Optimal Partitioners and End-Case Placers For Standard-Cell Layout

Andrew E. Caldwell, Andrew B. Kahng and Igor L. Markov

UCLA VLSI CAD Lab ABKGroup

http://vlsicad.cs.ucla.edu

{caldwell,abk,imarkov}@cs.ucla.edu

April 13, 1998

Supported by Cadence

Outline of the Talk

- Introduction
  - row-based placement; bounding box (HPWL) objective
  - min-cut partitioning
  - top-down partitioning-driven placement
  - end-case placement problems solved optimally
  - small partitioning instances and iterative heuristics
- Experimental questions
- Methods
- Results
- Conclusions and future work

Outline of the Talk

- Introduction
- Experimental questions
  - can placement results be improved by increasing thresholds:
    - for optimally solved placement end-cases?
    - for optimally partitioned instances?
  - find size and CPU time limits for optimal partitioners?
  - how suboptimal are FM solutions on average?
- Methods
- Results
- Conclusions and future work

Outline of the Talk

- Introduction
- Experimental questions
- Methods
  - robust, fully functional top-down placer
  - Gray code enumeration based optimal partitioners
  - branch-and-bound optimal partitioners and placers
- Results
- Conclusions and future work

Outline of the Talk

- Introduction
- Experimental questions
- Methods
- Results
  - optimal partitioning and placement helps
  - FM and CLIP FM solutions highly suboptimal on average
  - can quickly bipartition instances of up to 30 nodes
  - 0.01-0.1 sec per instance on average
  - ... plus some rather unexpected results!
- Conclusions and future work

Row-based VLSI Placement

- Circuit hypergraph
- Discrete legal cell locations (sites) in rows
- Constraints on cell locations, e.g., fixed pads
- Objective function for cell placement
- Find:
  - cell locations that minimize objective subject to constraints
- Additional considerations
  - timing- and routability-driven placement
  - problem sizes of interest: 10^5 and above

Half-Perimeter Bounding Box Objective

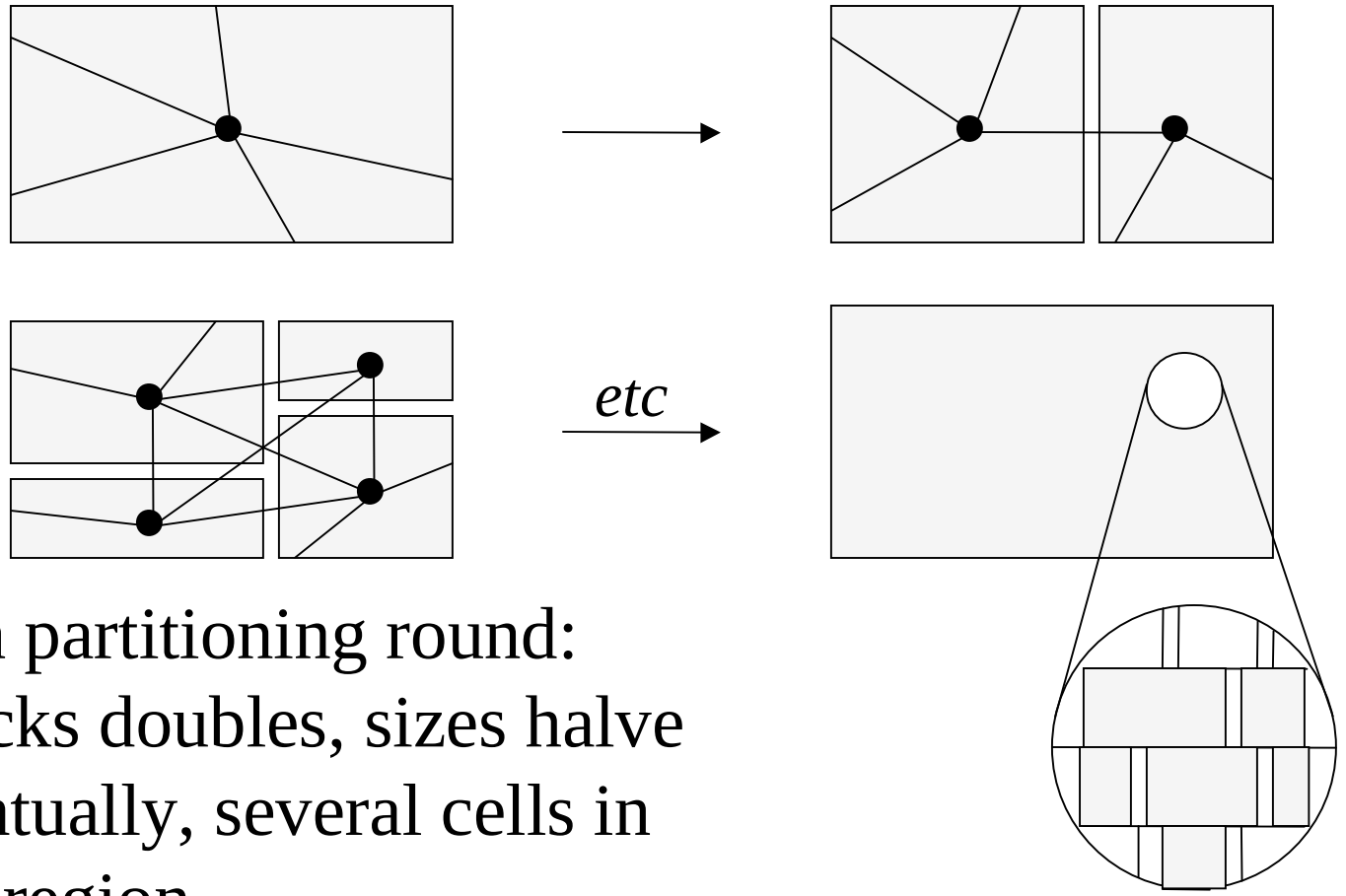
- h+w of bounding box of pin locations of each net
- Compromise between accuracy and efficiency:
  - easy to evaluate, approximates routed wirelength
- Direct minimization is slow, requires constraint relaxations
- Typically optimized by *recursive mincut bisection*
- Analytical placement can guide partitioning

Balanced Hypergraph Partitioning

- 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

      $\rightarrow$ large tolerance okay
  - Cuts ***parallel*** to rows
    - target determined by configuration of rows in block
    - cannot adjust cut line after partitioning

      $\rightarrow$ tolerance determined by whitespace, i.e., unused sites

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
    $\rightarrow$ 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 $\rightarrow$ each is a large % of total
  - cells larger than tolerances cannot move

$\rightarrow$ Small instances harder for iterative partitioners

How to improve wirelength, routability?

- Partitioning of large instances difficult to improve
  - use hMetis libraries (best available)
- Avoid very small partitioning instances

  $\rightarrow$ increase end-case placement threshold
- Avoid iterative partitioning on small instances

  $\rightarrow$ 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?
  - 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		1 Start		2 Starts		3 Starts		100 Starts	
	(Good)	Sub Opt.	Time	Cut	Time	Cut	Time	Cut	Time	Cut
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

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

Optimal partitioning vs FM: Results

- 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)

B&B for end-case placement

- Models cell placements as permutations
- Not all permutations need to be traversed
- Permutations ordered lexicographically
- B&B as a state machine
  - starts with no cells placed (all cells in a queue)
  - packs cells left to right (stack-like)
  - the rightmost cell can be popped AND enqueued
  - "wirelength so far" maintained on the stack
- Cost maintenance
  - when node is added (branching), cost cannot decrease
  - when node is removed(backtracking), cost readily available

Improving top-down placement

- Objectives
  - final placement cost
  - running time
- Degrees of freedom
  - end-case placer threshold
  - optimal partitioner threshold

Placement Results

Threshold		Test Case 1		Test Case 2		Test Case 3		Test Case 4		Test Case 5	
Part.	Placer	WL	Time	WL	Time	WL	Time	WL	Time	WL	Time
0	4	6.81	57	5.36	178	3.74	204	3.83	208	7.09	399
0	5	6.74	48	5.43	159	3.75	186	3.82	188	7.02	359
0	6	6.73	40	5.43	146	3.7	172	3.8	175	6.94	329
0	7	6.79	38	5.36	143	3.68	166	3.78	170	6.95	323
0	8	6.65	40	5.28	164	3.68	187	3.76	190	6.9	365
10	4	6.65	34	5.3	130	3.7	146	3.76	151	6.91	294
10	5	6.65	33	5.25	125	3.7	143	3.77	148	6.97	287
10	6	6.72	31	5.25	124	3.68	143	3.75	147	6.88	282
10	7	6.59	34	5.2	130	3.65	148	3.74	150	6.87	290
10	8	6.69	45	5.25	154	3.65	182	3.73	180	6.9	348
30	4	6.45	22	5.15	105	3.5	121	3.65	129	6.7	249
30	5	6.36	22	5.14	103	3.48	118	3.64	127	6.58	242
30	6	6.37	22	5.15	101	3.49	117	3.66	126	6.59	239
30	7	6.35	24	5.15	107	3.47	124	3.64	130	6.6	254
30	8	6.34	33	5.12	132	3.44	162	3.61	159	6.53	311
40	4	6.28	30	5.11	121	3.42	140	3.61	175	6.47	328
40	5	6.3	27	5.08	117	3.38	138	3.6	174	6.48	300
40	6	6.3	29	5.04	128	3.4	152	3.62	168	6.44	316
40	7	6.26	31	5.07	131	3.35	183	3.58	280	6.44	299
40	8	6.25	38	4.98	158	3.34	175	3.56	200	6.44	389

Summary of Results

- We can quickly solve
  - end-case placement problems of up to eight nodes
  - optimal partitioning for 30-35 nodes or less
- Higher end-case (placement) threshold is useful
  - smaller final wirelength
  - smaller overhead for creating partitioning and placement problems
- Optimal partitioning of small blocks is useful
  - **2x cut improvement**
  - **run-time improvement over FM with ≤ 25 nodes**
- 50% faster, 10% better WL

Current and Future Work

- Branch-and-bound codes: ***enabling technique***
  - big bag of tricks, most of which work
  - code complexity a big problem
  - challenge to object-oriented design
- Branch-and-bound codes: other applications
  - partitioning and placement onto FPGA architectures
  - ``essential'' subcircuits in large-scale circuit partitioning
- Further placement & routability improvements