Fast Floorplanning by Look-Ahead Enabled Recursive Bipartitioning

Jason Cong, Michail Romesis, and Joseph R. Shinnerl

Computer Science Department

University of California, Los Angeles

Los Angeles, CA 90095

{cong,michail,shinnerl}@cs.ucla.edu

Abstract

A new paradigm is introduced for floorplanning any combination of fixed-shape and variable-shape blocks under tight fixed-outline area constraints and a wirelength objective. Dramatic improvement over traditional floorplanning methods is achieved by explicit construction of strictly legal layouts for every partition block at every level of a cutsize-driven, top-down hierarchy. By scalably incorporating legalization into the hierarchical flow, post-hoc legalization is successfully eliminated. For large floorplanning benchmarks, an implementation, called PATOMA, generates solutions with half the wirelength of state-of-the-art floorplanners in orders of magnitude less run time. Experiments on standard GSRC industry benchmarks compare an implementation, called PATOMA, to the Traffic floorplanner and to both the default and high-effort modes of the Parquet-2 floorplanner. With all blocks hard, PATOMA's average wirelength is 38% shorter than Traffic's in the same run time. With all blocks soft, PATOMA on average produces wirelengths 16% shorter than Parquet-2's default mode and runs 37× faster. Compared to the high-effort mode of Parquet-2, PATOMA's average wirelength is 8% shorter, and it runs 824× faster, on average.

I. Introduction

Given a collection of interconnected variable-dimension "soft" rectangular blocks and fixed-dimension "hard" rectangular blocks, each block with its own prescribed fixed area, the floorplanning problem is to shape the soft blocks and arrange them together with the hard blocks in the plane without overlap. The objective is typically to minimize (a) the estimated total wirelength of the circuit; (b) its estimated timing performance; (c) the area of the smallest rectangle circumscribing the blocks;

or, usually, (d) some combination of these. The most useful algorithms target wirelength minimization under a fixed outline constraint, the aspect ratios of soft blocks constrained by simple upper and lower bounds.

Fast floorplanning is critical in the hierarchical physical design of VLSI circuits, for two reasons. First, system designers require a means of rapidly estimating the variation in performance of alternative architectures and logic designs. Second, multiscale [8], [24], [15] and mixed-size [26], [2], [17] placement algorithms typically solve some form of floorplanning problem at the coarsest level of approximation, in order to generate an initial coarse placement for subsequent iterative refinement. With the reuse of IP blocks for multi-million-gate ASICs and SOC designs, most modern IC designs consist of a very large number of standard cells mixed with many big macros, such as ROMs, RAMs and IP blocks. When clusters of standard cells are placed simultaneously with macros, the clusters may be treated as soft blocks.

Many floorplanning algorithms have been developed in recent years, varying mostly in the representation of geometric relationships among modules. They can be divided into two major categories: slicing and general algorithms. The first slicing algorithms were developed in the 1980's (e.g. [21], [31]). In the 1990's, general algorithms that do not require slicing cuts became more popular, especially after the introduction of the BSG [20] and Sequence Pair [19] representations. Other non-slicing representations include TCG [18], B*-tree [9], CBL [13], O-tree [11], and so on. Simulated annealing (SA) has been used to minimize area and/or wirelength under each of these representations.

Until a few years ago, the inherent slowness of SA was partially hidden by the lack of any need to floorplan more than 100 blocks at a time. Recently, however, growing numbers of IP blocks have increased the sizes of most floorplanning instances, prompting researchers to seek non-stochastic approaches. Ranjan et al. [23] proposed a two-stage fast floorplanning algorithm. In the first stage, a hierarchy is generated by top-down recursive bipartitioning. Cutline orientations are selected from the bottom up in a way that keeps subregion aspect ratios close to one. In the second stage, low-temperature SA improves wirelength by reshaping blocks to produce a more compact layout. Final, total wirelength was comparable to or better than that obtained by an SA-based algorithm [31], with speed-up of over 1000$\times$ in predictor mode (high-speed) and 20$\times$ in constructor mode (high-effort). More recently, a fast algorithm called Traffic [25] has been used to generate high-quality floorplans without simulated annealing. Traffic also uses two stages. In the first stage, the blocks are divided into layers by linear multi-way partitioning. In the second stage, every layer is optimized individually; the blocks in each layer are separately arranged into rows and then moved among the rows to balance row widths and reduce wirelength. In the end, pairs of rows are squeezed tightly after being transformed into trapezoids. This final step leads to very compact floorplans, but it also increases wirelength, because the cells are ordered according to their heights.

The impressive speedups obtained by the last two algorithms raise the question of whether a fast deterministic approach can be used to replace the widely used SA engine with the same or better solution quality. Fast, recursive, cutsize-driven, area bisection by multilevel netlist partitioning is very successfully and widely used in large-scale placement [6], [29], [7], [17]. To date, however, the quality of bisection-based floorplanners has generally fallen short of that of the best SA-based implementations. We believe that this deficiency has been caused largely by a thorny *legalization* problem [14] that traditionally accompanies the recursive bipartitioning paradigm. Legalization is the process of transforming the end result of recursive bipartitioning, or any so called *global* floorplan, which may still contain overlap and/or other constraint violations, to a configuration in which all shape, non-overlap, and floorplanning boundary constraints are strictly satisfied. As commonly practiced, floorplanning by recursive bipartitioning makes no guarantee that the blocks assigned to a subregion can actually be shaped and arranged there without overlap. In this scenario, defining base cases may be difficult, as many base cases may fail to have legal solutions. Local refinement usually suffices to legalize, but in general some form of global legalization becomes essential to guarantee proper termination. Recently reported progress in the legalization of standard-cell [5] and mixed-size [17] placements depends on a significant amount of available white space (20% or more). When area constraints are tight, legalization usually increases wirelength significantly. The work presented here is the first to define a floorplanning flow driven by recursive cutsize-driven bisection in which the satisfiability of all constraints is explicitly enforced at every step, so that the need for post-hoc legalization is completely removed. *Legal*, a.k.a. *feasible* solutions, strictly satisfying all non-overlapping, area, and shape constraints, are explicitly constructed for every subproblem at each intermediate level. Their feedback to the recursive bisection enables it to proceed significantly longer, deepening the partitioning hierarchy and thereby improving wirelength quality. Our implementation of this flow beats a leading SA-based engine on the GSRC benchmarks by 10–20% in average wirelength and by orders of magnitude in run time.

The only similar work to our look-ahead enabled recursive bipartitioning flow is found in Capo 9.0 [3] for mixed-size placement. Capo 9.0 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. The difference in our approach is that backtracking is never needed, since the look-ahead oracles guarantee the existence of a feasible solution at the parent level. At the same time, Capo still needs a simulated-annealing engine during this process, while PATOMA is totally deterministic.

The paper is organized as follows. Section II gives an overview of our implementation, called PATOMA.[1] Section III describes a zero-dead-space floorplanning algorithm for soft blocks (ZDS) and proves its properties. Section IV presents the adaptation of ZDS to wirelength minimization in PATOMA. Section V describes the ROB (Row-Oriented Block Packing) heuristic for floorplanning a combination of hard and soft blocks. Section VI compares PATOMA's performance with that of Parquet-2 [1] and other non-annealing-based floorplanners. The paper is concluded in Section VII.

II. Overview of the PATOMA Algorithm

PATOMA attempts to minimize total wirelength under a fixed-outline area constraint. It couples top-down, cutsize-driven, recursive bipartitioning with fast, area-driven floorplanning on all subproblems. The flow is outlined in Figure 1. At every level of the cutsize-driven, area-bipartitioning hierarchy, each node corresponds to a subset of blocks assigned by terminal propagation to a specific rectangular subregion of the chip. Before each application of cutsize-driven bipartitioning, however, one of two separate fast, area-driven floorplanners is used to check whether the given subproblem can be legalized. The fast floorplanner determines by a slicing construction whether the blocks assigned to each given subregion can in fact be shaped and laid out within that subregion without overlap. If so, then recursive cutsize-driven area bipartitioning continues in both subregions at the current level. If not, then the cutsize-driven solution at that level is discarded, and a wirelength-reducing symmetry of the previously computed, legal, "look-ahead" solution to the parent subproblem is used instead.[2] Because ZDS and ROB both produce slicing structures, their top-level cuts define floorplanning subproblems with known legal solutions. Cutsize-driven partitioning coupled with subproblem legalization then resumes recursively on these subproblems, until single-block base cases are reached.

The area-driven look-ahead floorplanners determine whether a legal solution exists for a given fixed-shape subregion and block subset. These algorithms must be fast and must usually find legal solutions if they exist. The first area-driven floorplanner, ZDS, is based on a recent study [10] of sufficient conditions for zero-dead-space floorplanning of soft blocks. ZDS is used only when all the blocks in the subregion are soft. Otherwise, a second area-driven floorplanner based on row-oriented block

[1]An acronym for "Partitioning To Optimize Module Arrangement." Pronounced *PAH-toh-ma*, from the Greek for "floor."

[2]Failure of ZDS (Section IV) or ROB (Section V) to find a legal *initial* solution, prior to recursive bipartitioning, is highly unlikely. We have not observed any such failure on any circuit.

```
Algorithm 2.1: PATOMA Floorplanning Algorithm
input: Set of blocks $\mathcal{S} = \{r_1, \ldots, r_m\}$; netlist; aspect ratio constraints for each block, rectangle $R$ of
fixed shape.
Each node of the partitioning tree is a set of blocks paired with a subregion. Generate the root node
$(\mathcal{S}, R)$ at level $i = 1$, and a legal floorplan for the root.
while there are still blocks to be placed
    while there are unvisited nodes at level $i$
        Select unvisited node $n = (\mathcal{S}_n, R_n)$ of level $i$.
        Use terminal propagation to model connections
            between $b_i \in \mathcal{S}_n$ and $b_j \notin \mathcal{S}_n$.
        Call hMetis to partition $\mathcal{S}_n$ into disjoint subsets $\mathcal{S}_{n1}$ and
            $\mathcal{S}_{n2}$, resp. assigned subregions $R_{n1}$, $R_{n2}$ of $R_n$.
        done := false.
        repeat
            remark Binary search for cutline position.
            for $i = 1, 2$
                if (all blocks in $\mathcal{S}_{ni}$ are soft)
                   fit[$i$] := ZDS($\mathcal{S}_{ni}, R_{ni}$).
                else fit[$i$] := ROB($\mathcal{S}_{ni}, R_{ni}$).
                end if
            end for
            if (fit[$j$] and not fit[$k$], $j, k \in \{1, 2\}$)
                slide the cutline toward $R_{nj}$
            else done := true.
            end if
        until (done or cutline search limit reached)
        if (fit[1] and fit[2])
            Create child nodes $n_1$ and $n_2$ of $n$.
            Store the solutions from or ZDS or ROB for possible
            future use.
        else replace the hMetis bipartitioning of $(\mathcal{S}_n, R_n)$ with
            a bipartitioning derived from earlier application
            of ZDS or ROB.
        end if
    end while
    $i := i + 1$.
end while
output: A floorplan of $\mathcal{S}$ in $R$ satisfying all area and aspect-ratio constraints.
```

Fig. 1. The PATOMA floorplanning algorithm.

packing (ROB) is used. ROB is somewhat similar to Traffic [25]; however, it handles both soft and hard blocks under a fixed-outline constraint. Both algorithms perform well in reasonable run time. They are reviewed in Sections IV and V below.

PATOMA uses the well-known multilevel partitioning package hMetis [16]. Neither of the two block subsets produced is allowed to hold more than 60% of the total area of all blocks in both subsets. This choice of area balance produced the best results in our experiments. Terminal propagation is used to account for connections between partitions.

Using feedback from the look-ahead floorplanners, PATOMA redistributes white space in order to make the result of cutsize-driven partitioning legalizable as often as possible. The exact location of the cutline is initially set in direct proportion to the total areas of the blocks in every partition. If a legal solution is found initially for R_1 but not for its sibling R_2, it may still be possible to find a legal solution for both partitions by moving white space from R_1 to R_2, i.e., by moving the cutline away from R_2 and toward R_1. Candidate cutline positions can be generated by binary search, as long as each cutline position results in a legal solution in at least one of the partitions.

III. The ZDS Algorithm

The ZDS algorithm generates zero-dead-space floorplans for a set of soft blocks inside a fixed-dimension region. The aspect ratios of the blocks of the final floorplan are uniformly bounded by the properties of the blocks. We will show that these bounds are realistic and satisfy the constraints for most existing benchmarks. Previous work on this subject either analyzed the theoretical upper bounds on the total area achieved by slicing floorplans of soft blocks [22], [32] or generate zero-dead-space floorplans with no consideration of aspect ratio constraints [30], [28]. The section also proves the properties of the algorithm.

The ZDS algorithm used here is based on recursive top-down area bipartitioning. At each step, the blocks in a region are separated into two groups such that the groups' total areas are as nearly equal as possible (i.e. for a group of blocks $a_1, ..., a_n$ the index j is found such that $D_j = |\sum_1^j a_i - \sum_{j+1}^n a_i|$ is minimized). The region is then cut parallel to its shorter side such that each group fits exactly into one of the regions. Cutting parallel to the shorter side keeps aspect ratios of subregions bounded in terms of the area variation among the blocks. Blocks are placed once they fill a sufficient fraction of their subregions; this fraction is expressed as the reciprocal of the parameter γ introduced below.

Figure 2 on page 7 shows the pseudocode for the ZDS algorithm. The notation is as follows. Given N rectangles $r_1, \ldots, r_N$ with fixed areas $a_1 \geq a_2 \geq \cdots \geq a_N$ but variable lengths ℓ_i and widths w_i, we seek to arrange them without overlap in a given rectangle $\mathcal{R}$ of area $\mathcal{A} = \sum_1^N a_i$ such that the aspect ratios

$$\rho_i = \rho(r_i) = \max(\ell_i/w_i,\ w_i/\ell_i)$$

are bounded close to 1.[3] The rectangle $\mathcal{R}$ is the *floorplanning region*. The rectangles r_i are the *blocks*.

Algorithm 3.1 is parametrized by $\rho(\mathcal{R}) \geq 1$ and $\gamma \geq 1$. By construction, Algorithm 3.1 has the following property.

Theorem 3.1: For $\rho(\mathcal{R}) \geq 1$ and $\gamma \geq 1$, Algorithm 3.1 generates a slicing floorplan with zero dead space. □

Although Algorithm 3.1 can accept as input any values $\rho(\mathcal{R}) \geq 1$ and $\gamma \geq 1$, the analysis shows that the generated floorplans display attractive properties for certain choices of $\rho(\mathcal{R})$ and γ. Specifically, if we define

$$\beta = \max_i a_i/a_{i+1}, \tag{1}$$

[3]By this definition, which we use for the remainder of the paper, the aspect ratio of every block is always at least 1.

```
Algorithm 3.1: Top-Down ZDS Floorplanning.
input
    (i) Rectangles $r_1, \dots, r_n$ with areas $a_1 \geq \dots \geq a_n$.
   (ii) Rectangular region $R$ of area $A = \sum_1^n a_i$ and dimensions $\ell \times w$, with $\ell / w = \rho(\mathcal{R}) \geq 1$.
  (iii) $\gamma \geq 1$.
end input
if the largest block $r_1$ satisfies $a_1 \geq \frac{1}{\gamma} A$ then
    (i) make the length of one side of $r_1$ equal to $w$.
    (ii) place $r_1$ with this shape against one side of $R$.
        remark In the analysis, let $R(r_1)$ denote this $R$.
    (iii) if $n > 1$ then
            Replace $R$ by the part of $R$ not used by $r_1$.
            Reindex $\{r_2, \dots, r_n\}$ to $\{r_1, \dots, r_{n-1}\}$.
            if $n == 2$ then
                place the last block in $R$ and return.
            else Replace $n$ by $n - 1$.
            end if
        else return
        end if
    remark If $R$ contains more than 2 blocks, we place at most one block
        before repartitioning.
end if

if $n > 1$ then
   1. Select $j \in \{1, \dots, n\}$ such that
          $$D_j = |\sum_1^j a_i - \sum_{j+1}^n a_i| \quad \text{is minimized.}$$
      Let $A_j = \sum_1^j a_i$ and $\bar{A}_j = \sum_{j+1}^n a_i \equiv A - A_j$.
   2. Cut $R$ parallel to its shorter side into two rectangular subregions $R_j$ and $\bar{R}_j$ of areas $A_j$ and
      $\bar{A}_j$ respectively. Assign $r_1, \dots, r_j$ to $R_j$ and $r_{j+1}, \dots, r_n$ to $\bar{R}_j$.
   3. Recur on the subregions $R_j$ and $\bar{R}_j$.
end if
```

Fig. 2. A Zero-Deadspace Floorplanning Algorithm.

then appropriate values for γ and $\rho(\mathcal{R})$ are

$$\gamma = \max\{2,\ \beta\} \quad \text{and} \quad \rho(\mathcal{R}) \in [1,\ \gamma + 1]. \tag{2}$$

A trace of the algorithm on a simple 5-block example is illustrated in Figure 3.

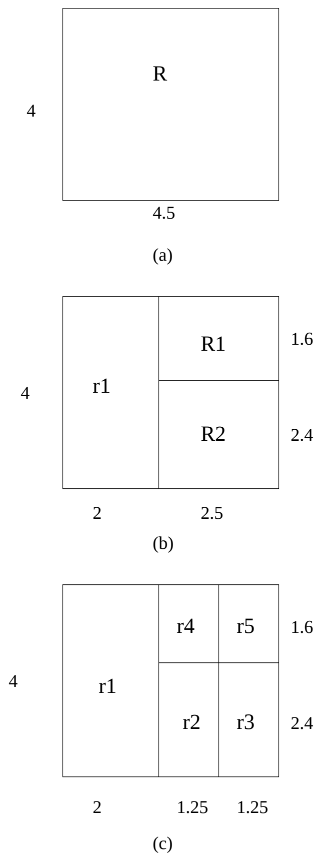

Fig. 3. (a) Creation of a ZDS benchmark with 5 blocks, where $a_1 = 8, a_2 = a_3 = 3, a_4 = a_5 = 2$. In this case, $\gamma = 2.67$. Region R has area 18 and is initialized to an aspect ratio of 1.125. (b) Since $a_1 \geq 18/2.67$, block r_1 is placed. After partitioning of the remaining blocks, region R_1 is assigned to blocks r_2 and r_3, and region R_2 to blocks r_4 and r_5. (c) Final placement of all blocks. The maximum aspect ratio is $2 \leq \gamma + 1$.

A. Analysis

The utility of Algorithm 3.1 rests in the fact that for nearly all realistic circuits, all the block aspect ratios it computes are guaranteed to lie within a single small interval of the form $[1, \gamma + 1]$, when γ is defined as in (2). Hence, if the blocks are arranged in nonincreasing sorted order by area, the aspect ratios are bounded by one plus the maximum ratio of consecutive block areas, when this latter ratio exceeds 2. Otherwise, the aspect ratios are bounded above by 3. These facts are established here, under Assumptions 3.1 below.

Assumptions 3.1: Block-locking threshold; floorplanning-region aspect ratio bound.

(a) For β as defined in (1), $\gamma \geq \max\{2, \beta\}$.

(b) The aspect ratio of the floorplanning region satisfies $\rho(\mathcal{R}) \leq \gamma + 1$.

Assumption 3.1(a) can be rephrased as follows: the threshold fraction of subregion area that a block must occupy in order to be shaped and locked in place is not set above $1/2$. Although these assumptions are stronger than necessary to achieve zero dead space and acceptably bounded block aspect ratios, they are not very restrictive on the sets of block areas that may be considered. Further discussion of the assumptions appears at the end of this section.

B. Derivation of the Aspect-Ratio Bound

Throughout this section, we consider the properties of Algorithm 3.1 under Assumptions 3.1. The first lemma shows that, in order to bound the aspect ratios of the blocks, it suffices to bound the aspect ratios of the regions in which they are placed.

Lemma 3.1: The aspect ratio ρ_i of any placed block r_i satisfies

$$\rho_i \leq \max\{\gamma,\ \rho(R(r_i))\},$$

where $\rho(R(r_i))$ denotes the aspect ratio of the smallest subregion in which r_i is placed.

Proof: Available online in a technical report [27]. □

The next lemma bounds the aspect ratio of sibling subregions in terms of their area ratio and the aspect ratio of their common parent subregion.

Lemma 3.2: Suppose subregion R is partitioned into subregions R_1 and R_2 with areas A_1 and A_2. Let

$$y = \max\left\{A_1/A_2,\ A_2/A_1\right\}.$$

Then

$$\max\{\rho(R_1),\, \rho(R_2)\} = \max\left\{\frac{y+1}{\rho(R)},\ \frac{y}{y+1}\rho(R)\right\}.$$

Proof: Available online in a technical report [27]. □

Lemma 3.3: With the notation in Algorithm 3.1,

$$D_j \leq \begin{cases} a_j & \text{if} \quad A_j > \bar{A}_j \\ a_{j+1} & \text{if} \quad A_j \leq \bar{A}_j. \end{cases}$$

Proof: Available online in a technical report [27]. □

Theorem 3.2: In Algorithm 3.1, under Assumptions 3.1,

$$\frac{1}{\gamma} \leq \frac{A_j}{\bar{A}_j} \leq \gamma.$$

Proof: Available online in a technical report [27]. □

From Lemma 3.2 and Theorem 3.2, we immediately obtain the following bound.

Corollary 3.1: Suppose subregion R is partitioned into subregions R_1 and R_2. Then

$$\max\{\rho(R_1),\, \rho(R_2)\} \leq \max\{\frac{\gamma+1}{\rho(R)},\, \frac{\gamma}{\gamma+1}\rho(R)\}.$$

Theorem 3.3: Under Assumptions 3.1, the result of Algorithm 3.1 is a slicing floorplan with zero dead space and every block's aspect ratio bounded above by $\gamma + 1$.

Proof: Follows directly from Corollary 3.1 and Assumptions 3.1, by induction. □

C. Remarks

The assumption $\gamma \geq 2$ presents no practical restriction on the sets of blocks that may be considered. It just means that the upper bound on block aspect ratios guaranteed by the analysis here for the given algorithm is at least 3. That is, consecutive-pairwise area bounds tighter than 2 (e.g., $a_i/a_{i+1} \leq 1.5$) are not guaranteed to reduce the maximum aspect ratio below what can be attained with $a_i/a_{i+1} \leq 2$.

Similarly, a large value of γ does not necessarily indicate any large aspect ratios in the final floorplan, as Figure III-C illustrates. In the figure, one large block occupies one subregion, and several small blocks occupy another subregion. Although the area ratio of the subregions may be arbitrarily large, the presence of sufficiently many small blocks used to fill the small subregion prevents any single block's aspect ratio from becoming large.

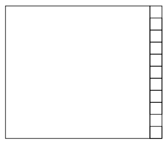

Fig. 4. Block aspect ratios may all remain small, even when some subregion has a large aspect ratio.

For some designs, the presence of a few very large or very small blocks may result in a large value of γ, if γ is defined simply as $\max\{2, \max_i a_i/a_{i+1}\}$. However, a few simple preprocessing steps can usually be used to reduce this value significantly. The basic idea is simply to aggregate smaller, similarly sized blocks together until the aggregates are more comparable to larger blocks or sets of blocks. Suppose $a_m/a_{m+1} = \max_i a_i/a_{i+1}$. Define $R_m, \bar{R}_m, A_m, \bar{A}_m$ as before, but with $j \equiv m$ instead of j being chosen to minimize D_j. If $A_m/\bar{A}_m < a_m/a_{m+1}$, then since a_{m+1} and a_m are placed in separate subregions, By recursive application of Theorem 3.2, γ is reduced to

$$\max\{A_m/\bar{A}_m,\, \max_{i\neq m} a_i/a_{i+1}\}.$$

If $\max_{i\neq m} a_i/a_{i+1} \gg A_m/\bar{A}_m$, a few recursive iterations on R_m and $\bar{R}_m$ can be used to reduce the bound further. Although it is trivial to construct examples where this preprocessing will be useless (e.g., when $N = 2$, or when $m = N - 1$), on practical examples with large N, the reduction in γ will likely be considerable, when m is sufficiently less than N.

For all the GSRC benchmarks, the value of γ is always 2. It is interesting that for the soft block version of these examples, the aspect ratio constraint for all the blocks is in the range [0.3, 3], which means that ZDS can find a zero-deadspace solution for all of them. This shows that the conditions required by ZDS are realistic and the algorithm itself can be used for the manipulation of soft blocks in a floorplanning algorithm.

IV. Wirelength-Aware ZDS Floorplanning

As described in the previous section, the ZDS algorithm ignores wirelength. In this section, the extensions of ZDS for PATOMA including wirelength consideration are presented.

PATOMA extends the original ZDS algorithm in two ways. First, available dead space is used to increase the frequency with which ZDS satisfies all aspect-ratio constraints. Let $\rho_{\max}$ denote the maximum aspect ratio allowed for any block. When $\gamma + 1 \leq \rho_{\max}$, success of ZDS is guaranteed, because the aspect ratios of the subregions for which ZDS is called are also in the range $[1/\rho_{\max}, \rho_{\max}]$, by the partitioning and cutline decisions made at the higher levels of the hierarchy. When $\gamma + 1 > \rho_{\max}$, the effective value of γ can be reduced by padding some of the blocks by dead space. If the reduction in γ is not enough to guarantee success, the ZDS algorithm is applied anyway, because its conditions for the creation of a legal solution are sufficient but not necessary. Second, in the original ZDS algorithm, the side of a subregion in which a block or block subset is placed is left unspecified. In PATOMA, when ZDS must be used instead of cutsize-driven bipartitioning to guarantee legalizability of the resulting subproblems, each block subset is placed in the subregion side that reduces the total lengths of connections between blocks in the subset and other blocks. An example is shown in Figure 5.

V. Row-based Floorplanning

The ROB (Row-Oriented Block Packing) heuristic is used by PATOMA for floorplanning a combination of fixed- and variable-dimension blocks. It is similar to Traffic [25] in that it organizes the blocks by rows according to their dimensions; however, it satisfies a a fixed-outline constraint and handles both hard and soft blocks. Assume given a set of blocks to be placed in a region with fixed height H and fixed width W. If $H > W$, the blocks will be organized in rows; otherwise, in columns. By organizing blocks in rows along the shorter subregion dimension, there is room to pack more rows, and therefore a wider variety of block heights can be efficiently supported. For the rest of this section, we assume, for simplicity, that the blocks are packed in rows.

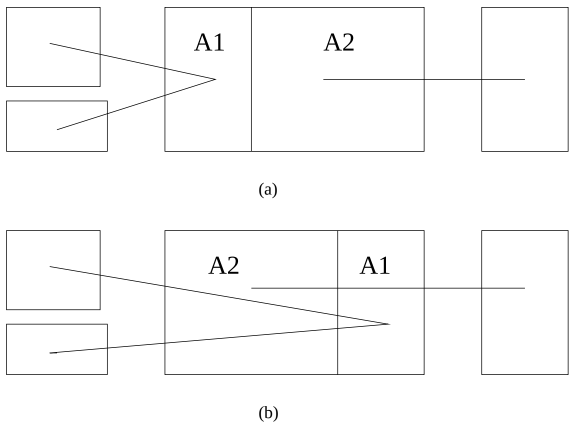

Fig. 5. Example of wirelength consideration during the enhanced ZDS algorithm. There are two alternatives for the relative location of the blocks in regions A_1 and A_2. The first alternative is selected because of its better wirelength.

ROB ignores connectivity. It consists of two stages. In the first stage, the blocks are grouped into rows according to their dimensions. In the second stage, emptier rows are merged with fuller rows until all rows fit inside the given, fixed-shape region. The outline of the ROB algorithm is shown in Figure 6. During the first stage, blocks are considered one by one and either added to existing rows or used to create new ones. Hard blocks are considered first. For every block, if one of its dimensions matches the height of an existing row and its addition to that row does not create overflow, it is placed there. Otherwise, a new row is generated with height equal to the smaller dimension of the block. Soft blocks are considered next. As they can be reshaped, they are more likely to match the height of an existing row. When a block can fit in multiple rows, the shortest one is preferred. If no such row can be found, a new one is generated with height equal to the smallest possible dimension of the block.

At the end of the first stage, a set of rows has been generated. Each row width is less than the fixed width W of the region [4], but it is possible that the sum of the row heights is larger than the fixed height H of the region. In the second stage, some rows are eliminated by redistributing blocks one by one. The rows are scanned in a decreasing height order. Blocks from rows shorter than the currently selected one are added to the selected row where possible. Priority is given to rows of smallest width. When a block is moved to another row, it is allowed to be rotated or reshaped for the purpose of matching the height of its new row as closely as possible without exceeding it. The procedure is repeated until either all the rows have been scanned, or enough rows have been eliminated such that the sum of the heights of the remaining rows is less than H. In the first case,

[4]... unless there is a block with both its dimensions larger than W. In that case a legal placement under the area constraints obviously cannot be found.

```
Algorithm 5.1: ROB Floorplanning Algorithm
input: Set of blocks $a_1, \ldots, a_n$; netlist; block aspect ratio constraints; region R with height $H$, width
$W$; $H \geq W$.
output: true if and only if a legal solution is found.
Order the blocks first by flexibility (hard blocks first) and then by area (larger area first).
for every block $b$
    Set of candidate rows initially empty
    for every row $r$
        if ( $B$ can match the height of the row by rotating
            or reshaping, the row width not exceeding $W$)
        then Add $r$ to the set of candidate rows
    end for
    if (set of candidate rows is empty)
        Create new row with height equal to the smallest
        valid dimension of $B$. Add $B$ to the new row.
    else
        Add $B$ to the shortest of the candidate rows.
    end if
end for
Curr_height := sum of row heights
if $Curr\_height \leq H$ return true
Order the rows according to height (taller first)
for every row $r$
    Add the rows shorter than $r$ to sequence $S$
    and order them by their widths (emptier first)
    for every row $r'$ in $S$
        for every block $a$ in $r'$
            Move $a$ to $r$ if no overflow in $r$. The height of $a$
            should be as close to (but not higher than) the
            height of $r$
            Update $Curr\_height$
                if $Curr\_height \leq H$ return true
        end for
    end for
end for
return false
```

Fig. 6. The ROB floorplanning algorithm.

the algorithm ends without finding a legal solution, while in the second it reports a success.

When legalizability of a cutsize-driven partition of a given subproblem cannot be ensured, ROB's solution to that subproblem is employed instead, by interpreting it as a partition. Since the solution of ROB is organized in rows (columns), it is guaranteed to have at least one slicing horizontal or vertical cut that can be used as the cutline for a bipartitioning of the blocks. The bipartitionings generated by these cuts are compared with their symmetric ones for wirelength, and the best bipartitioning is selected to replace the infeasible hMetis solution.

VI. Experiments and Results

We compare PATOMA to to Parquet-2 [1], a state-of-the-art SA-based floorplanner using the Sequence Pair geometric representation, Traffic [25] and FFPC, the fast floorplanner of Ranjan et al. [23]. For a fair comparison, all experiments were performed on the same machine, a 2.4GHz Pentium IV running RedHat Linux 8.0. We compared on four sets of benchmarks. For all the experiments, the floorplanners are trying to minimize the wirelength in a fixed outline. The first set of benchmarks includes the 4 largest GSRC circuits (size 200 - 300 blocks), where all the blocks are soft. For this set we compare only to Parquet-2, because in addition to the high-quality floorplans it produces, it is, as far as we know, the only freely available package online that can consider *both* fixed-outline constraints and soft blocks. We run Parquet-2 in two modes. The first mode is the default and is very fast, due to a shorter simulated-annealing schedule that hurts the wirelength quality. The second mode is a high-effort mode, where we impose a time limit of one hour to allow SA to attain a better solution.

A. Soft Blocks Only

Pad locations fix the amount of given white space at approximately 55–75%. In order to reduce wirelength, however, PATOMA restricts its floorplan to an inner core region with just 20% whitespace.[5] In the all-soft-block examples, PATOMA uses only the ZDS algorithm and not ROB to enforce the legalizability of all floorplanning subproblems. All blocks are allowed to be reshaped with any aspect ratios in $[1/3, 3]$. The end-case subproblems are observed to be small, containing fewer than 2 blocks on average and at most 27 on any instance. This result confirms that ZDS failures are quite rare and occur only for relatively small sets of blocks. The recursive, cutsize-driven flow thus proceeds on average almost all the way to individual blocks. The results are shown in Table I. The default mode of Parquet-2 produces results that are 19% higher in wirelength than PATOMA, while its runtime is 37$\times$ slower. The high-effort mode of Parquet-2 is 11% worse in wirelength and 824$\times$ slower than PATOMA.

B. Hard Blocks Only

The second set of experiments includes the same GSRC benchmarks, but with all blocks of given, fixed dimensions. In these examples, PATOMA uses only ROB and not ZDS to enforce the legalizability of floorplanning subproblems, because all blocks are hard. Table II shows the results for this set of experiments. On these benchmarks, PATOMA produces results of 10% lower wirelength than the default mode of Parquet-2, with a speedup of 33$\times$, and of 5% lower wirelength than the high-effort mode of Parquet-2, with an average speedup of 523$\times$.

The third set of experiments includes the same GSRC circuits all blocks hard, but without pads. PATOMA was compared

[5]On the soft-block examples, PATOMA's wirelength can actually be reduced a few percent by shrinking this inner region to hold just 5% white space.

Circuit	PATOMA		Parquet-2 (high-effort)		Parquet-2 (default)	
	WL	Run-time (sec)	WL Ratio	Run-time Ratio	WL Ratio	Run-time Ratio
n300	570388	4	1.20	501	1.34	46
n200	526011	3	1.08	561	1.15	29
n200b	537607	3	1.09	548	1.11	30
n200c	502128	2	1.10	863	1.15	43
Averages			1.12	824	1.19	37

TABLE I

COMPARISON OF PATOMA WITH PARQUET-2 ON THE LARGEST GSRC BENCHMARKS WITH ALL BLOCKS SOFT.

Circuit	PATOMA		Parquet-2 (high-effort)		Parquet-2 (default)	
	WL	Run-time (sec)	WL Ratio	Run-time Ratio	WL Ratio	Run-time Ratio
n300	645001	4	1.07	401	1.11	46
n200	550653	3	1.02	632	1.13	28
n200b	559772	3	1.05	528	1.08	29
n200c	526248	3	1.06	530	1.11	28
Averages			1.05	523	1.11	33

TABLE II

COMPARISON OF PATOMA WITH PARQUET-2 ON THE LARGEST GSRC BENCHMARKS WITH ALL BLOCKS HARD.

with Traffic and FFPC for these benchmarks, since these floorplanners do not use pads or shape soft blocks.[6] Table III lists the results of these experiments. FFPC's wirelength is 3% longer than PATOMA's, on average, while its run time is 6× longer. With Traffic's run-time limit set to PATOMA's run time, Traffic's average total wirelength is 60% longer than PATOMA's.

[6]FFPC does support *semi-soft* blocks, whose shapes must be selected from a fixed, finite set. Currently, PATOMA supports only soft blocks, not semi-soft. It is readily adapted to directly handle semi-soft blocks as well.

Circuit	PATOMA		FFPC		Traffic	
	WL	Run-time (sec)	WL Ratio	Run-time Ratio	WL Ratio	Run-time Ratio
n300	351463	4	1.03	6	1.71	1
n200	234085	3	1.09	9	1.52	1
n200b	193008	3	1.02	4	1.65	1
n200c	226147	3	0.96	5	1.53	1
Averages			1.03	6	1.60	1

TABLE III

COMPARISON OF PATOMA WITH TRAFFIC [25] AND FFPC [23] WITH ALL BLOCKS HARD. PADS ARE IGNORED.

C. Hard and Soft Blocks Together

In the fourth set of experiments, we generated large-scale floorplanning benchmarks from the IBM/ISPD98 suite [4] that include both hard and soft blocks on a fixed die with 20% whitespace. The soft blocks are clusters of standard cells generated by the First-Choice clustering heuristic [16]. The hard blocks are the same macros as in the original benchmarks. The allowed range of aspect ratios for the soft blocks was set at $[1/3, 3]$. The sizes of the benchmarks range from 500 to 2,000 blocks. We called this suite of benchmarks the HB-suite (hybrid blocks). These benchmarks are available online [12]. The characteristics of the benchmarks are shown in Table IV. The same table lists the results of both PATOMA and Parquet-2 (default mode) on them. For these examples, Parquet-2's wirelength is on average 104% higher than PATOMA's, while it is 209$\times$ slower.

Figure 7 shows a floorplan for the benchmark extracted from ibm01. All the benchmarks have an available deadspace of 20%. This is a reasonable value for modern fixed-size designs. The results show that the recursive-bisection flow of the PATOMA algorithm is very effective and efficient compared to other floorplanning algorithms.

VII. CONCLUSIONS

A new paradigm has been presented for floorplanning a combination of fixed- and variable-dimension blocks under a wirelength objective and a fixed-outline constraint. By constructively ensuring satisfiability of all constraints at each level by fast, area-driven heuristics, recursive cutsize-driven bipartitioning is allowed to proceed longer, and post-hoc legalization is eliminated. The resulting flow is scalable and produces superior wirelengths in orders of magnitude less run time than a leading

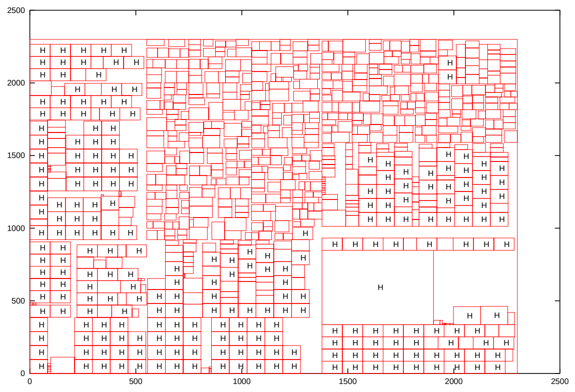

Fig. 7. Sample output of the PATOMA algorithm for the benchmark generated from ibm01. Hard blocks are shown with the H mark.

SA-based tool. In the current implementation, feedback to the bipartitioning is used only to adjust cutline positions, and only until satisfiability is ensured. More elaborate feedback can be expected to improve results significantly on benchmarks with more than 500 blocks.

VIII. Appendix

References

[1] S. Adya and I. Markov. Fixed-outline Floorplanning Through Better Local Search. In *Proc. International Conference on Computer Design*, pages 328–334, 2001.

[2] 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.

[3] 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, 2004.

[4] C.J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proc. Int'l Symp. on Phys. Design*, pages 80–85, 1998.

[5] U. Brenner, A. Pauli, and J. Vygen. Almost Optimum Placement Legalization by Minimum Cost Flow and Dynamic Programming. In *Proc. Int'l Symp. on Phys. Design*, pages 2–8, 2004.

[6] M.A. Breuer. Min-Cut Placement. *J. Design Automat. Fault Tolerant Comp.*, 1(4):343–362, Oct 1977.

[7] A.E. Caldwell, A.B.Kahng, and I.L. Markov. Improved algorithms for hypergraph partitioning. In *Proc. Asia South Pacific Design Automation Conf.*, 2000.

[8] T.F. Chan, J. Cong, T. Kong, and J. Shinnerl. *Multilevel Circuit Placement*, chapter 4 of *Multilevel Optimization in VLSICAD*. Kluwer, Boston, 2003.

[9] Y.C. Chang, Y.W. Chang, G. Wu, and S. Wu. B*-trees: A New Representation for Non-Slicing Floorplans. In *Proc. Design Automation Conference*, pages 458–463, 2000.

[10] J. Cong, G. Nataneli, M. Romesis, and J. Shinnerl. An Area-Optimality Study of Floorplanning. In *Proc. Int'l Symposium on Physical Design*, pages 78–83, 2004.

Circuit	# soft blocks	# hard blocks	PATOMA		Parquet-2	
			WL	Run-time (sec)	WL Ratio	Run-time Ratio
HB01	665	246	3.10E+06	12	1.81	159
HB02	1200	271	6.42E+06	35	2.40	101
HB03	999	290	9.80E+06	26	2.03	130
HB04	1289	295	1.10E+07	33	2.58	119
HB05	564	0	1.46E+07	20	1.53	179
HB06	571	178	8.83E+06	25	1.62	169
HB07	829	291	1.70E+07	41	2.16	128
HB08	968	301	1.88E+07	47	2.40	140
HB09	860	253	1.87E+07	34	2.47	175
HB10	809	786	5.39E+07	46	1.72	296
HB11	1124	373	2.89E+07	53	2.77	160
HB12	582	651	5.86E+07	42	1.35	465
HB13	530	424	3.69E+07	41	2.18	197
HB14	1021	614	6.60E+07	90	2.77	260
HB15	1019	393	9.04E+07	102	2.74	175
HB16	633	458	1.03E+08	84	2.20	245
HB17	682	760	1.46E+08	107	2.25	263
HB18	658	285	7.32E+07	71	2.40	237
Averages					2.19	209

TABLE IV
COMPARISON OF PATOMA WITH PARQUET-2 ON FLOORPLANNING BENCHMARKS DERIVED FROM THE IBM CIRCUITS.

[11] P. Guo, C. Cheng, and T. Yoshimura. An O-tree Representation of Non-slicing Floorplan and its Applications. In *Proc. Design Automation Conf.*, pages 328–334, 1999.

[12] http://cadlab.cs.ucla.edu/cpmo/HBsuite.html/.

[13] X. Hong, S. Dong, G. Huang, Y. Ma, Y. Cai, C. Cheng, and J. Gu. A Non-slicing Floorplanning Algorithm Using Corner Block List Topological Representation. In *Proc. Design Automation Conf.*, pages 268–273, 1999.

[14] A.B. Kahng. Classical Floorplanning Harmful? In *Proc. Int'l Symp. on Phys. Design,* pages 207–213, April 2000.

[15] A.B. 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] J. Lin and Y. Chang. TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans. In *Proc. Design Automation Conf.*, pages 764–769, 2001.

[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] S. Nakatake, K. Fujiyoshi, H. Mirata, and Y. Kajitani. Module Packing Based on the BSG-structure and IC Layout Applications. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 17, pages 519 – 530, 1998.

[21] R. Otten. Automatic Floorplan Design. In *Proc. Design Automation Conf.*, pages 261–267, 1982.

[22] H. Peixoto, M. Jacome, A. Royo, and J. Lopez. A tight upper bound for slicing floorplans. In *Proc. of IEEE VLSI 2000*, Jan. 2000.

[23] 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.

[24] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern Placement Techiques*. Kluwer, Boston, 2002.

[25] P. Sassone and S.K. Lim. A Novel Geometric Algorithm For Fast Wire-Optimized Floorplanning. In *Proc. International Conference on Computer-Aided Design*, 2003.

[26] C. Sechen. Chip Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing. In *Proc. Design Automation Conf.*, pages 73–80, 1988.

[27] J. Shinnerl. A theorem on partitioning a sorted list of numbers with an application to VLSI floorplanning. Report 040006, Computer Science Dept., University of California, Los Angeles, `http://www.cs.ucla.edu/~shinnerl`, Feb. 2004.

[28] K. Wang and W.-K. Chen. A class of zero wasted area floorplan for vlsi design. In *ISCAS*, pages 1762–1765, 1993.

[29] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large circuits. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 260–263, Apr 2000.

[30] S. Wimer, I. Koren, and I. Cederbaum. Floorplans, planar graphs, and layouts. In *IEEE Transactions on Circuits and Systems*, pages 267–278, 1988.

[31] D.F. Wong and C.L. Liu. A New Algorithm for Floorplan Design. In *Proc. Design Automation Conference*, pages 101 – 107, 1986.

[32] F.Y. Young and D.F. Wong. How Good are Slicing Floorplans? In *Proc. Int'l Symp. on Phys. Design*, pages 144–149, 1997.