Capo: Robust and Scalable Open-Source Min-Cut Floorplacer

Jarrod A. Roy, David A. Papa, Saurabh N. Adya*,
Hayward H. Chan, Aaron N. Ng, James F. Lu, Igor L. Markov
University of Michigan, EECS Department, Ann Arbor, MI 48109-2122
* Synplicity Inc., 600 W. California Ave., Sunnyvale, CA 95054
{royj,iamyou,hhchan,aaronnn,jflu,imarkov}@umich.edu * saurabh@synplicity.com

ABSTRACT

In this invited note we describe Capo, an open-source software tool for cell placement, mixed-size placement and floorplanning with emphasis on routability. Capo is among the fastest academic placers and scales to millions of movable objects. This note surveys the overall structure of Capo, discusses recent improvements and describes ongoing research.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Design

Keywords

Physical Design, Placement, Floorplanning

1. INTRODUCTION

The success of min-cut techniques in fixed-die placement is based on the speed and strength of multi-level hypergraph partitioners, the convenient top-down framework that efficiently captures available on-chip resources, and the fact that modern VLSI circuits admit a large number of good placements, which include slicing placements. The recent trend for large amounts of whitespace, clearly visible in the ISPD05 contest benchmarks, particularly increases the flexibility in the placement problem. Further, recent research alleviated several known weaknesses of min-cut placement.

The Capo placer first released at DAC2000 [10] sought to produce routable placements with a pure min-cut algorithm. To this end, Capo 8.0 was successful for most industrial benchmarks evaluated, even though it did not build or use congestion maps. For example, it produced a routable placement of an industrial design with 200K cells in 1.5 hours on a single-processor workstation. Its overall performance was on par with commercial tools. However an ISPD 2002 paper [18] proposed a new set of benchmarks on which Capo was less successful compared to a newer tool, Dragon. Dragon found routable placements in most cases by building congestion maps and biasing the placement process accordingly. This suggested that congestion-driven placement was far from solved and several papers in 2004 reported even better results.

Detailed analysis of the experiments in [18] revealed that superficial differences in parsers caused Dragon and Capo to be evaluated on incompatible variants of the same benchmarks. Specifically, the Capo variant had additional obstacles scattered throughout the core region, which decreased available whitespace and otherwise complicated the layout. The Capo-specific benchmarks have recently been withdrawn. Capo's parser can now process the original benchmarks and after recent algorithmic improvements surveyed below, Capo 9.1 placements typically take less time to route and have better routed wirelength (see Table 1); in all cases, the use of Capo led to fewer vias. This is remarkable as Capo does not use congestion maps and runs 8-15 times faster (3-6 with feedback) than Dragon.

Earlier versions of Capo distributed whitespace approximately uniformly, according to the hierarchical whitespace distribution formula from [13]. However more recent work [1] introduces tunable whitespace distribution for improved wirelength, while preserving a minimum amount of local whitespace in most regions to ensure routability. Whitespace allocation and detail placement have been further improved by analyzing the performance of Capo on *feature benchmarks* [15] designed to stress different aspects of placers.

Unlike Dragon and FengShui [5], Capo does not explicitly use multi-way partitioning. The recent addition of *placement feedback* [14] counteracts this potential limitation. Additionally, cutline shifting in recursive bisection adds flexibility in partition shapes and sizes, as well as whitespace allocation; this is not readily available in direct min-cut multi-way partitioning.

Using the min-cut floorplacement algorithm from [3], Capo 9.1 performs (i) scalable multi-way partitioning, (ii) state-of-the art standard-cell placement, (iii) integrated mixed-size placement competitive with best published results and (iv) wirelength-driven fixed-outline floorplanning that outperforms existing floorplanners by far.

For the last three years, Capo has been improving at the rate of 2-3% per year on standard benchmarks, but exhibits larger improvements on select proprietary benchmarks. Active ongoing development promises further improvements. Source code and executables of Capo 9.1 are available at **http://vlsicad.eecs.umich.edu/BK/PDtools/**

2. MIN-CUT PLACEMENT IN CAPO

Row-Based Placement. Internally, Capo's placement representation closely resembles the LEF/DEF and Bookshelf [12] file formats, which represent row information in standard-cell layout. Configurations of rows supply constraints for cell placement. Each row consists of non-overlapping subrows aligned to the coordinate of the row. All subrows in a row share the same coordinate, height, site width and site spacing. Placement instances in the Bookshelf format consist of several rows composed of one or more subrows.

Fixed objects may displace sites in the core region. Since fixed objects prevent standard cells from being placed in those sites, they are *obstacles*. Figure 1 shows two placements with many such obstacles. To prevent the placer from using sites occupied by obstacles, one solution is to remove the sites beneath all fixed objects. Capo accomplishes this by fracturing the rows containing the occupied sites into subrows, excluding the sites beneath the obstacle [10, Sec. 4.2]. The result is a row-based placement structure containing only legal locations for placing standard cells.

ISPD'05, April 3–6, 2005, San Francisco, California, USA.
ACM 1-59593-021-3/05/0004.

	"Easy" variants of IBM Dragon v2 benchmark suite [18]															
Benchmarks →	ibm01		ibm02		ibm07		ibm08		ibm09		ibm10		ibm11		ibm12	
↓ Placers ↓	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol
Capo 9.1	**7.8e5**	0	2.2e6	0	4.5e6	0	4.5e6	0	**3.4e6**	0	**6.6e6**	0	**5.0e6**	0	**9.9e6**	0
Dragon 3.01 [18]	8.4e5	0	**2.1e6**	0	**4.5e6**	0	4.6e6	0	3.7e6	0	7.0e6	0	5.4e6	0	1.1e7	0
FengShui 2.6 [5]	time-out 24h		2.2e6	0	4.8e6	77	**4.5e6**	0	3.5e6	0	6.8e6	0	5.3e6	0	1.0e7	33

	"Hard" variants of IBM Dragon v2 benchmark suite [18]															
Benchmarks →	ibm01		ibm02		ibm07		ibm08		ibm09		ibm10		ibm11		ibm12	
↓ Placers ↓	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol	rWL	viol
Capo 9.1	**7.7e5**	23	**2.1e6**	0	4.6e6	0	**4.8e6**	0	**3.3e6**	0	**6.5e6**	0	**4.9e6**	0	1.0e7	0
Dragon 3.01 [18]	9.1e5	84	2.2e6	0	**4.5e6**	0	5.0e6	0	3.5e6	0	7.0e6	0	5.4e6	0	**9.9e6**	0
FengShui 2.6 [5]	time-out 24h		2.3e6	0	4.7e6	251	5.1e6	52	3.4e6	0	6.7e6	0	5.3e6	0	time-out 24h	

Table 1: Comparison of three leading placers by routed wirelength and violation counts. In all cases Capo produces best via counts.

Min-Cut Bisection. Top-down placement algorithms seek to decompose a given placement instance into smaller instances by subdividing the placement region, assigning modules to subregions and cutting the netlist hypergraph [10]. Min-cut placers generally use either bisection or quadrisection to divide the placement area and netlist. Capo uses bisection as it allows for greater flexibility in cutline shifting to adapt to changing partition sizes [10, Sec. 3.2].

Each hypergraph partitioning instance is induced from a rectangular region, or bin, in the layout. In this context a *placement bin* represents (i) a placement region with allowed module locations (*sites*), (ii) a collection of circuit modules to be placed in this region, (iii) all signal nets incident to the modules in the region, and (iv) fixed cells and pins outside the region that are adjacent to modules in the region (*terminals*). Top-down placement can be viewed as a sequence of passes where each pass examines all bins and divides some of them into smaller bins.

Capo implements three types of min-cut partitioners – optimal (branch-and-bound [11]), middle-range (Fiduccia-Mattheyses [8]) and large-scale (multi-level Fiduccia-Mattheyses partitioner MLPart [9]). Bins with seven or fewer cells use an optimal end-case placer. This variety of algorithms facilitates partitioning with small tolerance, allowing Capo to distribute the available whitespace uniformly [13] so as to facilitate routing. This provides a convenient baseline for further wirelength improvement [1] by non-uniform distribution (this configuration is now used by default).

The efficiency of the partitioners and placers implemented in Capo as well as the min-cut placement framework are directly responsible for Capo's speed and scalability. To this end, large-scale partitioning is performed in $O(P \log P)$ time, where P is the number of pins in the hypergraph. The overall run-time spent on middle-range partitioning (FM) scales linearly, and so do cumulative runtimes of all calls to optimal paritioning and placement. Further complexity analysis shows that Capo's asymptotic run-time scales as $O(P \log^2 P)$ on standard-cell designs.

3. RECENT IMPROVEMENTS TO CAPO

Placement Feedback. Terminal propagation relates wirelength optimization in placement to cut optimization in partitioning. Ambiguous terminal propagation arises due to terminals that are nearly equidistant to child bins of a bin being partitioned. The concept of *placement feedback* [14] is a solution to this problem whereby future node locations control present terminal propagation. In feedback a given collection of bins is partitioned N times, without requiring steady improvement, to achieve more consistent terminal propagation. Experiments show consistent improvements for routed wirelength with a smaller run-time penalty than reported in [14].

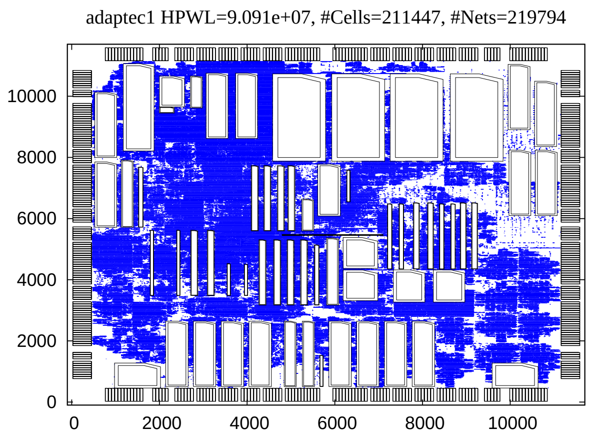
Variable-effort Partitioning. While the effort of partitioning is tuned to the average difficulty, we observed that some partitioning instances appearing in Capo are more difficult. Normally two independent partitioning starts are performed and the better result is V-cycled. We now compare the cuts of the two solutions and perform more partitioning calls if the two cuts are sufficiently far apart. MLPart [9] now repeats V-cycling until no improvement.

Unified Partitioning, Floorplanning and Placement. Min-cut placers scale well in terms of run-time and wirelength minimization, but cannot produce non-overlapping placements of modules with a wide variety of sizes, e.g. mixed-size placement instances. On the other hand, annealing-based floorplanners can handle vastly different module shapes and sizes, but for relatively few (100-200) modules at a time. Solution quality and run-time suffer on larger numbers of modules. Following the work in [3], Capo now applies min-cut placement as much as possible and performs fixed-outline floorplanning using Parquet [4] when necessary.

Since min-cut placement generates a slicing floorplan, it can be viewed as an implicit floorplanning step. Min-cut placement breaks down on bins with modules that are close in size to bin outlines. When such a module appears in a bin, Capo switches from recursive bisection to "local" floorplanning where the fixed outline is determined by the bin. This "correct-by-construction" approach preserves wirelength, congestion and delay estimates, and avoids the need to legalize overlapping macros. Floorplaning determines the locations of large objects, and the remaining small objects are placed by further partitioning. In the case where fixed-outline floorplanning fails to meet area constraints we undo the previous partitioning step and merge the failed bin with its sibling bin, replacing both bins. The merged bin includes all modules contained in the two smaller bins, and its rectangular outline is the union of the two rectangular outlines. This bin is floorplanned, and in the case of failure can be merged with its sibling. After successful floorplanning, the locations of all large modules are returned to the top-down placer, snapped to rows and considered fixed obstacles. Min-cut placement then resumes with a bin that has no large modules in it.

Failure-induced backtracking steps incur some overhead in failed floorplan runs. To avoid floorplanning calls that cannot be successful, Capo uses ad hoc look-ahead [3]. For example, if the two largest blocks in a child-bin do not fit into its outline, one should floorplan immediately. We revised and extended such ad hoc conditions from [3], to the improvement of both runtime and solution quality. When none of these conditions trigger on sufficiently small bins, we invoke pure area-driven floorplanning with Parquet (ignoring wire saves runtime) as another form of look-ahead.

Improvements in Parquet. Parquet 3.1 used in Capo 9.1 implements both sequence pairs and B*-tree. The former typically produces better wirelengths, but the latter packs better. Thus when floorplanning with sequence pairs fails to satisfy a given fixed outline, Parquet switches to B*-trees. Each annealing move in Parquet 3.1 is faster and this allowed extention of the temperature schedule for better solution quality. The sum effect of the mixed-size placement improvements is 3% better results on average on the IBM-MSwPins benchmarks from [3] so that Capo 9.1 is behind FengShui 2.6 by only 2% while not packing to the left or right.



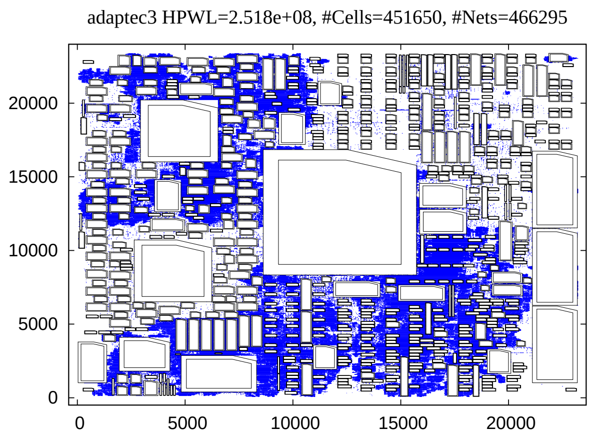


Figure 1: Legal Capo placements of trial benchmarks. Capo places adaptec1 and adaptec3 in approximately 1.5 and 3.5 hours respectively on a 2.4 GHz Xeon workstation. Fixed obstacles are drawn with double lines. To indicate orientation, north-west corners of blocks are truncated.

4. ONGOING WORK

Weighted Terminal Propagation. Recent work by Selvakkumaran and Karypis notes the inaccuracy of representing wirelength objective by cut objective in partitioning driven placement [16]. They developped a new terminal propagation method that allows the partitioner to better map the half-perimeter wirelength cost to min-cut cost using weighted nets. Optimizing the half-perimeter wirelength objective directly through partitioning provides potential improvements over simple min-cut objective.

Advanced Whitespace Management. Capo uses optimal placers when bins contain fewer than a threshold of cells. These "small placers" [11] pack cells, and whitespace is accounted for by adding fake cells with no pins. To model whitespace accurately, one fake cell per site is needed. To limit run-time, Capo uses up to three fake cells of varying sizes to model whitespace. To further improve placements, we are considering techniques for optimal allocation of whitespace without changing relative placement of cells [7, 17]. The scalability of network-flow solvers used in [7, 17] is a concern.

Planning Cutlines Relative to Obstacles. IP blocks, large macros and area array I/Os are increasingly common, and it is necessary to effectively handle such circuit elements. These features can act as boundaries that, in effect, divide a bin into two or more regions. They are also just a few ways in which obstacles can introduce placement bins with non-rectangular fixed outline. Many steps in min-cut floorplacement are frustrated by non-rectangular bins. In particular, floorplanning has trouble satisfying area constraints for non rectangular regions. We propose to plan the geometric placement of cutlines to leverage these observations about obstacles in the placement region, extending [10, Sec. 3.2]. One goal is to place the cutline through objects that act as natural cuts, which already divide the bin into regions. Another promising avenue is to choose cutlines that restore bins to rectangular shapes as soon as possible.

Integration of Analytic Placement. We observe that benchmarks in Figure 1 have many obstacles distributed through the layout. This motivates the use of quadratic placement, and we seek to integrate it into Capo, e.g. to control partition balances [6].

Acknowledgements. We thank the following contributors to Capo and related infrastructure: Sherief Reda from UCSD, Michael Wrighton from CalTech, Dr. Patrick Madden from SUNY Binghamton, and Dr. Xiaojian Yang from Synplicity.

This work was partially supported by the DARPA/MARCO Gigascale Systems Research Center, the National Science Foundation, the Design Automation fellowship, IBM, Synplicity, as well as equipment donations from Intel and IBM.

5. REFERENCES

[1] S. N. Adya, I. L. Markov, P. G. Villarrubia, "On Whitespace and Stability in Mixed-size Placement and Physical Synthesis," *ICCAD* 2003, pp. 311-318.

[2] S. N. Adya et al., "Benchmarking for Large-Scale Placement and Beyond," *IEEE Trans. on CAD* 23(4), pp. 472-488, 2004.

[3] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, I. L. Markov,"Unification of Partitioning, Placement and Floorplanning," *ICCAD*, 2004, pp. 550-557.

[4] S. N. Adya and I. L. Markov, "Combinatorial Techniques for Mixed-size Placement," *ACM Trans. on Design Autom. of Elec. Sys.*, 10(5), Jan 2005, pp. 58-90. Also see *ISPD 2002*, pp. 12-17.

[5] A. Agnihotri et al., "Fractional Cut: Improved recursive bisection placement," *ICCAD*, 2003, pp. 307-310.

[6] C. J. Alpert, G.-J. Nam, P. G. Villarrubia, "Free-Space Management for Cut-Based Placement," *ICCAD* 2002, p. 746.

[7] U. Brenner and J. Vygen, "Faster Optimal Single-Row Placement with Fixed Ordering," *DATE* 2000, pp. 117-121.

[8] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning," *ACM J. on Experimental Algorithms*, vol. 5, 2000.

[9] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning," *ASPDAC* 2000, pp. 661-666.

[10] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" *DAC*'00, p. 477.

[11] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Trans. on CAD* 19(11), pp. 1304-1314, 2000.

[12] A. E. Caldwell, A. B. Kahng, I. L. Markov, "VLSI CAD Bookshelf" `http://vlsicad.eecs.umich.edu/BK`. Also see A. E. Caldwell, A. B. Kahng, I. L. Markov, "Toward CAD-IP Reuse: The MARCO GSRC Bookshelf of Fundamental CAD Algorithms," *IEEE Design and Test*, May 2002, pp. 72-81.

[13] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Hierarchical Whitespace Allocation in Top-down Placement," *IEEE Transactions on CAD* 22(11), Nov, 2003, pp. 716-724.

[14] A. B. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-cut Placement," *DAC* 2004, pp. 357-362.

[15] D. A. Papa, S. N. Adya, I. L. Markov, "Constructive Benchmarking for Placement," *GLSVLSI* 2004, pp. 113-118. `http://vlsicad.eecs.umich.edu/BK/FEATURE/`

[16] N. Selvakkumaran and G. Karypis, "THETO: A Fast and High-Quality Partitioning Driven Global Placer," Technical Report 03-046, 2003, University of Minnesota.

[17] X. Tang, R. Tian, M. D.F. Wong, "Optimal Redistribution of White Space for Wire Length Minimization," *ASPDAC* 2005, p. 412.

[18] X. Yang, B.-K. Choi, M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement," *ISPD* 2002, pp. 42-50.