

Robust Mixed-Size Placement Under Tight White-Space Constraints

Jason Cong, Michail Romesis[†], and Joseph R. Shinnerl
UCLA Computer Science Department {cong,shinnerl}@cs.ucla.edu

[†] Magma Design Automation, Inc., Eindhoven, The Netherlands michalis@magma-da.com^{*}

ABSTRACT

A novel and very simple correct-by-construction top-down methodology for high-utilization mixed-size placement is presented. The POLARBEAR algorithm combines recursive cut-size-driven partitioning with fast and scalable legalization of every placement subproblem generated by every partitioning. The feedback provided by the legalizer at all stages of partitioning improves final placement quality significantly on standard IBM benchmarks and dramatically on low-white-space adaptations of them. Compared to Feng Shui 5.1 and Capo 9.3, POLARBEAR is the only tool that can consistently find high-quality placements for benchmarks with less than 5% white space. With white space at 5%, POLARBEAR beats Capo 9.3 by 10% in average total wirelength while Feng Shui 5.1 frequently fails to find legal placements altogether. With 20% white space, POLARBEAR still beats Capo 9.3 by 1% and Feng Shui 5.1 by 1% in average total wirelength, in comparable run times.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids—*placement and routing*; G.4 [Mathematical Software]: Algorithm Design and Analysis; J.6 [Computer-Aided Engineering]: Computer-Aided Design

Keywords

Mixed-Size Placement, Legalization, Recursive Bipartitioning, White Space, Utilization

1. INTRODUCTION

Advancing IC technology has brought increased use of large intellectual-property (IP) blocks in multi-million-gate ASICs and SOC designs. Most modern designs consist of

^{*}The second author was at UCLA during the period in which this research was completed.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

JCCAD '05 San Jose, California USA

Copyright 2005 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

a very large number of standard cells mixed with many big macros, such as ROMs, RAMs, and IP blocks. The placement of mixed-size cells has thus become a very important topic in physical design. The problem can be defined as follows. Given a fixed rectangular region \mathcal{R} divided into rows of uniform height, arrange a set of rectangular standard cells and large macros in \mathcal{R} such that adjacent cells and macros do not overlap. Standard cells all have a common height equal to the row height. Macros are significantly bigger and often span multiple rows. The objective is usually the minimization of total wirelength, expressed as the sum of the half-perimeters of the bounding boxes of the nets. However, other objectives — routed wirelength, timing — can be used instead, with various constraints — power, temperature, etc.

Compared to standard-cell placement, most of the increased difficulty in mixed-size placement is attributable to overlap removal, or *legalization*. Although in general legalization is NP-complete, legalization of a standard-cell placement is typically easy, because all standard cells have the same height and differ only in their widths. Most placement tools are able to produce legal standard-cell solutions, even when little white space is available, without sacrificing much wirelength. However, when large multi-row blocks are added to the design, placement becomes similar to floorplanning in complexity. In this context, it is often possible that even a good legalization algorithm can fail to find an overlap-free placement which retains the basic structure of a given global placement. Moreover, in designs of high row utilization, i.e., low white space, experiments show that publicly available state-of-the-art software may fail to find a legal solution altogether, even when a given global placement is known to be good in both wirelength and block density distribution.

In practice, both low- and high-utilization designs are important (i.e., high- and low-white-space designs, resp.). The recently released ISPD2005 benchmark suite [20], for instance, consists exclusively of low-utilization test cases. Large designs produced in low or moderate volume often have very high white space in order to support a resynthesis flow in the presence of complex constraints, e.g., tight timing requirements, routability, or design for manufacturability (DFM) restrictions. In this scenario, changes to the netlist and/or the layout may sometimes require significant, unanticipated increases in either the total cell area or in the smallest possible core area containing the placed cells and blocks. In general such changes can only be accommodated if sufficient white space is available; time-to-market pressures preclude vigorous optimization of the die size.

Given the high pressure to reduce IC cost, however, de-

signs with tighter area constraints are equally or even more important in a low-margin consumer market. The fabrication cost of an IC increases rapidly with the area of its layout. In practice, designs with white space between 5 and 10% white space are common; designs with less than 5% white space are not unusual [27, e.g.]. The ability to find high-quality overlap-free mixed-size placements scalably and reliably in a low-white-space setting is clearly important; yet, as discussed below, this ability has yet to be satisfactorily achieved, despite recent progress [2]. Frequent failures by many existing placement tools, including almost all academic tools, on designs with white space below 15% may present a serious limitation on their applicability.

The work presented here demonstrates that mixed-size placement by recursive bipartitioning can be done very successfully in a scalable way that removes any need for post-hoc legalization *or backtracking*. The overall flow may be viewed as an enhanced application to mixed-size placement of the look-ahead legalization paradigm introduced by the PATOMA floorplanner [10] (differences between the flow here and PATOMA are summarized in the next section). A non-overlapping solution is obtained during global placement through the explicit construction of strictly legal layouts for every partition block at every level of the top-down hierarchy. These legal layouts are computed for all placement subproblems as soon as they are generated by min-cut bipartitioning. Targeting area as its primary objective, the legalizer has linear complexity, runs extremely fast, and finds legal solutions with a very high success rate, even with row utilizations over 99%. Its role is best understood not as predictor, but as *guarantor*. Most legal layouts for larger subproblems near the top of the bipartitioning hierarchy are not actually used. However, should legalization fail on a given subproblem, the existence of a known legal layout for its parent subproblem ensures that legality can be restored quickly and locally, without a potentially costly backtracking through multiple generations of ever larger ancestor regions. In this situation, feedback from the legalizer implicitly guides subsequent partitioning, significantly improving the quality of the final placement [13].

Dramatically improved performance on low-white-space designs is achieved by this approach. By scalably incorporating legalization into the hierarchical flow, better partitioning decisions are made during global placement, and placements of superior quality are obtained in extremely fast $\mathcal{O}(N \log N)$ run time. The implementation of our algorithm, POLARBEAR (“Placement by Legalized Recursive Bipartitioning is more Robust”), consistently obtains high-quality placements on standard mixed-size benchmarks, even as the amount of white space is decreased to just 1%. In contrast, Feng Shui 5.1 [17] cannot consistently obtain legal solutions on these benchmarks with less than 15% white space, and Capo 9.3 frequently fails in the range 1–5% white space. All three tools use comparable run time and show similar scalability. Among other leading academic tools — Dragon [23], mFar [12], Aplace [15, 14], and mPL5 [6] — only mPL5 has released publicly a binary for mixed-size placement, and only very recently. As all these tools employ legalization only after global placement has completed, none can provide any formal assurance of successful termination under tight white-space constraints.

The remainder of the paper is organized as follows. Section 2 briefly describes recent related work on mixed-size

placement. Section 3 presents the main elements of the POLARBEAR algorithm, including RBP. Experiments and results are summarized in Section 4. Extensions and conclusions are discussed in Sections 5 and 6.

2. RELATED WORK

Mixed-size placement has recently drawn considerable attention, with several recent papers reporting large improvement. We divide this work into two categories: (i) methods explicitly requiring legalization after global placement, and (ii) methods whose global placements are intended to be overlap-free by construction.

As of August 2005, the best published wirelength results are obtained by methods requiring legalization after global placement. Feng Shui 5.1 [17] uses recursive-bisection with iterative deletion, iterative repartitioning, relaxed rows not aligned with standard cell rows (“fractional cut”), and a simple Tetris-style approach to legalization. A-place [15, 14] employs a multiscale, force-directed formulation and nonlinear conjugate-gradient iterations.

To our knowledge, CPLACE [24] is the first partitioning-based placer to incorporate explicit legalization into every level of the top-down partitioning hierarchy. In CPLACE, this progressive legalization supports accurate modeling of complex constraints such as “irregular images, fixed objects, fixed IOs, large objects, timing-driven placement, and free-space distribution.” But legalization at each level is performed after partitioning without any formal assurance of its success. In contrast to CPLACE, POLARBEAR is more narrowly focused on wirelength minimization under the formal assurance of legal termination.

Most other previously published correct-by-construction algorithms for mixed-size placement rely on simulated annealing in some crucial way. mPG [7] builds a cluster hierarchy for multiscale optimization in a physical-hierarchy-generation framework. mPG uses simulated annealing (SA) on the Sequence-Pair [19] floorplanning representation over nested grids at every level of the cluster hierarchy for legalization. Reliance on SA slows mPG down considerably.

Correctness by construction is a relatively recent addition to Capo [5, 3, 1, 2], which remains very competitive. Capo 9.3 proceeds top down by cutsize-driven recursive bipartitioning until certain ad-hoc tests suggest that newly generated subproblems may be difficult to legalize. At that point, standard cells in each subproblem are clustered, and these clusters are treated as soft macros. SA-based fixed-outline floorplanning is then attempted on the hard macros and soft clusters for the given subregion. If it succeeds, the locations of the macros are then fixed, and further refinement proceeds on the declustered soft macros. If it fails, then the subproblem is merged with its sibling, the previous partition of the parent subproblem is discarded, and floorplanning is attempted for the parent subproblem. In principle, this backtracking may continue indefinitely until some ancestor is successfully floorplanned or until failure at the top level occurs. In practice, the ad-hoc tests used to determine when to commence floorplanning are observed to be good enough that backtracking is only rarely needed. However, when white space is particularly scarce, e.g., less than 4%, Capo 9.3 reports failures, presumably because its ad-hoc tests are insufficient to prevent floorplanning on subproblems that are too large for its SA-based floorplanner to solve scalably. Moreover, clustering standard cells to form

rectangular blocks may prevent a hierarchical method from finding an optimal or near-optimal solution in terms of wirelength and delay minimization [9].

Although POLARBEAR has some superficial similarity to Capo 9.3 [2], its differences are quite significant, as evidenced by its superior performance and robustness. First, POLARBEAR guarantees legality by look-ahead rather than backtracking. Failures in its intermediate-level legalizations are used to provide important feedback to its cutsizes-driven partitioner. Ultimately, this feedback allows its recursive bipartitioning to continue to single-cell end cases. Second, all its core algorithms are scalable and deterministic. Rather than use floorplanning to enforce legality, it uses extremely fast and simple row-oriented block packing (RBP). Because RBP legalization is so fast and scalable, it can be applied to every subproblem at every level, even on the flat problem at the very top level. In this way, backtracking is avoided, and run time is used very consistently and predictably.

Recent advances in fast floorplanning by recursive bipartitioning [22, 10] are most closely related to the work in this paper.¹ In particular, the PATOMA floorplanner [10] also constructs explicit floorplans for each of the floorplanning subproblems generated by recursive bipartitioning. However, POLARBEAR is more than a simple adaptation of PATOMA to handle standard-cell row constraints. Rather, POLARBEAR presents significant refinements and extensions of the feedback between prelegalization and optimization components in PATOMA. As described in Sections 3.3 and 4 below, these enhancements improve POLARBEAR’s quality of results by approximately 15–20%, on average.

3. THE POLAR BEAR ALGORITHM

POLARBEAR employs top-down, cutsizes-driven, recursive bipartitioning in combination with explicit area legalization of every placement subproblem generated by each partitioning. Cut-size driven area bipartitioning is done by hMetis [16]. Subproblem legalization is performed by a simple, scalable heuristic which we call row-oriented block packing (RBP). The main contribution of POLARBEAR is not in either of these components, but in how they are combined in the recursive-bipartitioning flow. The feedback from RBP guides post-partitioning modification of the hMetis solution used to legalize the two generated subproblems. The recursive partitioning process continues until every block is assigned to its own subregion. Placement of blocks in their assigned subregions automatically produces a legal layout. This layout is further improved by greedy swapping of neighboring cells in detailed placement.

3.1 Flow Overview

Pseudocode for POLARBEAR is shown in Figure 1. By an instance, we mean a rectangular placement region \mathcal{R} ; a set of cells and macros \mathcal{V} , each $v \in \mathcal{V}$ of prescribed area and shape; and a hypergraph netlist $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where each net $e \in \mathcal{E}$ is a subset of \mathcal{V} . For brevity, we refer to cells and macros collectively as “blocks.” Initially, a completely legal overlap-free placement to the given instance is computed by RBP without regard to wirelength. Because RBP is scalable and rectangle packing is NP-hard, RBP cannot in general *guarantee* that it will find a legal solution, if one ex-

ists. However, by ignoring wirelength, RBP is typically able to find tightly packed placements quickly, even with total white space below 1%. We have yet to observe an RBP failure at this initial stage on any available benchmark circuit. Henceforth, we assume that this initial application of RBP succeeds; therefore, every subproblem is descended from a parent subproblem for which an *explicit*, legal, overlap-free solution is known. Under this assumption, POLARBEAR is guaranteed to terminate with a legal placement.

Given an RBP placement of the given instance, the algorithm proceeds recursively in the same way on that instance as on every subproblem it subsequently generates. Cells and macros are partitioned by hMetis into two subsets, the ratio ρ of whose total areas satisfying $2/3 \leq \rho \leq 3/2$ (balance factor 10%). The partitioning attempts to minimize the total number of nets connecting blocks in both subsets. Connections to fixed terminals along the placement region or in other subregions are modeled by terminal propagation. The placement region \mathcal{R} is then sliced to create two new placement subproblems, one for each of the two block subsets. Initially, the cutline is placed in proportion to the areas of the block subsets, and its initial orientation minimizes the aspect ratios of the subregions it creates. As described below, its position and orientation may subsequently be changed.

The key feature distinguishing POLARBEAR from other min-cut-based placement algorithms is its incorporation of legalization into its top-down flow. Before recurring with hMetis on the two new subproblems, POLARBEAR first calls RBP in order to construct legal, overlap-free placements of those two subproblems. Although the initial application of RBP may fail to legalize one or both of them, one of four separate correction strategies described in Section 3.3 below is guaranteed to succeed, given the legal layout of the parent subproblem. These explicitly constructed legal subproblem solutions then allow recursive, legalized, cutsizes-driven bipartitioning to continue separately on each of them. The algorithm continues in this fashion down to end cases of subregions containing one block each.

3.2 Prelegalization by Block Packing (RBP)

Because legalization in POLARBEAR is always its *first* step for computing a placement, it is also called “prelegalization.” The purpose of prelegalization is to find a legal, non-overlapping configuration of a given set of blocks in a given region as quickly as possible. This legal configuration is used to guide actual placements only if one of its child subproblems defined by cutsizes-driven partitioning cannot also be legalized. Otherwise, once legal placements are obtained for both its child placements, the prelegalized solution is discarded. For speed, simplicity, and modularity, wirelength is ignored during prelegalization.² In order that the prelegalizer may serve as guarantor of the legalizability of subsequent placement subproblems contained within its subregion, each of its solutions is required to contain at least one straight-line *slice*, either horizontal or vertical, which can be used as a cutline, if necessary. Wirelength-driven refinement of an RBP placement is described in Section 3.3 below.

Prelegalization in POLARBEAR is an extremely simple form of row-oriented block packing (RBP). Macros and cells are

¹Other recent work has also drawn attention to the connection between mixed-size placement and floorplanning [2].

²Though some wirelength gain might be obtained by incorporating network flows into prelegalization, such techniques were not attempted in the current implementation.

PolarBear Mixed-Size Placement

input: Set of hard blocks $\mathcal{V} = \{v_1, \dots, v_n\}$; netlist $\mathcal{H} = (\mathcal{V}, \mathcal{E})$; rectangular region \mathcal{R} of fixed dimensions.

remark: Each node of the bipartitioning tree is a triple: (i) a set of blocks V , (ii) a rectangular subregion R , and (iii) a legalized placement $P(V, R)$ of V in R .

Apply RBP to \mathcal{V} in \mathcal{R} .

if (RBP fails to prelegalize \mathcal{V} in \mathcal{R})
 Report a failure of POLARBEAR to the caller and **exit**.

else
 Denote RBP’s legal placement of \mathcal{V} in \mathcal{R} by \mathcal{P} .
 Set the root node to $(\mathcal{V}, \mathcal{R}, \mathcal{P})$.

end if

Create a queue Q of prelegalized placement subproblems.
enqueue the root node $(\mathcal{V}, \mathcal{R}, \mathcal{P})$ in Q .

while (Q is nonempty) **do**
 dequeue a prelegalized subproblem $S = (V, R, P)$.
 Partition V into disjoint subsets V_1, V_2 by hMetis with terminal propagation. Slice R into subregions R_1, R_2 , and assign V_1, V_2 to them.
 Let $P_1 := \text{RBP}(V_1, R_1)$ and $P_2 := \text{RBP}(V_2, R_2)$.
 notation: P_i is **true** if and only if P_i is legal.
 if (**not** (P_1 and P_2))
 if (cutline search legalizes P_1 and P_2)
 continue
 else if (repartitioning legalizes P_1 and P_2)
 continue
 else if (block swapping legalizes P_1 and P_2)
 continue
 else refine $P = \text{RBP}(V, R)$ to reconstruct legal P_1 and P_2 .
 end if
 remark. P_1 and P_2 are now legal.
 if ($|V_1| > 1$) **enqueue** (V_1, R_1, P_1) in Q .
 if ($|V_2| > 1$) **enqueue** (V_2, R_2, P_2) in Q .
end do

output: a legal placement of \mathcal{V} inside region \mathcal{R}

Figure 1: Overview of the PolarBear algorithm.

taken in nonincreasing-height order and placed in consecutive rows in the subregion. Each block is placed in the first row in which it fits in a way that preserves at least one slice. Individual rows are filled from left to right. Macros typically span multiple rows. Therefore, stacks of smaller blocks may appear to the right of larger blocks (Figure 2). The top edge of a block is not allowed higher than the top edge of its left neighbor. If at any point, a macro or a standard cell cannot fit in the specified region, the algorithm reports failure. The row-oriented structure ensures that either (i) a horizontal slice along a row boundary exists; or, (ii) the tallest macro spans all rows, creating a vertical slice. A small sample RBP layout is shown in Figure 2 below.

3.3 RBP Feedback to Bipartitioning

If RBP initially fails to find a legal solution to a given subproblem, four separate corrective measures are attempted in sequence. The first three measures — cutline repositioning, repartitioning, and iterated block swapping — are not guaranteed to legalize. When they succeed, however, they

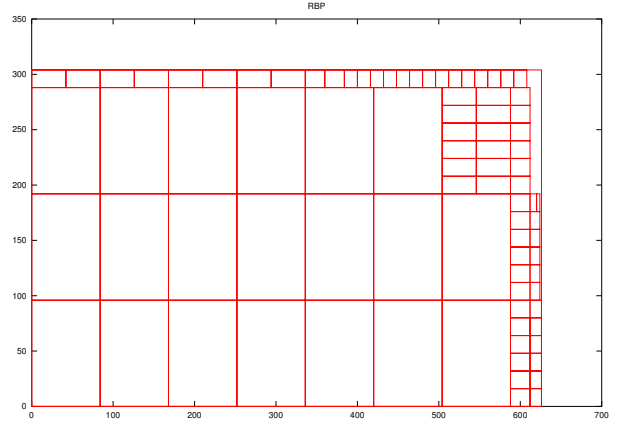


Figure 2: Sample RBP Layout

preserve a given cutsizes-driven partitioning as closely as possible. If all three fail, then cutsizes-driven partitioning of the parent subproblem is abandoned, and the prelegal RBP solution to the parent subproblem is instead adopted and refined. Overall, these improved feedback measures reduce POLARBEAR’s average total wirelength by 15–20%, on average.

Cutline Search. When RBP finds a legal solution to one of the subproblems but not the other, the cutline can be moved away from the failed case and toward the solved one. A limited number of iterations (3–12) of binary search on the cutline position is performed. The block subsets of the subregions are held fixed, and for each candidate cutline position, RBP is attempted anew on the same block subsets in the new candidate subregions.

Repartitioning. If one of the placement subproblems still cannot be solved after cutline search, the entire process is repeated for up to 10 new hMetis partitionings or until legality is obtained for both subproblems. Experiments to date produce the best quality/run-time tradeoff with 2 runs of hMetis for each of 5 different balance factors: 10, 15, 5, 20, and 25%. Overall, replacing these multiple runs of hMetis by just one run at balance factor 10% increases total wirelength by 9%.

Iterated Block Swapping. When repartitioning and cutline search reach their limits, the first hMetis solution with 10% balance factor is restored for attempted correction by iterative refinement. Suppose that RBP successfully finds a legal placement for subregion \mathcal{A} but not for its sibling subregion, \mathcal{B} . Usually, a small number of small adjustments to the given cutsizes-driven partitioning suffices to determine legal solutions to both subproblems. A partial solution of \mathcal{B} is generated by running RBP while skipping the placement of the blocks that do not fit in the subregion. The legal solution to \mathcal{A} and the partial solution to \mathcal{B} are used as a starting point. First, the blocks not contained in \mathcal{B} by its partial solution are moved across the cutline from \mathcal{B} to \mathcal{A} . This step legalizes the placement in \mathcal{B} but typically renders the solution to \mathcal{A} illegal. In order to re-legalize \mathcal{A} , the cutline is first moved as far toward \mathcal{B} as possible, so that the width of \mathcal{B} is the same as the width of the widest row of blocks there. Then RBP is rerun on the new subproblem for \mathcal{A} . If RBP fails on this new subproblem, then the above steps are repeated with the roles of regions \mathcal{A} and \mathcal{B} reversed. This refinement

continues up to a maximum of 10 iterations until either (i) legal placements to both subregions are found, or (ii) cycling occurs; i.e., a given set of leftover blocks appears more than once for different iterations of the same subproblem. When a legal target layout is found, there are usually multiple blocks of the same dimensions which can be relocated in order to obtain the legal layout from the original. The blocks actually moved are selected to reduce wirelength, as estimated by placing all pins at subregion centers.

Experiments demonstrate that iterated block swapping is the most effective of the correction heuristics used in POLARBEAR. When it is omitted, average total wirelength increases by 14%, while run time decreases by only 3%.

Refining an RBP Solution. If iterated block swapping fails to legalize a given placement subproblem, then POLARBEAR returns to its parent subproblem, for which a legal RBP solution has already been computed and stored. A non-legalized *target* solution to this subproblem is then computed by traditional min-cut placement: recursive cutsizes-driven bipartitioning coupled with terminal propagation, cutline specification, and assignment of the block subsets to the subregions defined by the cutline position. Locations of blocks in this target solution are used to guide the refinement of the given RBP solution, as follows. Blocks of identical dimensions in the RBP solution are permuted in order to move them as close to their locations in the target solution as possible. I.e., the original RBP solution is viewed as a template for the ultimate assignment of its blocks to the subregions currently associated with the blocks.

In POLARBEAR, the permutation is generated by sorting the block locations in the RBP solution by their y -coordinates, if a partition along the x -dimension will follow, or by their x -coordinates, if the partition will be along the y -dimension. The target locations are sorted in the same fashion. Juxtaposing these orderings gives the assignment.

The permuted RBP placement is bipartitioned, and the main algorithm resumes separately on each of its two child subproblems. In order to guarantee the legality of subproblem solutions, the permuted RBP placement is partitioned along one of its row or column boundaries, and not by generic, cutsizes-driven hMetis bipartitioning.³ A few nearly centered, row or column-separating cutlines for the RBP solution and its symmetric solution (flipped across the cutline) are considered for its bipartitioning. For each of these candidates, wirelength is estimated by placing all blocks in each subregion at the subregion’s center and modeling external connections by terminal propagation. The selected cutline produces the least estimated wirelength. An example of RBP refinement is shown in Figure 3. On average, this refinement of the guarantor RBP solution reduces final, total wirelength by 3%.

4. EXPERIMENTS AND RESULTS

The POLARBEAR algorithm was implemented with the gcc 3.2.3 compiler on a 2.4 GHz Pentium 4 processor in a Red-Hat 9.0 Linux environment. A sample POLARBEAR solution is illustrated in Figure 4. It was compared with two leading mixed-size placement algorithms publicly available online: Feng Shui 5.1 [28] and Capo 9.3 [25]. Both these tools use

³In our experiments, the vast majority of the cases where POLARBEAR has to resort to this step were observed to be very small in the number of blocks — always less than 30.

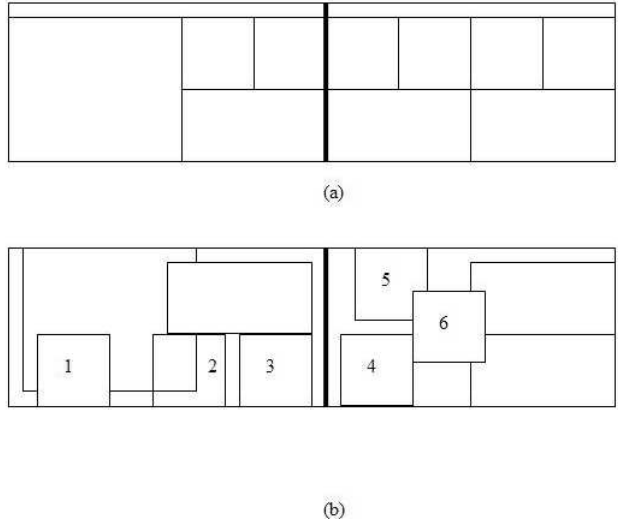


Figure 3: Refining the RBP Layout. The nonlegalized “ideal” min-cut placement (b) is used to guide permutations of identically sized blocks 1 – 6 in the legal RBP layout (a).

recursive bipartitioning, but their methodologies are different (cf. Section 2). Feng Shui is very aggressive during global placement; it shows relatively little consideration for nonoverlapping constraints. After global placement, it uses a simple Tetris-like legalization algorithm [11, 18] to remove overlap. However, this combination consistently fails to produce legal placements on the ICCAD04 benchmarks when white space is decreased below 10%. Capo 9.3 uses backtracking and SA-based floorplanning to construct correct layouts without post-hoc legalization. However, as white space decreases to near 3%, it often reports failures also, presumably because its backtracking proceeds to subproblems too large for its floorplanner to handle with acceptable run time, so it resorts to a macro legalization procedure that is not guaranteed to work in all cases.

4.1 Comparison on ICCAD04 Benchmarks

POLARBEAR was run on the IBM/ICCAD 2004 benchmarks for mixed-size placement [2] with the default 20% white space. The results are shown in Table 1 below. On average over these examples, Capo 9.3’s wirelengths are 1.0% longer than POLARBEAR’s, and FengShui 5.1’s wirelengths are 0.8% longer than POLARBEAR’s.

4.2 Results under Parametrized White Space

The results for the same ICCAD 2004 benchmarks [2] for mixed-size placement with parametrized white-space percentages are reported in Figure 5. Twenty different versions of the benchmarks were generated by setting the white space available in the placement region from 1% up to 20% white space in increments of 1%. POLARBEAR is clearly much more robust than Feng Shui 5.1 and Capo 9.3. It successfully computed a legal placement for every benchmark tested, with every value of white space, down to 1% white space.⁴ Solutions produced by Feng Shui 5.1 are consistently

⁴It seems evident that legal placements can also be computed under much less than 1% white space, at some cost in increased wirelength.

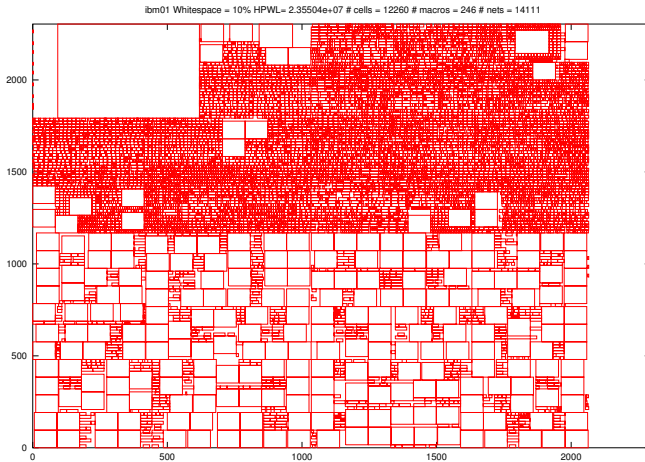


Figure 4: The solution of PolarBear to the IBM01 benchmark.

legal over all the benchmarks only with white space at least 15%. Solutions produced by Capo 9.3 are consistently legal over all the benchmarks only with white space at least 5%. Capo 9.3 typically does find legal solutions when white space is as low as 1%, but not consistently. Table 1 reports for each benchmark both the least value of white space for which a legal solution was found and the greatest value of white space for which a failure was observed.

With 20% white space, POLARBEAR’s wirelength is on average 0.8% shorter than Feng Shui 5.1’s and 1.0% shorter than Capo 9.3’s. On IBM02, for example, POLARBEAR beats Feng Shui by 25% on average over the different white space values where Feng Shui succeeds. In runtime, POLARBEAR is slightly slower than Feng Shui (1.3 \times) and twice as fast as Capo.⁵ Overall, the results suggest that, compared to existing methods, enforcing legalization during min-cut, mixed-size placement leads both to much improved robustness and better placement quality.

4.3 Results for the FARADAY Benchmarks

The results for the FARADAY benchmarks [2] are reported in Table 1. The table reports the performance of PolarBear, Capo and FengShui in both wirelength and runtime. For these benchmarks, we noticed that the default parameters of PolarBear do not produce good results for the RISC1 circuit. We observed that by changing the number of allowed repartitionings with different balance factors from 5 (the default value, cf. Section 3.3) to 49 (from 1% to 49%), the performance of PolarBear was much better, but with a large increase in runtime (2.5 \times). We call this version of PolarBear “BigBear.” It is more suitable for benchmarks with large macros and low white space. We believe the additional repartitionings are needed by POLARBEAR in these test cases to overcome two limitations of POLARBEAR’s current implementation: (i) an inability to handle fixed blocks, and (ii) lack of any special handling of large macros.

⁵This possibility was not investigated.

⁵Capo provides options for faster runtime or better quality. We used the default values.

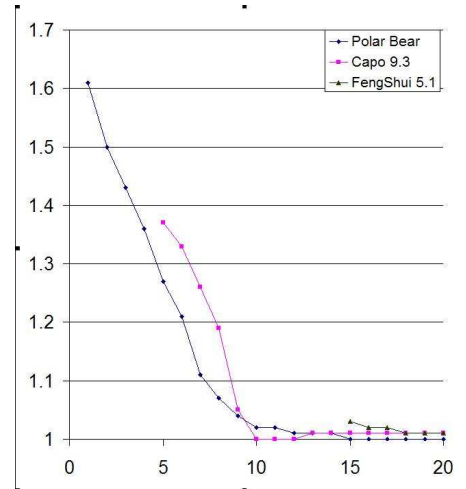


Figure 5: The performance of PolarBear, Feng Shui 5.1 and Capo 9.3 under limited white space. Half-perimeter wirelengths are normalized to PolarBear’s results for 20% white space. For each white space value, results for each tool are shown here only when that tool finds a legal placement for every benchmark in the suite.

Table 1 reports the results of both POLARBEAR and BigBear as well as Capo 9.3 and FengShui 5.1 on the FARADAY test cases. From the results, we see that Capo’s quality is between that of BigBear (4.9% worse) and PolarBear (8.2% better). FengShui produced legal results for only one of the five benchmarks.

4.4 Quantification of Enhancements

We evaluated the effect of three important steps of our algorithm separately from each other by removing them from the flow and comparing the results to the original flow. The first heuristic that we evaluated was the application of multiple runs of hMetis to find a feasible solution for both partitions before moving to the post-partitioning process. We replaced that by one run only of hMetis. On average the wirelength worsens by 9%, but the runtime improves by 32%. The second heuristic was the iterative block swapping which is applied as a post-partitioning process to modify the hMetis solutions as little as possible while ensuring feasibility of both subregions. This was the most efficient heuristic of our approach, since by removing it the wirelength worsens by 14%, while the runtime improves by 3% only. The third heuristic we evaluated is the RBP solution refinement which is applied only if all the other heuristics have failed, by imposing the direct non-wirelength optimized RBP layout in the parent subregion of the failed problem. By removing it the wirelength worsened by 3%, while the runtime improves by 6%. We observed that in the latter case the detailed placement algorithm that we apply is very helpful and is able to compensate for the lack of the RBP layout refinement. This behavior can be explained by the small sizes of the subproblems where this heuristic has to be applied (as mentioned in the previous section). However, we noticed that when running the program for little white space, these regions become larger, and the RBP layout refinement heuristic is more

Circuit	PB HPWL	PB CPU	FS HPWL	FS CPU	FS Best	FS Worst	Capo HPWL	Capo CPU	Capo Best	Capo Worst
ibm01	2.34	6	1.05	0.33	9%	8%	1.04	1.00	5%	4%
ibm02	5.04	5	1.25	0.80	5%	10%	1.02	2.00	4%	3%
ibm03	7.90	5	1.06	1.00	15%	14%	0.96	2.60	5%	4%
ibm04	8.71	12	1.02	0.42	7%	10%	0.93	1.17	4%	3%
ibm05	10.70	6	0.92	1.00	1%	-	0.97	2.00	1%	-
ibm06	6.99	9	0.96	0.78	14%	13%	0.97	2.00	4%	3%
ibm07	12.00	11	0.93	0.90	7%	6%	1.13	3.55	3%	2%
ibm08	14.34	15	0.94	0.73	10%	9%	0.96	1.87	2%	1%
ibm09	13.69	12	1.07	1.00	13%	12%	1.08	2.50	1%	-
ibm10	31.47	23	1.09	0.74	8%	7%	1.00	2.22	5%	4%
ibm11	19.89	20	1.04	0.75	12%	11%	1.09	2.15	1%	-
ibm12	38.37	31	1.03	0.58	13%	12%	1.00	1.87	5%	4%
ibm13	24.65	30	1.04	0.67	10%	9%	1.07	1.90	4%	3%
ibm14	39.53	47	0.97	0.81	4%	5%	1.00	2.17	5%	4%
ibm15	53.38	160	0.98	0.29	11%	10%	1.03	0.77	1%	-
ibm16	63.34	75	0.95	0.69	8%	7%	0.95	1.96	2%	1%
ibm17	76.27	69	0.91	0.83	4%	3%	0.96	2.30	3%	2%
ibm18	48.13	65	0.93	0.92	3%	2%	0.98	2.14	1%	-
Avg.	1	1	1.01	0.74			1.01	2.01		

Table 1: Comparison of PolarBear (PB) with Feng Shui 5.1 (FS) and Capo 9.3. Reported runtime for PolarBear is in minutes. HPWL values refer to the 20% white-space test cases only; wirelengths for Feng Shui and Capo are normalized to PolarBear’s results. The columns labeled “Worst” report the greatest white space percentages for which Feng Shui and Capo reported at least one failure. The columns labeled “Best” report the least white space percentages for which Feng Shui and Capo reported at least one success.

Circuit	PB HPWL	PB CPU	BB HPWL/PB	BB CPU /PB	Capo HPWL/PB	Capo CPU /PB	FS HPWL/PB	FS CPU /PB
DMA	4.57E+08	7 min	0.99	2.71	0.98	0.29	-	-
DSP1	10.21E+08	27 min	0.95	1.22	1.04	0.33	-	-
DSP2	9.58E+08	14 min	0.96	1.79	0.93	0.64	0.98	0.57
RISC1	33.73E+08	31 min	0.44	5.18	0.49	0.58	-	-
RISC2	14.35E+08	28 min	1.00	2.03	1.14	0.61	-	-
Avg.			0.869	2.587	0.918	0.490	-	-

Table 2: Half-perimeter wirelength (HPWL) comparison of PolarBear (PB) with Big Bear (BB), Feng Shui 5.1 (FS) and Capo 9.3 on the Faraday benchmarks. Reported runtime for PolarBear is in minutes. HPWL values for BB, Feng Shui and Capo are normalized to PolarBear’s results.

helpful there.

POLARBEAR uses mPL’s detailed placement (DP) algorithm only to improve the wirelength of its result and not for legalization. We also evaluated the impact of this use of DP on wirelength. On average, mPL’s DP algorithm improves POLARBEAR’s half-perimeter wirelength by 7%, while it takes 18% of the total placement time. Table 3 summarizes these results.

5. EXTENSIONS

In practice, a useful placement engine must generally satisfy a variety of complex constraints, e.g., constraints on timing, temperature, manufacturability, and routability. In the standard-cell context it has been argued that routability is a principal challenge for placement under tight white-space constraints [30]. Whether the prelegalization framework proposed here and in Patoma [10] can successfully be extended to other constraints in addition to non-overlap is an open question. In the case of routability, for instance, there is no apparent way to obtain a certificate of routability for a given placement that might be used as guarantor in a top-down framework. Because POLARBEAR can trade white space for wirelength in a consistent and robust way, it seems well suited to the popular cell-inflation technique [21, 4, 30], in which cells and macros located in regions of high congestion are artificially enlarged in order that subsequent overlap removal can also serve the goal of congestion reduction. But even if this ad-hoc approach should prove effective,

it would not by itself provide any *formal* assurance of routability.

6. CONCLUSION

Integrating legalization with global min-cut placement produces solutions of superior quality, speed, and robustness on high-utilization benchmarks. The approach is scalable and suitable for the incorporation of complex constraints. It is also adaptable to combinations with other techniques, e.g., recent advances in analytical placement [8, 29, 6] can be used to guide partitioning in place of cutsizes minimization.

Although low-utilization designs are important in practice, the incorporation of ever-increasing amounts of white space as a means of coping with increasing design complexity may be a losing proposition in the long run. As the complexity and heterogeneity of modern designs continue to increase, a more robust “correct by construction” approach may hold a distinct cost advantage. The work here is intended as a “proof-of-concept” demonstration of prelegalization in the restricted context of high-utilization wirelength-driven mixed-size placement. The promising results obtained by our prototype implementation, in comparison with those of established recursive-bisection-based tools, suggest that this advantage may extend well to more complicated design constraints, on further development.

7. ACKNOWLEDGMENTS

Circuit	MhMetis HPWL	MhMetis CPU	BLSwap HPWL	BLSwap CPU	RBP Ref HPWL	RBP Ref CPU	NoDP HPWL	NoDP CPU
ibm01	1.06	0.72	1.09	0.98	1.02	0.96	1.09	0.83
ibm02	1.05	0.75	1.12	0.99	1.02	0.97	1.08	0.60
ibm03	1.07	0.78	1.08	0.98	1.01	0.94	1.08	0.80
ibm04	1.06	0.74	1.10	0.97	1.03	0.97	1.10	0.83
ibm05	1.02	0.87	1.02	1.00	1.00	0.98	1.05	0.67
ibm06	1.08	0.72	1.12	0.96	1.03	0.95	1.06	0.89
ibm07	1.08	0.71	1.11	0.98	1.01	0.94	1.07	0.82
ibm08	1.10	0.68	1.09	0.97	1.00	0.91	1.06	0.80
ibm09	1.09	0.66	1.14	0.98	1.02	0.94	1.09	0.83
ibm10	1.18	0.59	1.22	0.95	1.05	0.89	1.06	0.87
ibm11	1.08	0.65	1.17	0.98	1.04	0.96	1.09	0.85
ibm12	1.10	0.64	1.16	0.97	1.04	0.95	1.06	0.94
ibm13	1.12	0.67	1.19	0.98	1.02	0.91	1.09	0.87
ibm14	1.09	0.66	1.19	0.98	1.03	0.94	1.06	0.79
ibm15	1.14	0.56	1.20	0.96	1.04	0.91	1.02	0.96
ibm16	1.10	0.61	1.14	0.97	1.03	0.94	1.08	0.80
ibm17	1.08	0.62	1.15	0.97	1.03	0.95	1.06	0.83
ibm18	1.11	0.62	1.19	0.97	1.03	0.93	1.07	0.74
Avg.	1.09	0.68	1.14	0.97	1.03	0.94	1.07	0.82

Table 3: The effect of three heuristics and detailed placement on the performance of PolarBear. The first heuristic is the multiple applications of hMetis, the second the iterative block swapping and the third the RBP layout refinement. The table shows the performance of PolarBear when each heuristic is skipped. It also shows the effect of detailed placement (the NoDP columns). The results in terms of wirelength and runtime are normalized to the results of the original algorithm.

Financial Support from Semiconductor Research Consortium Contract 2003-TJ-1091 and National Science Foundation Grant CCF-0430077 acknowledged. The authors would like to thank Min Xie for providing them with the detailed placer of mPL. They would also like to thank Prof. Igor Markov and Prof. Patrick Madden for providing the latest binaries of Capo and FengShui.

8. REFERENCES

- [1] S. Adya, I. Markov, and P. Villarrubia. On whitespace and stability in mixed-size placement and physical synthesis. In *Proc. Int'l Conf. on Computer-Aided Design*, 2003.
- [2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, placement, and floorplanning. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 12–17, Nov. 2004.
- [3] S. N. Adya and I. L. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proc. Int'l Symp. on Phys. Design*, pages 12–17, April 2002.
- [4] U. Brenner and A. Rohe. An effective congestion-driven placement framework. *TCAD*, 22(4):387–394, April 2003.
- [5] A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection produce routable placements? In *Proc. 37th IEEE/ACM Design Automation Conf.*, pages 477–482, 2000.
- [6] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. Int'l Symp. on Phys. Design*, pages 185–192, 2005.
- [7] C.-C. Chang, J. Cong, and X. Yuan. Multilevel placement for large-scale mixed-size ic designs. In *Proc. Asia South Pacific Design Automation Conference*, pages 325–330, 2003.
- [8] C. Chu and N. Viswanathan. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. In *Proc. Int'l Symp. on Phys. Design*, pages 26–33, April 2004.
- [9] J. Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528, 2001.
- [10] J. Cong, M. Romesis, and J. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Asia South Pacific Design Automation Conf.*, 2005.
- [11] D. Hill. Method and system for high speed detailed placement of cells within an integrated circuit design. In *US Patent 6370673*, Apr 2002.
- [12] B. Hu and M. Marek-Sadowska. Fine granularity clustering based placement. *TCAD*, Apr. 2004.
- [13] A. Kahng and S. Reda. Placement feedback: A concept and method for better min-cut placements. In *Proc. Design Automation Conf.*, pages 357–362, June 2004.
- [14] A. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 565–572, 2004.
- [15] A. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. In *Proc. Int'l Symp. on Phys. Design*, pages 18–25, 2004.
- [16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526–529, 1997.
- [17] A. Khatkhate, C. Li, A. R. Agnihotri, S. Ono, M. C. Yildiz, C.-K. Koh, and P. H. Madden. Recursive bisection based mixed block placement. In *Proc. Int'l Symp. on Phys. Design*, 2004.
- [18] C. Li and C.-K. Koh. On improving recursive bipartitioning-based placement. Technical Report TR-ECE 03-14, Purdue University, 2003.
- [19] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing-based module placement. In *Proc. International Conference on Computer-Aided Design*, pages 472–479, 1995.
- [20] G. Nam, C. J. Alpert, P. Villarubia, B. Winter, and M. Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proc. Int'l Symp. on Phys. Design*, pages 216–220, 2005.
- [21] P. N. Parakh, R. B. Brown, and K. A. Sakallah. Congestion driven quadratic placement. In *Proc. Design Automation Conf.*, pages 275–278, 1998.
- [22] A. Ranjan, K. Bazargan, S. Ogrenci, and M. Sarrafzadeh. Fast floorplanning for effective prediction and construction. In *IEEE Trans. on VLSI Sys.*, pages 341 – 351, 2001.
- [23] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern Placement Techniques*. Kluwer, Boston, 2002.
- [24] P. Villarrubia, G. Nusbaum, R. Masleid, and E. Patel. IBM RISC chip design methodology. In *ICCD*, pages 143–147, 1989.
- [25] <http://vlsicad.eecs.umich.edu/bk/pdtools/tar.gz/>.
- [26] <http://er.cs.ucla.edu/Dragon/>
- [27] <http://www.faraday-tech.com/structuredasic/download.html>.
- [28] <http://vlsicad.cs.binghamton.edu/software.html>
- [29] Z. Xiu, J. Ma, S. Fowler, and R. Rutenbar. Large-scale placement by grid warping. In *Proc. Design Automation Conf.*, pages 351–356, June 2004.
- [30] X. Yang, B. Choi, and M. Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *TCAD*, 22(4):410–419, April 2003.