar (to finish tomorrow night)
  - Sanity-checks: points on a line, points on a circle, points on a lattice, …
  - Visualization (plotting), small cases, etc. are necessary before moving to larger instances
- John Nale code via Prof. Mandoiu doesn’t compile: if this takes > a day to fix, then implement from scratch (should take a day)
  - Sweet spot is 50-100 sinks, so runtime efficiency is not the issue
- “BST-DME is fine”
  - (What does “fine” mean? → communicate validations that have been made)
- *What needs to be changed in Charikar / Zelikovsky to comprehend or acknowledge the “P” and the “Z” in PBST?* → compile thoughts on this by 5/16 (start of 8th week)
- More on goals/targets
  - Not just a “plain” comparison of “I implemented A, B, C – here is what happened”
  - BST-DME and others: can you improve/explore topology?
  - Rooted-Kruskal: again, can you explore other orderings (bounded-discrepancy search)?
  - In what ranges (between B = 0, B = +infty; between N = small and N = large) do the various methods have their best relative performances?
Known Optimal Solutions (discussion #2)

- Topic has converged: sizing for low power under (setup) timing constraints
- Most-related work = DAC10 Eye Charts
- Qualitative goals: “scalable” (= composable?), “realistic”, “known optimal”
- New idea
  - Start with “eye chart” which is very simple: chains, stars, meshes
  - Augment “eye charts” to be more realistic without losing “known optimality”
  - IDEA: insert “fixed-size” cells (buffers) to decouple the sizing problem at various places in the netlist, to enable augmentation of topology while preserving “known optimality”
  - Entire library (following DAC-10 ideas) has no slew-dependence, which enables DP for sizing
- Open issues
  - Are you sure that optimality is preserved? (We agreed: Yes)
  - How to use and connect those gray buffers to achieve “realistic topology”?  
  - Does the introduction of the fixed-size buffers and extra nets really make the problem harder for optimizers?
    - Should there be little slack (forces the tool to the correct solution?) or lots of slack (doesn’t help the tool?) on the new paths involving the gray buffers?
    - Can you study existing sensitivity functions (SensOpt, DUET, etc.) and engineer buffer sizes and connections to “fool” or “misdirect” existing optimizers?
  - (Can the original library, including slew-dependence, be used? (Doubtful: probably prevents application of DP, and hence “known optimal”.)
Adaptive Testing (discussion #1)

- Method of Uezono et al. (VTS 2010) has unbounded error
- NP-completeness hasn’t been established; min-cost flow or other polynomial algorithms also not found (suspicion is this is a hard problem)
- Parameters:  \(k = \#\)clusters; \(M = \#\)process conditions; \(P = \#\)paths
  - \(k < 10\)?
  - \(M\) can be large – e.g., up to 1000?
  - \(P\) can be large, but this doesn’t matter too much
- Limiting cases to consider
  - \(k = 2\) clusters
  - All Pr the same, different sizes
  - All sizes the same, different Pr
- Cost = waste is not “intuitive” to work with, but it is monotone
  - Monotone: Can apply “DP-RP” on Uezono or other greedy ordering … (min-perimeter, max-adjacency, etc.)
  - Interesting observation: cost decreases monotonically from \(k = 2\) to \(k = M\)
- Many heuristics possible
  - B&B for \(k = 2\)? (then, apply recursively)
  - Better criterion (than Uezono’s) (with bounded error?) for clumping 2 process conditions together? For clumping 3 together? (then, apply iteratively)
  - Iterative improvement (KL-FM or better)? (apply to some initial clustering construction)
- Issue: “testcases” (validation plan)
  - Probability distribution over process conditions
  - Overlap between process conditions
Many-Core Placement (discussion #1)

- Problem statement = ?
- Qualitative (novel) goal: $\Theta(P)$ speedup on P processors, avoid “pollution” from previous works / think out of the box!
- Key background (~3-4 pp. tex)
  - Relevant methods for Partitioning and clustering; Global placement (not legal, but spread); Detailed placement $\rightarrow$ which are relevant, which are not, and why? (e.g., is “hierarchical” a priori off the table?)
  - Many-core architecture and HW-SW platform options
- New ideas = ?
  - How to combine non-disjoint solutions of subproblems?
  - Are changes to the traditional cluster-GP-DP flow warranted?
- Techniques to develop
- Schedule
  - To the extent that “DP” is orthogonal and smaller in space of possibilities, implementation can proceed in parallel with GP development?
- Testbed
  - Problem instances, quality metrics
  - Hardware (SDSC BlueGene clusters)
Many-Core Placement (discussion #2)

- Alpert / Nam (IBM) clustering method is in Aplace3.0 code
- Still suggest: need a “religious” stance here of “no serial computation” (cf. “New Ideas” from previous discussion)
  - “Absolute scalability” is your stake in the ground; everything else is built around it
  - What does this break in your approach? E.g., no top-down decomposition, no guarantees of disjoint clusters, …
  - Another stake in the ground: don’t use P processors to solve the same problem P times (bounded-discrepancy search, GWTW). And, assume that symmetry is broken somehow.
  - Can you place, say, 10*P clusters using P processors, where each cell belongs on average to 10 clusters? (P ~ sqrt(N))?
  - How to keep clusters from overlapping too much in any area?
  - How to deduce a cell placement from the cluster placement?
  - How to combine non-disjoint solutions of subproblems?
  - Are changes to the traditional cluster-GP-DP flow warranted?
- Details of how shared memory is actually accessed?
  - How are the clustering, the clusters, the original netlist, the induced netlist(s) over clusters, the placement (locations of clusters and cells): stored, accessed, updated?
Known Optimal Solutions (discussion #1)

- Problem Statement = ?
- Arena: **sizing**, placement, partitioning, routing, …
- Related Works (constructions with known optima; “scaling”)
  - Sizing: DAC10 “eye charts”
  - Placement/Partitioning: HHK95, PEKO/PEKU, “Planted Partition” Bui, Garbers, etc.
  - Routing: SLIP11 scaling
  - “Synthetic Benchmark” netlist constructions (Ghent ?, UCSC ?, Toronto circ/gen, …)
- Qualitative goals: “scalable” (= composable?), “realistic”, “general conclusions vs. tractability”
- Issues
  - How to simplify “sizing” so that it is tractable (e.g., DAC10 paper eliminated slew-dependence in its testbed)?
- **New** and/or basic ideas = ?
  - Metric of “realism” (see the “synthetic benchmark” literature; note that realism and tractability to analysis are opposing goals)
  - Maybe “known optimal” can be in a probabilistic sense (cf. planted partition idea) or a bounded sense (cf. HHK95 scaling suboptimality)
  - Always try to optimally solve as large a “small case” as possible (e.g., B&B or SA running for weeks gives a proof point later)
Planar Bounded-Skew Tree (discussion #1)

- Literature review ("annotated bibliography" kind of summary ~1 page) (5/4)
  - On Optimal Interconnections; Edahiro93 Greedy-DME, Planar (Zhu-Dai92, KahngT96); BST-DME; Charikar; Zelikovsky [Note: everyone should have such a summary ~5/4]

- Motivations / "fundamental questions to address"
  - Why single-layer / planar required? (Interposer in 3DIC; clock distribution on MCM substrate; avoid vias using "dedicated clock layer"?)
  - How does "planar" change the BST problem? E.g., how would you change KahngT96 when “Z” becomes “B”? (5/6)
  - Ignoring planarity, “BST” gives continuum between “ZST” and “SMT”. Then, “pBST” gives continuum between “pZST” (PlanarDME) and “SMT” should achieve such behavior as skew bound B is varied (in particular, should “match” other methods for B = 0, ∞)
  - What intuitions have you built so far? Pencil and paper: points on a line, on a circle, on a 2-D lattice (5/6)

- Code
  - BST-DME code from Bookshelf (need to get it to compile/run – 5/2)
  - Implement Charikar, Zelikovsky methods (what are “planar”, “bounded” versions?)

- Scope (are you still comfortable with PBST as the topic?)
  - Linear delay model (ignore Elmore)
  - No buffering (no clock gating, no sink polarity, no obstacles, …)
  - Topology design
    - Open: minimum-cost Planar BST consistent with given topology “topology given” is unclear
    - Charikar: Is there a “terminal ordering” (e.g., as in k-center heuristic) that gives a good topology somehow?
Outline

- Fiduccia-Mattheyses Hypergraph Partitioning
- Partitioning With Terminals
- More Tuning Examples
- Multilevel Partitioning and Experimental Reporting
- End-Case Processing
Hypergraphs in VLSI CAD

- Circuit netlist represented by hypergraph
Fiduccia-Mattheyses (FM) Approach

- **Pass:**
  - start with all vertices free to move (*unlocked*)
  - label each possible move with immediate change in cost that it causes (*gain*)
  - iteratively select and execute a move with highest gain, lock the moving vertex (i.e., cannot move again during the pass), and update affected gains
  - best solution seen during the pass is adopted as starting solution for next pass

- **FM:**
  - start with some initial solution
  - perform passes until a pass fails to improve solution quality
Figure 1. Example of cell gains

Figure 2. Bucket list structure

Figure 3. Critical nets
Cut During One Pass (Bipartitioning)
Key Elements of FM

- Three main operations
  - computation of initial gain values at beginning of pass
  - retrieval of the best-gain (feasible) move
  - update of all affected gain values after a move is made

- Contribution of Fiduccia and Mattheyses:
  - circuit hypergraphs are sparse
  - move gain is bounded between $+2 \times \text{max vertex degree}$, $-2 \times \text{max vertex degree}$
  - hash moves by gains (gain bucket structure)
  - each gain affected by a move is updated in constant time
  - linear time complexity per pass: $O(#\text{pins})$
Taxonomy of Algorithm and Implementation Improvements

- Modifications of the algorithm
- Implicit decisions
- Tuning that can change the result
- Tuning that cannot change the result
Modifications of the Algorithm

- Important changes to flow, new steps/features
  - lookahead tie-breaking (Krishnamurthy84)
  - CLIP (Dutt, Deng)
    - instead of actual gain, maintain “updated gain”
      = actual gain minus initial gain (at start of pass)
    - WHY ???
  - cut-line refinement
    - insert nodes into gain structure only if incident to cut nets
  - multiple unlocking (Dasdan, Aykanat)
Modifications of the Algorithm

- Important changes to flow, new steps/features
  - lookahead tie-breaking
  - CLIP
    - instead of actual gain, maintain “updated gain”
      = actual gain minus initial gain
    - promotes “clustered moves” (similar to “LIFO gain buckets”)
  - cut-line refinement
    - insert nodes into gain structure only if incident to cut nets
  - multiple unlocking
Implicit Decisions

- Tie-breaking in choosing highest gain bucket
- Tie-breaking in where to attach new element in gain bucket
  - LIFO vs. FIFO vs. random ... (known issue: HagenK95)
- Whether to update, or skip updating, when “delta gain” of a move is zero
- Tie-breaking when selecting the best solution seen during pass
  - first encountered, last encountered, best-balance, ...
Tuning That Can Change the Result

- Threshold large nets to reduce runtime
- Skip gain update for large nets
- Skip zero delta gain updates
  - changes resolution of hash collisions in gain container
- Loose/stable net removal
  - perform gain updates for only selected nets
- Allow illegal solutions during pass
Tuning That Can’t Change the Result

- Skip updates for nets that cannot have non-zero delta gain
- Netcut-specific optimizations
- 2-way specific optimizations
- Optimizations for nets of small degree
- ...

... 41 years since KL70, 29 years since FM82, 100’s of papers in literature
Zero Delta Gain Update

- When vertex $x$ is moved, gains for all vertices $y$ on nets incident to $x$ must potentially be updated.

- In all FM implementations, this is done by going through incident nets one at a time, computing changes in gain for vertices $y$ on these nets.

- Implicit decision:
  - reinsert a vertex $y$ when it experiences a zero delta gain move (will shift position of $y$ within the same gain bucket)
    - E.g., at head of gain bucket (what about “20% closer to the head”?)
  - skip the gain update (leave position of $y$ unchanged)
Tie-Breaking Between Highest-Gain Buckets

- Gain container typically implemented such that available moves are segregated, e.g., by source or destination partition

- There can be more than one highest-gain bucket

- When balance constraint is anything other than “exact bisection”, moves at multiple highest-gain buckets can be legal

- Implicit decision:
  - choose the move that is from the same partition as the last vertex moved (“toward”)
  - choose the move that is not from the same partition as the last vertex moved (“away”)
  - choose the move in partition 0 (“part0”)
How Much Can This Matter?

- 5% ?
- 10% ?
- 20% ?
# Implicit Decision Effects: IBM01

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>IBM01 with <strong>unit</strong> areas and <strong>10%</strong> balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>Flat LIFO</td>
</tr>
<tr>
<td>All ( \Delta \text{gain} ) Away</td>
<td>856/1723 (12.8)</td>
</tr>
<tr>
<td>All ( \Delta \text{gain} ) Part0</td>
<td>356/1226 (16.3)</td>
</tr>
<tr>
<td>All ( \Delta \text{gain} ) Toward</td>
<td>188/577 (12.6)</td>
</tr>
<tr>
<td>Nonzero Away</td>
<td>201/529 (8.44)</td>
</tr>
<tr>
<td>Nonzero Part0</td>
<td>201/436 (8.81)</td>
</tr>
<tr>
<td>Nonzero Toward</td>
<td>197/454 (9.29)</td>
</tr>
</tbody>
</table>
### Implicit Decision Effects: IBM02

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>IBM02 with <strong>unit</strong> areas and <strong>10%</strong> balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates Bias</td>
<td>Flat LIFO</td>
</tr>
<tr>
<td>All Δgain Away</td>
<td>402/1404  (30.8)</td>
</tr>
<tr>
<td>All Δgain Part0</td>
<td>307/1468  (43.2)</td>
</tr>
<tr>
<td>All Δgain Toward</td>
<td>283/585   (23.7)</td>
</tr>
<tr>
<td>Nonzero Away</td>
<td>275/471   (18.9)</td>
</tr>
<tr>
<td>Nonzero Part0</td>
<td>262/444   (18.4)</td>
</tr>
<tr>
<td>Nonzero Toward</td>
<td>265/453   (17.0)</td>
</tr>
</tbody>
</table>
Effect of Implicit Decisions

- Very large average cutsize difference for flat partitioner with worst vs. best combination
  - far outweighs “new improvements”

- One wrong decision can lead to misleading conclusions w.r.t. other decisions
  - “part0” is worse than “toward” with zero delta gain updates
  - better or same without zero delta gain updates

- Stronger optimization engines mask flaws
  - ML CLIP > ML LIFO > Flat CLIP > Flat LIFO
  - less dynamic range → ML masks bad flat implementation
Tuning Effects

- Comparison of two CLIP-FM implementation
- Min and Ave cutsizes from 100 single-start trials

<table>
<thead>
<tr>
<th>Tolerance</th>
<th>Algorithm CLIP</th>
<th>Ibm01</th>
<th>Ibm02</th>
<th>Ibm03</th>
<th>Ibm04</th>
<th>Ibm05</th>
<th>Ibm06</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>Paper1</td>
<td>Min</td>
<td>471</td>
<td>1228</td>
<td>2569</td>
<td>17782</td>
<td>1990</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ave</td>
<td><strong>2456</strong></td>
<td><strong>12158</strong></td>
<td><strong>16695</strong></td>
<td><strong>20178</strong></td>
<td><strong>3156</strong></td>
</tr>
<tr>
<td></td>
<td>Paper2</td>
<td>Min</td>
<td>329</td>
<td>298</td>
<td>797</td>
<td>653</td>
<td>2557</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ave</td>
<td><strong>485</strong></td>
<td><strong>472</strong></td>
<td><strong>1635</strong></td>
<td><strong>1233</strong></td>
<td><strong>3074</strong></td>
</tr>
<tr>
<td>10%</td>
<td>Paper1</td>
<td>Min</td>
<td>246</td>
<td>439</td>
<td>1915</td>
<td>488</td>
<td>2146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ave</td>
<td><strong>462</strong></td>
<td><strong>4163</strong></td>
<td><strong>9720</strong></td>
<td><strong>1232</strong></td>
<td><strong>3016</strong></td>
</tr>
<tr>
<td></td>
<td>Paper2</td>
<td>Min</td>
<td>237</td>
<td>266</td>
<td>675</td>
<td>527</td>
<td>1775</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ave</td>
<td><strong>424</strong></td>
<td><strong>406</strong></td>
<td><strong>1325</strong></td>
<td><strong>893</strong></td>
<td><strong>2880</strong></td>
</tr>
</tbody>
</table>

- Question: Why did this happen?
  - Original inventor of CLIP-FM couldn’t figure it out
  - **Hint**: some modern IBM benchmarks have large macro-cells
Corking Effect in CLIP

- CLIP begins by placing all moves into the 0-gain buckets
  - CLIP chooses moves by cumulative delta gain ("updated gain")
  - initially, every move has cumulative delta gain = 0

- Historical legacy (and for speed): FM partitioners typically look only at the first move in a bucket
  - if it is illegal, skip the rest of the bucket (possibly skip all buckets for that partition)

- If the move at the head of each bucket at the beginning of a CLIP pass is illegal, pass terminates without making any moves
  - even if first move is legal, an illegal move soon afterward will "cork"

- New test cases (IBM) have large cells
  - large cells have large degree, and often large initial gain
  - CLIP inventor couldn’t understand bad performance on IBM cases
Tuning to Uncork CLIP

- Don’t place nodes with area > balance constraint in gain container at pass initialization
  - actually, can be useful for all FM variants
  - zero CPU overhead

- Look beyond the first move in a bucket
  - extremely expensive
  - hurts quality (partitioner doesn’t operate well near balance tolerance)
  - not worth it, from our experience

- Simply do a LIFO pass before starting CLIP
  - spreads out nodes in gain buckets
  - reduces likelihood that large node has largest total gain
Outline

- Fiduccia-Mattheyses Hypergraph Partitioning
- Partitioning With Terminals
- More Tuning Examples
- Multilevel Partitioning and Experimental Reporting
- End-Case Processing
Goals

- Flavors of hypergraph partitioning
  - **abstract:** free hypergraph context $\rightarrow$ no terminals
  - **practical:** top-down placement context $\rightarrow$ terminals

- Terminals *change* the partitioning problem
  - empirical study of effects on FM performance

- New heuristics needed that exploit terminals
  - early pass termination in FM

- Open issues
Partitioning in the Research Literature

“Given hypergraph $H = (V,E)$, partition $V$ into $V_1$ and $V_2$ with $|V_1| \sim |V_2|$ so as to minimize the number of cut hyperedges…”

- balance constraints $\rightarrow$ NP-hard
- pass-based KLFM variants most successful

Benchmark-driven research

- partitioning benchmarks have no fixed-terminal information

Entire literature is on “free hypergraphs”
Recall: Partitioning in Top-Down Placement

- Global placement
  - map cells of netlist into layout area
  - satisfy performance constraints, minimize area

- Top-down divide-and-conquer approach

- Divide step: hypergraph partitioning
  - connections among blocks modeled as fixed vertices (terminals) in the partitioning instance
Placement Blocks Have Many Terminals!

- Rent’s rule: \( \#\text{terminals} = k \cdot (\#\text{cells})^p \)

- For given Rent parameter value \( p \), below what \( \#\text{cells} \) will more than \( y\% \) of vertices be terminals?

<table>
<thead>
<tr>
<th>Rent parameter</th>
<th>( y=5% )</th>
<th>( y=10% )</th>
<th>( y=20% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = 0.60 )</td>
<td>40992</td>
<td>7250</td>
<td>1281</td>
</tr>
<tr>
<td>( p = 0.65 )</td>
<td>186943</td>
<td>25800</td>
<td>3561</td>
</tr>
<tr>
<td>( p = 0.70 )</td>
<td>1413600</td>
<td>140250</td>
<td>13915</td>
</tr>
</tbody>
</table>
disconnect!

- Top-down placement *always* generates instances *with* fixed terminals

- Partitioning research has focused on instances *without* fixed terminals

**obvious questions**

- Is effect of terminals on algorithm performance sufficient to require new techniques?

- Can we exploit, rather than tolerate, terminals?
Demonstration: Effects of Terminals

- Experiment with *well-assigned* terminals
  - find “good solution”: best of 100 partitioner runs
  - make increasing % of nodes into terminals fixed as in good solution
  - “good solution” cost - *an upper bound for min cost* of all instances (by construction)
  - run partitioner again - how does it do?

- Expectations
  - problem gets easier as more terminals are fixed
  - smaller runtime, better average quality
Expectations Partly Wrong

- “Well-assigned” terminals can hurt!
  - good solutions are harder to find
  - spike at 5%
Presence of Terminals is Significant

- Interpretation of the spike
  - failure of FM
  - other heuristics may be more successful
Can We **Exploit Terminals?**

- Best for free hypergraphs ≠ best with terminals
  - need methods specifically to exploit terminals
  - different trade-offs/tunings of traditional heuristics

- Example: shorter passes in FM
  - terminals shorten the useful part of the pass (find best sooner)
  - Data: Average number of passes per run, and average percentage of nodes moved per pass (excluding the first pass) for 50 runs of LIFO-FM

<table>
<thead>
<tr>
<th>Testcase</th>
<th>0% Fixed</th>
<th>10% Fixed</th>
<th>20%Fixed</th>
<th>30%Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#P</td>
<td>%Moved</td>
<td>#P</td>
<td>%Moved</td>
</tr>
<tr>
<td>IBM 01</td>
<td>12</td>
<td>20.5%</td>
<td>9.9</td>
<td>9.1%</td>
</tr>
<tr>
<td>IBM 02</td>
<td>9.6</td>
<td>12.4%</td>
<td>8.1</td>
<td>6.1%</td>
</tr>
<tr>
<td>IBM 03</td>
<td>10</td>
<td>6.8%</td>
<td>8.3</td>
<td>5.7%</td>
</tr>
<tr>
<td>IBM 04</td>
<td>13</td>
<td>8.3%</td>
<td>9.9</td>
<td>4.4%</td>
</tr>
<tr>
<td>IBM 05</td>
<td>33</td>
<td>37.0%</td>
<td>13</td>
<td>4.7%</td>
</tr>
</tbody>
</table>
Terminals in FM Partitioning

No Fixed Terminals

Fixed Terminals
FM Partitioning With Pass Limits

- Allow *at most x% of nodes* to be moved in a pass

<table>
<thead>
<tr>
<th>Max % To Move</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>nolimit</td>
<td>596.2</td>
<td>1041.8</td>
<td>513.8</td>
<td>303.4</td>
</tr>
<tr>
<td>50%</td>
<td>855.9</td>
<td>1027.3</td>
<td>555.2</td>
<td>290.3</td>
</tr>
<tr>
<td>IBM01 25%</td>
<td>959.4</td>
<td>1249.1</td>
<td>521.7</td>
<td>297.4</td>
</tr>
<tr>
<td>10%</td>
<td>1233.6</td>
<td>1435.3</td>
<td>514.7</td>
<td>298.0</td>
</tr>
</tbody>
</table>

- **Pass limits:**
  - hurt results when there are no terminals
  - help given “sufficiently many“ terminals
Observations

- Fixed terminals matter
- Current methods do not adequately comprehend presence of fixed terminals
- Better methods to exploit fixed terminals are possible
- Many questions open
  - Quantify effects of terminals
  - Interpret “sufficiently many terminals” (“# terminals” is meaningless in general)
  - Explain nonmonotonicity for < 5% fixed terminals
  - Variant pass-limiting schemes
  - Stronger effects for multi-way?
  - New heuristics specialized for fixed terminals
Outline

- Fiduccia-Mattheyses Hypergraph Partitioning
- Partitioning With Terminals
- More Tuning Examples
- Multilevel Partitioning and Experimental Reporting
- End-Case Processing
More FM Improvements (VRW)

- **Initial solution**
  - typically chosen to be balanced ("legal")
    - all moves preserve balance
  - we choose "very illegal" initial solution (**VILE**)
    - all vertices are initially in one partition
    - vertex moves must not worsen the balance
    - balance is easily restored in 1-2 FM passes

- **Randomization**
  - classic FM is *deterministic* (except for init. sol.)
  - we *randomly reorder vertices* before each pass
  - \( \Rightarrow \) FM can use VILE and better escape local min.
New Improvement (VRW)

- Recall
  - classic FM and its variants prioritize moves/vertices
  - leaves ties among equal-priority moves
  - when vertex moves, its neighbors are reprioritized
  - tie-breaking seriously affects performance of FM
    - Krishnamurthy used it in look-ahead partitioner (slow)
    - Hagen/Kahng: LIFO FM is better than FIFO FM
  - Unused degrees of freedom exist

- We break ties toward neighbors of fixed vertices
  - at the start of each pass,
    move fixed vertices back and forth (“wiggle”)
  - this reprioritizes the neighbors of fixed vertices
Experimental Methodology

- Benchmarks with fixed vertices
  - ISPD 99 suite, produced from ISPD `98 circuits from IBM

- Comparison methodology
  - run-time / solution quality trade-off: record solution quality achieved in given time

- Compare average results of $K$ starts of
  - LIFO FM (Hagen/Kahng `94)
  - CLIP FM (Dutt/Deng `95)
  - FM VRW (this method)
<table>
<thead>
<tr>
<th>Algo</th>
<th>1 start</th>
<th>2 starts</th>
<th>4 starts</th>
<th>8 starts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IBM01D H 2%, 6139 movable, 2155 fixed, 7330 nets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>706.6(1.7)</td>
<td>648.8(3.4)</td>
<td>600.5(6.7)</td>
<td>566.2(13.4)</td>
</tr>
<tr>
<td>CLIP</td>
<td>654.7(5.4)</td>
<td>593.6(10.8)</td>
<td>537.2(21.7)</td>
<td>492.0(43.4)</td>
</tr>
<tr>
<td>VRW</td>
<td>624.2(1.7)</td>
<td>588.6(3.4)</td>
<td>570.6(6.7)</td>
<td>555.5(13.4)</td>
</tr>
<tr>
<td></td>
<td>IBM01D H 10%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>676.9(1.6)</td>
<td>612.3(3.2)</td>
<td>562.8(6.5)</td>
<td>504.2(12.9)</td>
</tr>
<tr>
<td>CLIP</td>
<td>482.4(6.0)</td>
<td>412.0(12.1)</td>
<td>368.3(24.2)</td>
<td>348.3(48.4)</td>
</tr>
<tr>
<td>VRW</td>
<td>419.9(1.4)</td>
<td>403.8(2.9)</td>
<td>395.9(5.8)</td>
<td>390.4(11.6)</td>
</tr>
<tr>
<td></td>
<td>IBM01D V 2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>664.3(2.1)</td>
<td>600.0(4.2)</td>
<td>557.8(8.3)</td>
<td>521.9(16.6)</td>
</tr>
<tr>
<td>CLIP</td>
<td>587.7(5.5)</td>
<td>536.8(11.1)</td>
<td>499.0(22.2)</td>
<td>471.2(44.4)</td>
</tr>
<tr>
<td>VRW</td>
<td>535.6(1.8)</td>
<td>508.3(3.7)</td>
<td>489.4(7.4)</td>
<td>480.1(14.8)</td>
</tr>
<tr>
<td></td>
<td>IBM01D V 10%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>623.9(1.8)</td>
<td>582.2(3.51)</td>
<td>551.3(7.0)</td>
<td>528.9(14.1)</td>
</tr>
<tr>
<td>CLIP</td>
<td>559.4(5.6)</td>
<td>521.3(11.2)</td>
<td>488.3(22.4)</td>
<td>465.2(44.8)</td>
</tr>
<tr>
<td>VRW</td>
<td>519.8(1.4)</td>
<td>494.6(2.8)</td>
<td>474.6(5.5)</td>
<td>461.9(11.1)</td>
</tr>
</tbody>
</table>
Observations

- Winning combinations are in bold in table
  - all FM VRW combinations are winning
  - FM VRW is superior to LIFO FM and CLIP FM

- Extending LIFO FM to FM VRW
  - Very simple (one-day effort)

- With very many terminals (see proceedings)
  - FM VRW is faster than hMetis, similar quality

- With few terminals (see proceedings)
  - FM VRW is better on some benchmarks, much worse on others
  - can perform both LIFO FM and FM VRW starts
Outline

- Fiduccia-Mattheyses Hypergraph Partitioning
- Partitioning With Terminals
- More Tuning Examples
- Multilevel Partitioning and Experimental Reporting
- End-Case Processing
Multilevel Partitioning

- **Used in the most efficient partitioners**
  - Karypis et al., DAC `97
  - Alpert et al., DAC `97

- **Basic idea (two levels)**
  - group original vertices together (in clusters)
  - induce clustered hypergraph
  - partition clustered hypergraph
  - assign original vertices to set1/set2 accordingly
  - refine by additional FM passes

- Also see 1998 paper by Wichlund and Aas
Multilevel Partitioning

Clustering

Refinement
Multi-level Partitioning

- Multi-level FM (MLFM)
  - does not directly use an initial partitioning
  - much better than “flat” FM, produces very good solutions in near-linear time
  - critical to performance of top-down placers

- Implementation choices in MLFM
  - V-cycling (“iterative ML”) - Karypis et al., DAC’97
    - a method of using an initial solution
    - avoids clustering vertices that are in different sets
    - allows us to run ML to improve results of previous runs
  - Top-level partitioning with small tolerance
    - first partition top level with lax tolerance
    - use the result as initial solution for another FM run
    - decrease tolerance to what it should be, run FM

- New clustering methods
- Of course, lots of tuning (solution pool size, un/coarsening ratios, etc.)

- One main conclusion: Locally optimized implementations look very different!
Clustering in Multilevel

- Needs to be fast
- HEM (Heavy-Edge Matching)
- EC (Edge Clustering) of Karypis et al.
  - for all pairs of adjacent vertices/clusters, add up weights of all hyperedges incident to both
  - every hyperedge contributes weight/degree (= 1/k model)
  - merge pairs in order of total weights
- PinEC
  - hyperedges of degree==2 contribute 2*weight, all others contribute weight
    - this corresponds to #pins removed by merging
  - Seems to work better in UCLA MLPart, but there are many other contextual differences (#V-cycles, #starts, …)
Reporting of Metaheuristic Experiments

- Long-discussed in the metaheuristics community (Gent94, Barr95)
- BSF = resource-equalized comparison of metaheuristics
Outline

- Fiduccia-Mattheyses Hypergraph Partitioning
- Partitioning With Terminals
- More Tuning Examples
- Multilevel Partitioning and Experimental Reporting
- End-Case Processing
Balanced Hypergraph Partitioning in Placement

- **Leading heuristics**
  - Kernighan-Lin (KL) `70, Fiduccia-Mattheyses (FM) `82,
  - Dutt and Deng (CLIP) `95, Multi-Level FM (MLFM) `90s

- **Advantages**
  - represents a “divide and conquer” approach
  - well-studied, fast, successful in practice

- **Disadvantages**
  - “wrong objective” for routable placement?
    - under-utilizes routing area over the cutlines!!!
  - few non-netcut objectives can be efficiently optimized
  - timing constraints difficult to capture
Partitioning-Driven Top-down Placement

- Placement blocks represent cells and layout area

- Each partitioning round:
  #blocks doubles, sizes halve
  Eventually, several cells in tiny region
Reduction of Top-Down Placement to Balanced Hypergraph Partitioning

- Blocks typically cut in the longer direction
  - Cuts *perpendicular* to rows
    - target balance 50/50
    - cut line adjusted after partitioning
      → large tolerance okay
  - Cuts *parallel* to rows
    - target determined by configuration of rows in block
    - cannot adjust cut line after partitioning
      → tolerance determined by whitespace, i.e., unused sites

- Some “subtleties”
  - Discrete legal cell locations (sites) in rows
  - Constraints on cell locations, e.g., fixed pads
  - Relationship of top-down mincut to net HPWL objective (recall: HPWL = practically optimizable estimator)?
Difficulties in Top-Down Placement

- Small partitioning tolerance
  - typically under few % of total cell area, often set to min cell size
  - smaller, more fragmented solution space
  - → iterative partitioners are hampered
  - problem worsens with modern cell libraries (more macro functions, larger range of drive strengths)
  - existing workarounds are distant from standard FM

- Small problem instances
  - few cells in block → each is a large % of total
  - cells larger than tolerances cannot move

→ Small instances harder for iterative partitioners
How to Improve Wirelength, Routability?

- Partitioning of large instances difficult to improve
  - use hMetis or MLPart

- Avoid very small partitioning instances
  - increase end-case placement threshold

- Avoid iterative partitioning on small instances
  - use enumerative and branch-and-bound methods

- Experimental questions:
  - does this improve placement quality?
  - how suboptimal is FM?
  - how much faster is FM than optimal partitioning?
  - what are the practical size limits for end-case placement and optimal partitioning?
Methods and Tools

- Complete top-down placer
  - reads industry-standard formats (Cadence LEF/DEF)
  - produces results comparable to commercial tools
- Partitioning and placement instances saved
- Optimal partitioners and placers
  - enumeration accelerated with Gray codes
  - straightforward branch-and-bound (no polytopes etc)
- Optimal partitioners/placers quickly implementable
Gray Code Enumeration of Partitionings

- Use model
  - traverse all solutions without repetition
  - re-assigning one node gives new solution: O(1) update
  - incrementally maintain partition balances and net cut
  - faster than lexicographic enumeration, where average-case update takes O(N) time
  - save best-seen solution

- Software for Gray code generation
  - simple: dozen lines in C (see paper)
  - lightning-fast, but memory hog
  - also works for k-way
B&B for Balanced Partitioning

- Not all solutions are traversed
- Based on lexicographic ordering of partitionings
- Assumes an ordering of nodes
- Branch-and-bound as a finite state machine
  - start with all nodes unassigned
  - assign/unassign nodes one by one using a stack
  - traverse the lexicographic tree of partial solutions
- Maintain "cut so far" on the stack
- Save first/best-seen complete solutions
- Prune branches with cut as big as best-seen
B&B for Balanced Partitioning

- Not all solutions traversed
- Based on lexicographic ordering of partitionings
- Assumes an ordering of nodes
- Branch-and-bound as a finite state machine
- Maintain "cut so far" on the stack
  - when node assigned (branching), cut can not decrease
  - when node unassigned (backtracking), cut readily available
- Save first/best-seen complete solutions
- Prune branches with cut as big as best-seen
B&B for Balanced Partitioning

- Not all solutions traversed
- Based on lexicographic ordering of partitionings
- Assumes an ordering of nodes
- Branch-and-bound as a finite state machine
- Maintain "cut so far" on the stack

- When first/best-seen complete solution reached
  - save solution
  - update best cut seen
  - unassign nodes (backtrack) to examine other branches

- Prune branches with cut as big as best-seen
Comparison Methodology

- Partitioning instances saved by placer
  - each instance has 10-50 nodes
  - instances binned by size for averaging
  - format publicly available (see ISPD-99 proceedings)

- “Good” instances
  - non-zero optimal cut (want to divide FM cut by optimal cut)
  - branch-and-bound takes longer than 0.0001 sec
    (FM always takes much longer)

- Indicators
  - run time ratio (X starts of FM vs one B&B run)
  - cut ratio (best of X starts of FM vs one B&B run)
## Comparison with LIFO-FM

| Num Nodes | Num Instances (Good) | Num Instances Sub Opt. | Num Instances | Num Instances | Num Instances | Num Instances | Num Instances | Num Instances | Num Instances | Num Instances |
|-----------|----------------------|------------------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| 10        | 160(134)             | 32                     | 20.73         | 1.976         | 41.46         | 1.7           | 62.19         | 1.564         | 2073.1        | 1.08          |
| 12        | 94(83)               | 8                      | 17.02         | 1.948         | 34.05         | 1.671         | 51.08         | 1.537         | 1702.7        | 1.029         |
| 14        | 58(55)               | 11                     | 11.14         | 1.892         | 22.29         | 1.623         | 33.44         | 1.496         | 1114.9        | 1.042         |
| 16        | 65(62)               | 20                     | 6.796         | 1.846         | 13.59         | 1.634         | 20.38         | 1.53          | 679.6         | 1.053         |
| 18        | 40(40)               | 25                     | 4.43          | 1.907         | 8.86          | 1.717         | 13.29         | 1.628         | 443           | 1.149         |
| 20        | 42(40)               | 29                     | 2.761         | 1.913         | 5.523         | 1.726         | 8.284         | 1.635         | 276.1         | 1.178         |
| 22        | 27(27)               | 22                     | 1.429         | 2.001         | 2.857         | 1.81          | 4.286         | 1.721         | 142.8         | 1.217         |
| 24        | 30(30)               | 27                     | 0.871         | 2.088         | 1.743         | 1.896         | 2.614         | 1.805         | 87.14         | 1.294         |
| 26        | 38(38)               | 38                     | 0.512         | 2.368         | 1.023         | 2.171         | 1.535         | 2.072         | 51.16         | 1.512         |
| 28        | 31(31)               | 31                     | 0.357         | 2.227         | 0.713         | 2.054         | 1.07          | 1.963         | 35.67         | 1.468         |
| 30        | 25(25)               | 24                     | 0.151         | 1.973         | 0.302         | 1.834         | 0.453         | 1.765         | 15.11         | 1.39          |
| 32        | 13(13)               | 9                      | 0.261         | 1.698         | 0.522         | 1.595         | 0.783         | 1.55          | 26.08         | 1.287         |
| 34        | 13(13)               | 13                     | 0.078         | 2.773         | 0.155         | 2.562         | 0.233         | 2.447         | 7.759         | 1.816         |
## Comparison with CLIP-FM

<table>
<thead>
<tr>
<th>Num Nodes</th>
<th>Num Instances (Good)</th>
<th>Num Instances Sub Opt.</th>
<th>1 Start Time</th>
<th>1 Start Cut</th>
<th>2 Starts Time</th>
<th>2 Starts Cut</th>
<th>3 Starts Time</th>
<th>3 Starts Cut</th>
<th>100 Starts Time</th>
<th>100 Starts Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>160(124)</td>
<td>27</td>
<td>24.23</td>
<td>1.971</td>
<td>48.47</td>
<td>1.688</td>
<td>72.71</td>
<td>1.552</td>
<td>2423.8</td>
<td>1.07</td>
</tr>
<tr>
<td>12</td>
<td>94(86)</td>
<td>9</td>
<td>17.96</td>
<td>1.985</td>
<td>35.93</td>
<td>1.698</td>
<td>53.9</td>
<td>1.563</td>
<td>1796.8</td>
<td>1.035</td>
</tr>
<tr>
<td>14</td>
<td>58(55)</td>
<td>9</td>
<td>10.47</td>
<td>1.867</td>
<td>20.95</td>
<td>1.601</td>
<td>31.43</td>
<td>1.473</td>
<td>1047.9</td>
<td>1.036</td>
</tr>
<tr>
<td>16</td>
<td>65(65)</td>
<td>26</td>
<td>7.488</td>
<td>1.89</td>
<td>14.97</td>
<td>1.67</td>
<td>22.46</td>
<td>1.564</td>
<td>748.7</td>
<td>1.099</td>
</tr>
<tr>
<td>18</td>
<td>40(40)</td>
<td>26</td>
<td>3.926</td>
<td>1.908</td>
<td>7.851</td>
<td>1.72</td>
<td>11.77</td>
<td>1.623</td>
<td>392.5</td>
<td>1.157</td>
</tr>
<tr>
<td>20</td>
<td>42(42)</td>
<td>29</td>
<td>3.15</td>
<td>1.922</td>
<td>6.301</td>
<td>1.736</td>
<td>9.451</td>
<td>1.645</td>
<td>315</td>
<td>1.177</td>
</tr>
<tr>
<td>22</td>
<td>27(27)</td>
<td>21</td>
<td>1.422</td>
<td>1.999</td>
<td>2.843</td>
<td>1.817</td>
<td>4.265</td>
<td>1.72</td>
<td>142.1</td>
<td>1.245</td>
</tr>
<tr>
<td>24</td>
<td>30(30)</td>
<td>29</td>
<td>0.923</td>
<td>2.1</td>
<td>1.846</td>
<td>1.912</td>
<td>2.769</td>
<td>1.818</td>
<td>92.29</td>
<td>1.3</td>
</tr>
<tr>
<td>26</td>
<td>38(38)</td>
<td>37</td>
<td>0.519</td>
<td>2.38</td>
<td>1.037</td>
<td>2.185</td>
<td>1.556</td>
<td>2.086</td>
<td>51.86</td>
<td>1.541</td>
</tr>
<tr>
<td>28</td>
<td>31(31)</td>
<td>31</td>
<td>0.361</td>
<td>2.219</td>
<td>0.723</td>
<td>2.038</td>
<td>1.084</td>
<td>1.947</td>
<td>36.13</td>
<td>1.421</td>
</tr>
<tr>
<td>30</td>
<td>25(25)</td>
<td>24</td>
<td>0.155</td>
<td>1.988</td>
<td>0.311</td>
<td>1.849</td>
<td>0.466</td>
<td>1.781</td>
<td>15.53</td>
<td>1.369</td>
</tr>
<tr>
<td>32</td>
<td>13(13)</td>
<td>9</td>
<td>0.289</td>
<td>1.691</td>
<td>0.578</td>
<td>1.593</td>
<td>0.867</td>
<td>1.554</td>
<td>28.88</td>
<td>1.305</td>
</tr>
<tr>
<td>34</td>
<td>13(13)</td>
<td>13</td>
<td>0.08</td>
<td>2.747</td>
<td>0.161</td>
<td>2.54</td>
<td>0.241</td>
<td>2.427</td>
<td>8.049</td>
<td>1.816</td>
</tr>
</tbody>
</table>
For small partitioning instances saved from top-down placement:

- Gray code enumeration is faster than B&B for up to 9 nodes
- One start of FM and CLIP FM on a problem with 10 nodes is 20-23 times (!) slower than B&B on average
- CPU time ratio decreases to one at 23-30 nodes
- One start of FM or CLIP FM produces solutions with twice the optimal cut (on average)