Min-Cut Floorplacement*

Jarrod A. Roy†, Saurabh N. Adya‡, David A. Papa† and Igor L. Markov†

† University of Michigan, EECS Department, Ann Arbor, MI 48109-2122

‡ Synplicity Inc, 600 W. California Ave., Sunnyvale, CA 95054

Abstract

Large macro blocks, pre-designed datapaths, embedded memories and analog blocks are increasingly used in ASIC designs. However, robust algorithms for large-scale placement of such designs have only recently been considered in the literature. Large macros can be handled by traditional floorplanning, but are harder to account for in min-cut and analytical placement. On the other hand, traditional floorplanning techniques do not scale to large numbers of objects, especially in terms of solution quality.

We propose to integrate min-cut placement with fixed-outline floorplanning to solve the more general placement problem, which includes cell placement, floorplanning, mixed-size placement and achieving routability. At every step of min-cut placement, either partitioning or wirelength-driven, fixed-outline floorplanning is invoked. If the latter fails, we undo an earlier partitioning decision, merge adjacent placement regions and re-floorplan the larger region to find a legal placement for the macros. Empirically, this framework improves the scalability and quality of results for traditional wirelength-driven floorplanning. It has been validated on recent designs with embedded memories and accounts for routability. Additionally, we propose that free-shape rectilinear floorplanning can be used with rough module-area estimates before logic synthesis.

*A preliminary version of this work [4] was presented at ICCAD 2004.

1 Introduction

The amount of embedded memory used on a chip is expected to grow dramatically in the next few years [29], from around 50% of the die area in 2003 to 70% today, and 90% by 2011. This growth is mostly fueled by integrated circuits for high-bandwidth communication, portable multimedia, interactive consumer electronics and industrial embedded systems. While memories and random logic have traditionally been manufactured using different semiconductor processes, today most foundries offer hybrid processes that can produce reasonably dense memories embedded in random logic with fast gates and sophisticated interconnect [29]. The use of on-chip memories substantially improves energy-efficiency and response latency, while reducing weight, form factor and assembly costs.

Physical design with large pre-designed circuit blocks is more difficult than conventional standard-cell layout. While commercial layout tools have improved considerably in the last two years, the locations of large blocks are still typically determined manually. Moreover, this step is generally performed only once and separate from cell placement. This tends to lower utilization, increase die size, lower yield and increase cost [32]. Perhaps the most obvious challenge is the minimization of wirelength, which also affects routability. Optimization of wirelength is the most prevalent approach to placement and floorplanning, and enables other optimizations through the use of net weights and bounds [14, 17]. Moreover, wirelength optimization appears necessary — a recent study [26] from Intel shows that 51% of dynamic power in currently-shipped microprocessors is consumed when driving signals over interconnects, including local and global wires.

A recent article in EETimes [32] claims that traditional IC designs are moving from a “sea of cells” to a “sea of hard macros” System on Chip (SoC) paradigm (see Figure 1). They estimate that the number of hard macros in SoC designs will grow at a tremendous rate while the total area occupied by macros will grow much more slowly (see Figure 2). With this trend, SoC designs will be dominated by many small, hard macros which will only exacerbate the current difficulties in mixed-size design.

Automated placement of embedded memories, IP blocks and datapaths can improve time-to-market by quickly generating many high-quality layout scenarios, from which experienced

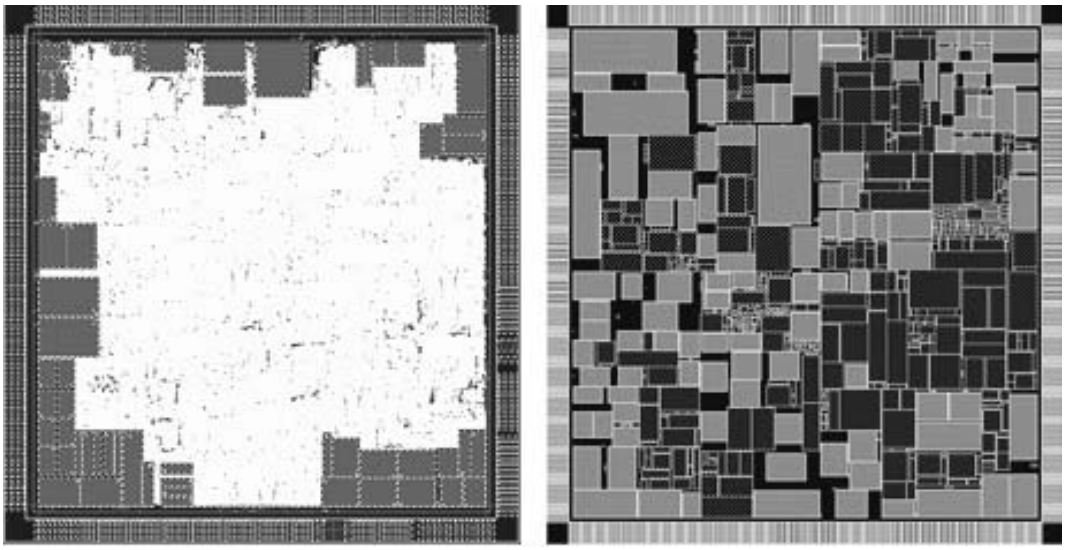
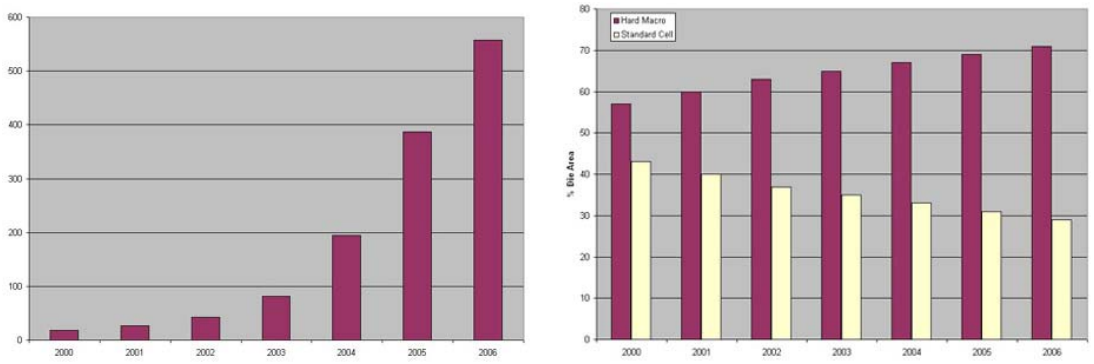


Figure 1: Traditional “sea of cells” IC vs. “sea of hard macros” SoC. Source: EETimes [32].

designers can select smaller candidate sets, using their domain knowledge. While there can be hundreds of large placeable circuit blocks, ideal block locations can also be influenced by millions of small standard cells. Accounting for this effect is often beyond human capabilities and is difficult in classical methodologies for automatic layout where floorplanning and placement are performed in separate steps. Traditionally, a circuit is first partitioned, and then floorplanned with rectangular shapes. The macro locations are fixed, and soft blocks are shaped, followed by standard-cell placement. In the past partitioning and floorplanning have often been used to increase the capacity of older placement algorithms which did not scale beyond half a million movable objects. However, modern placement algorithms, and even some of the academic tools used in this work, are routinely used on flat netlists with over four million movable objects.

From an optimization point of view, floorplanning and placement are very similar problems – both seek non-overlapping placements to minimize wirelength. They are mostly distinguished by scale and the need to account for shapes in floorplanning, which calls for different optimization techniques (see Table 1). Notice, however, that netlist partitioning is often used in placement algorithms, where geometric shapes of partitions can be adjusted. This considerably blurs the separation between partitioning, placement and floorplanning, raising the possibility that these three steps can be performed by one CAD tool. In this work, we develop such a tool and term the unified layout optimization *floorplacement* following Steve Teig’s keynote speech at ISPD



(a) (b)

Figure 2: (a) Growth in the number of hard macros in SoC designs. (b) Hard macros vs. standard cell area. Source: EETimes [32].

2002. We concentrate on fundamental algorithm development and present basic empirical validation. Clearly, industrial use will also require additional support with new methodologies, e.g., to allocate repeaters and optimize timing.

Our floorplacer Capo 9.2 is derived from an existing standard-cell placer and can also be used as a multi-way partitioner. Added functionalities include (1) completely integrated mixed-size placement competitive with best published results, (2) wirelength-driven fixed-outline floorplanning, that outperforms existing floorplanners by far, and (3) free-shape floorplanning that simultaneously determines locations and shapes of modules to optimize interconnect. Empirically, most modules are shaped as rectangles, with a noticeable fraction of L-, T- and U-shapes. However, we observe significantly smaller wirelengths and runtimes compared to purely rectangular floorplans.

One of the benchmark sets used in our empirical evaluation incorporates embedded memories with complete routing information. Embedded memories often use only two layers of metal (aside from power stripes) and do not block routing tracks at other metal layers. Therefore, our benchmarks mainly emphasize the effect of embedded memories on the placement of standard cells and can be viewed as a minimal sanity-check for mixed-size placement. In particular, we evaluate recent work on mixed-size placement [7, 25] which relies on greedy legalization of cell macro locations through left (or right) packing. Such strategies typically produce unroutable standard-cell placements [35, 6], and careful re-distribution of whitespace shown in [35] to improve routability may be less effective with large circuit blocks present, due to the fragmentation of layout. More

Characteristics	Partitioners	Floor-planners	Placers	Floor-placers
Scalable runtime	Yes	No	Yes	Yes
Scalable wirelength	N/A	No	Yes	Yes
Explicit non-overlapping constraints	No	Yes	No	Yes
Handling large modules	Yes	Yes	No	Yes
Routability optimization	No	N/A	Yes	Yes
Can optimize orientation of modules	No	Yes	No	Yes
Support for non-rectangular blocks	Yes	Limited	No	Yes
Support for soft rectangular blocks	Yes	Yes	No	Yes
Handling net weights	Yes	Yes	Yes	Yes
Handling length bounds	No	Yes	Yes	Yes

Table 1: A comparison of common algorithms for partitioning, floorplanning, and placement, contrasted with what can be achieved by a unified *floorplacer*. Published floorplanning algorithms assume a particular shape for each block, e.g., rectangle, L-shape or T-shape, but floorplacers may be able to automatically choose an acceptable shape.

generally, it seems that reliable incremental modification of mixed-size layouts is more difficult than that of pure standard-cell layouts. Therefore, in this work we attempt to minimize the need for such modification.

The rest of the paper is structured as follows. Section 2 describes relevant previous work. In Section 3 we integrate floorplanning into partitioning-based placement. Mixed-size placement benchmarks introduced in [4] are used for empirical validation in Section 4. Section 5 concludes our paper.

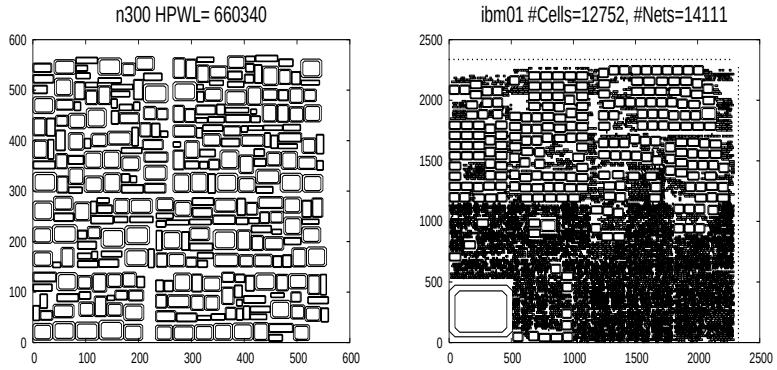
2 Relevant Previous Work

As pointed out in [20, 11, 3], modern hierarchical ASIC design flows are typically based on **fixed-die** floorplanning, placement and routing, rather than the older **variable-die** style. In such a flow, each top-down step may start with a floorplan of prescribed aspect ratio and with blocks of bounded, but not always fixed, aspect ratios.

2.1 Min-Cut Placement

Top-down placement algorithms seek to decompose a given placement instance into smaller instances by sub-dividing the placement region, assigning modules to subregions and cutting the netlist hypergraph [11]. In this context a *placement bin* represents (i) a placement region with allowed module locations (*sites*), (ii) a collection of circuit modules to be placed in this region, (iii) all signal nets incident to the modules in the region, and (iv) fixed cells and pins outside the region that are adjacent to modules in the region (*terminals*). The top-down placement process can be viewed as a sequence of passes where each pass examines all bins and divides some of them into smaller bins. Most commonly the division step is accomplished with balanced min-cut partitioning that minimizes the number of signal nets connecting modules in multiple regions. These techniques leverage well-understood and scalable algorithms for hypergraph partitioning and typically lead to routable placements.

This work uses the top-down placer Capo [11], which implements three min-cut partitioners — optimal (branch-and-bound), middle-range (Fiduccia-Mattheyses) and large-scale (multi-level Fiduccia-Mattheyses). Bins with seven cells or less are processed with an optimal end-case placer. To allow the partitioners to find better cuts, Capo often shifts the cutline to accommodate an excess of circuit modules in one partition. This also allows Capo to distribute the available whitespace uniformly [13] so as to facilitate easier routing. Non-uniform distribution can be easily achieved by pre-processing [1]. Recent enhancements are based on the concept of *placement feedback* [22] in which a given collection of bins is partitioned N times, without requiring steady improvement, to achieve more consistent terminal propagation. This change improves both wirelength and routability. Table 2 compares routability of placements produced by three leading min-cut placers on the IBM-Dragon (v2) benchmarks. We run Dragon 3.01 [35] in a mode where it spreads whitespace according to congestion. This significantly increases wirelength, but produces more routable placements. As of August 2004, FengShui [25] does not have such a mode and shifts all cells to the left (or right), typically yielding unroutable placements. We attribute Capo's routability to the fact that it generally produces placements with little or no overlap and allocates whitespace carefully and effectively.



(a) Block-based (b) Mixed-size

Figure 3: Layout styles. Standard-cell layout is shown in Figure 9(a).

2.2 Fixed-Outline Floorplanning

A typical floorplanning formulation deals with a set of circuit modules, each characterized by *area* and *shape type*. Rectangular modules (*blocks*) may have varying aspect ratios (*soft blocks*). This is common for IP blocks available in several shapes, and for hierarchical partitions where area can be estimated before synthesis. A *floorplan* specifies module locations and shapes such that modules do not overlap. Classical floorplanning minimizes a linear combination of floorplan area and total net length. However, in modern design flows the floorplan often has a fixed outline [20], which accentuates the minimization of wirelength, reminding of placement.

The floorplanner Parquet [3, 12] performs fixed-outline floorplanning with rectangular modules (supporting soft blocks) by combining Simulated Annealing with a new mechanism for move selection, based on *floorplan slack* [3]. Slack represents the amount of horizontal or vertical space next to each block and can be computed quickly. To improve the width of a floorplan, one must relocate a block with zero horizontal slack (similarly for height). Such moves are performed at regular time intervals during Simulated Annealing to bias the aspect ratio of the current floorplan to that of the desired outline. When the temperature schedule runs out, the final floorplan may still violate the outline. Parquet empirically achieves high rates of success on fixed-outline instances with 15% whitespace [3]. While speed is always a concern, in this work we target production-quality layout as opposed to fast estimation or virtual prototyping [27, 28].

Circuit	Capo 9.2 -feedback		Dragon 3.01 -fd		FengShui 2.6	
	Routed WL	V	Routed WL	V	Routed WL	V
ibm01e	**778885**	0	843001	0	time-out	932
ibm01h	**772578**	23	916508	84	time-out	2698
ibm02e	2183484	0	**2084652**	0	2201037	0
ibm02h	**2079678**	0	2215514	0	2277078	0
ibm07e	4533963	0	**4494732**	0	4755975	77
ibm07h	4590724	0	**4523371**	0	4707403	251
ibm08e	4552921	0	4600996	0	**4458175**	0
ibm08h	**4767853**	0	4961449	0	5056263	52
ibm09e	**3357281**	0	3704774	0	3519513	0
ibm09h	**3335810**	0	3494239	0	3395169	0
ibm10e	**6590734**	0	6948407	0	6808528	0
ibm10h	**6483647**	0	6981768	0	6715945	0
ibm11e	**5038608**	0	5371049	0	5300752	0
ibm11h	**4940703**	0	5400287	0	5260182	0
ibm12e	**9895311**	0	10458805	0	10146630	33
ibm12h	10145044	0	**9904268**	0	time-out	3418

Table 2: Routing results on IBM-Dragon V2 benchmarks with a 24-hour time-out. *V* stands for routing violations. Capo 9.2 was run with the -feedback option and uniform whitespace allocation, Dragon 3.01 was run in fixed die mode, and FengShui 2.6 was run with default parameters. Cadence WarpRoute typically routes Dragon's and Capo's placements, sometimes with a small number of violations. WarpRoute often fails on FengShui 2.6 placements. FengShui 5.0 produces routable placements on fewer benchmarks than FengShui 2.6.

2.3 Mixed-Size Placement

For the reasons outlined in the introduction, mixed-size placement is becoming increasingly important. Much progress has been made recently [1, 2, 16, 25, 34], and we survey relevant algorithms below.

The force-directed algorithm Kraftwerk [18] models interconnect with attraction forces and introduces additional repulsion forces between overlapping modules. The new module locations achieved by applying those forces are estimated by solving the Poisson equation, which is reduced to solving large sparse systems of linear equations. Forces are recomputed for each new placement, and the algorithm is applied until convergence. Kraftwerk is fast and can successfully handle large mixed-size placement instances *with significant amounts of whitespace*, but often fails to resolve overlaps between large modules in realistic circumstances where blocks may be difficult to pack [2]. In a recent empirical comparison of standard-cell placers [6], Kraftwerk was

outperformed by several min-cut placement tools. Another potential shortcoming of this analytical algorithm is having no provisions for optimizing orientations of large modules — a clearly discrete optimization problem.

MMP [34] attempts to solve the mixed-size placement problem by a bottom-up clustering of standard cells and subsequent cluster placement. The placement engine is a combination of quadratic and min-cut techniques. It balances partition areas by shifting the cut-line after each min-cut optimization. As described, the algorithm assumes pre-determined orientations for all circuit modules and does not attempt to optimize them. No empirical comparisons to other techniques or scalability data are available. It is especially unclear if this technique can handle large, fixed-size, difficult-to-pack blocks.

The work in [2] proposes a methodology for mixed-size placement that combines floorplanning and standard-cell techniques as follows.

Step 1. During pre-processing, each large module is shredded into small fake cells connected by a grid of fake wires. Pins are propagated to shredded cells to reflect pin offsets. Assigning sufficiently high weights to fake wires ensures that fake cells belonging to the same large module are placed next to each other if the placer minimizes linear wirelength. A black-box standard-cell placer is applied to the shredded netlist.

Step 2. Initial locations of large modules are computed by averaging the locations of respective fake cells. A module is rotated according to the prevailing orientation in the grid that models it. To remove overlaps between large modules, small cells are clustered (bottom up, based on locations) into soft blocks to create a fixed-outline floorplanning instance with 100-200 blocks.

Step 3. Non-overlapping locations of large modules are generated by running a fixed-outline floorplanner, e.g., Parquet [3]. Initial locations can be discarded, or else can be re-used with low-temperature annealing during floorplanning.

Step 4. Large modules are fixed, and remaining soft blocks are disintegrated into original standard cells. The black-box standard-cell placer is called again to re-place small cells.

Observe that the shredding process facilitates physical (location-based) clustering of small cells and thus improves final locations of large modules, even if their initial locations are discarded. A major advantage of this methodology is its robustness — it often produces legal place-

ments when other approaches leave large overlaps or place modules out of core. It also optimizes module orientations. This fully-automated methodology successfully competed with a major commercial tool in 2002 and has been recently improved by more judicious handling of whitespace [1]. Yet, the main scalability bottleneck remains in the use of Simulated Annealing at the top-level floorplanning stage. It affects both runtime and the quality of wirelength optimization.

The multi-level placer mPG-MS [16] clusters the netlist bottom-up to build a hierarchy. The top-level coarse netlist of approximately 500 clusters is placed using Simulated Annealing, after which the netlist is gradually unclustered so as to improve the placement of smaller clusters by incremental annealing. All intermediate cluster placements in mPG-MS are non-overlapping, which is enforced with specially-designed data structures and yet takes considerable computational effort. This and the pervasive use of Simulated Annealing make mPG very slow. While mPG finds better placements than those reported in [1], even better placements have been produced recently by the min-cut technique below, which is also much faster.

The work in [25] advocates a two-stage approach to mixed-size placement. First, the min-cut placer FengShui [7] generates an initial placement for the mixed-size netlist without trying to prevent all overlaps between modules. The placer only tracks the global distribution of area during partitioning and uses the *fractional cut* technique [7], which further relaxes book-keeping by not requiring placement bins to align to cell rows. While giving min-cut partitioners more freedom, these relaxations prevent cells from being placed in rows easily and require additional repair during detail placement. This may particularly complicate the optimization of module orientations, not considered in [25] (relevant benchmarks use only square blocks with all pins placed in the centers).

The second stage consists of removing overlaps by a fast legalizer designed to handle large modules along with standard cells. The legalizer is essentially greedy and attempts to shift all modules towards the left edge of the chip (or to the right edge, if that produces better results). In our experience, the implementation reported in [25] leads to horizontal stacking of modules and sometimes yields out-of-core placements, especially when several very large modules are present (the benchmarks used in [25] contain numerous modules of medium size). See Figure 10 for examples of this behavior. Another concern about packed placements is the harmful effect of

such a strategy on routability, explicitly shown in [35]. Overall, the work in [25] demonstrates very good legal placements for common benchmarks, but questions remain about the robustness and generality of the proposed approach to mixed-size placement. We address these questions with additional benchmarking in our work.

3 Integration of Partitioning, Placement and Floorplanning

In this section we introduce our correct-by-construction approach to floorplacement, which does not rely on post-placement legalization procedures for large modules. We first discuss the main algorithm that combines partitioning, placement and floorplanning. Next we describe improved construction of floorplanning instances to improve the speed of the algorithm. For the rare case that our technique fails to produce a non-overlapping placement, we next discuss overlap removal. An alternative technique that can simultaneously place and shape non-rectangular modules is also presented. Lastly we discuss practical considerations and sketch implementation details.

3.1 Unified Placement and Floorplanning

We first observe that min-cut placers scale well in terms of runtime and wirelength minimization, but cannot produce non-overlapping placements of modules with a wide variety of sizes. On the other hand, annealing-based floorplanners can handle vastly different module shapes and sizes, but only for relatively few (100-200) modules at a time. Otherwise, either solutions will be poor or optimization will take too long to be practical. As explained in Section 2.3, the loose integration of fixed-outline floorplanning and standard-cell placement proposed in [2] suffers from a similar drawback because its single top-level floorplanning step may have to operate on numerous modules. Bottom-up clustering can improve the scalability of annealing, but not sufficiently to make it competitive with other approaches. Therefore, in this work we apply min-cut placement as much as possible and delay explicit floorplanning until it becomes necessary. In particular, since min-cut placement generates a slicing floorplan, we view it as an implicit floorplanning step, reserving explicit floorplanning for “local” non-slicing block packing.

We start with a single placement bin representing the entire layout region with all the placeable objects initialized at the center of the placement bin. Using min-cut partitioning, the bin

```
 Variables:  queue of placement bins
 Initialize queue with top-level placement bin
1   While (queue not empty)
2     Dequeue a bin
3     If (bin has large/many macros or is marked as merged)
4       Cluster std-cells into soft macros
5       Use fixed-outline floorplanner to pack
          all macros (soft+hard)
6       If fixed-outline floorplanning succeeds
7         Fix macros and remove sites underneath the macros
8       Else
9         Undo one partition decision. Merge bin with sibling
10        Mark new bin as merged and enqueue
11    Else if (bin small enough)
12      Process end case
13    Else
14      Bi-partition the bin into smaller bins
15      Enqueue each child bin
```

Figure 4: Our floorplacement algorithm. Bold-faced lines 3-10 are different from traditional min-cut placement.

is split into two bins of similar sizes, and during this process the cut-line is adjusted according to actual partition sizes. Applying this technique recursively to bins (with terminal propagation) produces a series of gradually refined slicing floorplans of the entire layout region, where each room corresponds to a bin.[1] In very small bins, all cells can be placed by a branch-and-bound end-case placer [9]. However, this scheme breaks down on modules that are greater than their bins. When such a module appears in a bin, recursive bisection cannot continue, or else will likely produce a placement with overlapping modules. Indeed, the work in [25] continues bisection and resolves resulting overlaps later. However, in this work we switch from recursive bisection to "local" floorplanning where the fixed outline is determined by the bin. This is done for two main reasons: (1) to preserve wirelength [10], congestion [8] and delay [21] estimates that may have been performed early during top-down placement, and (2) avoid the need to legalize a placement with overlapping macros. In particular, we are not convinced that existing legalization algorithms are robust enough to handle a wide variety of module shapes and sizes in realistic netlists (see Figure 10). We also anticipate difficulty ensuring routability while shifting macros and standard cells at the same time.

[1] If every cut-line is fixed *apriori* to the center of its bin, recursive bisection generates a grid-like floorplan.

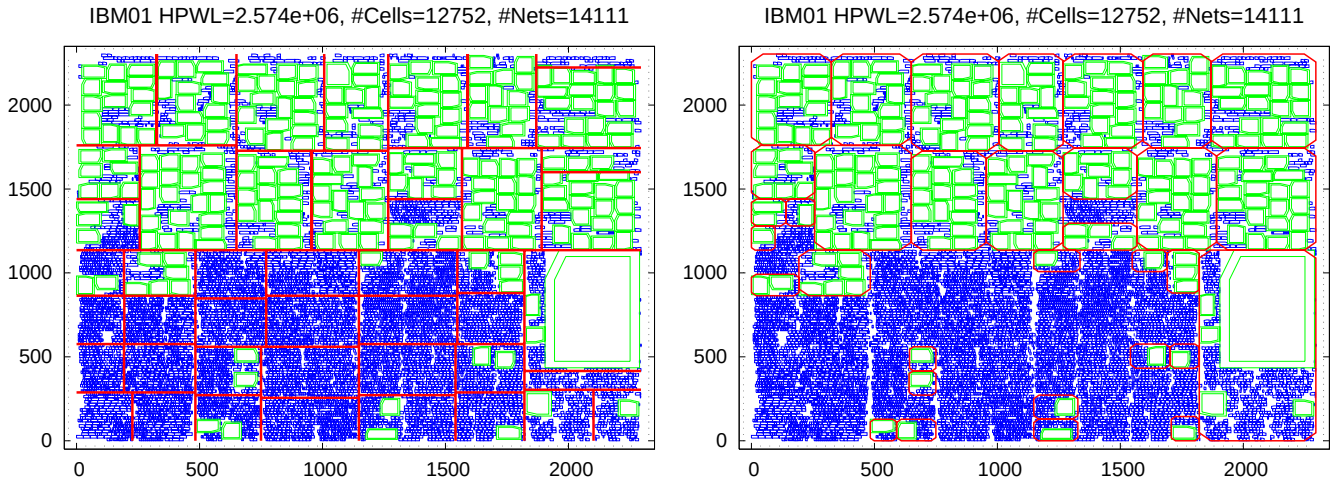


Figure 5: These images illustrate the progress of mixed-size placement by Capo 9.2 on the IBM01 benchmark from `IBM-MSwPins`. The picture on the left shows how the cut lines are chosen during the first six layers of min-cut bisection. On the right is the same placement but with the floorplanning instances highlighted by "rounded" rectangles. Floorplanning failures can be detected by observing nested rectangles.

While deferring to fixed-outline floorplanning is a natural step, successful fixed-outline floorplanners have appeared only recently [3]. Additionally, the floorplanner may fail to pack all modules within the bin without overlaps. As with any constraint-satisfaction problem, this can be for two reasons: either (i) the instance is unsatisfiable, or (ii) the solver is unable to find any of existing solutions. In this case, we undo the previous partitioning step and merge the failed bin with its sibling bin, whether the sibling has been processed or not, then discard the two bins. The merged bin includes all modules contained in the two smaller bins, and its rectangular outline is the union of the two rectangular outlines. This bin is floorplanned, and in the case of failure can be merged with its sibling again. The overall process is summarized in Figure 4 and an example is depicted in Figure 5.

It is typically easier to satisfy the outline of a merged bin because circuit modules become relatively smaller. However, Simulated Annealing takes longer on larger bins and is less successful in minimizing wirelength. Therefore, it is important to floorplan at just the right time, and our algorithm determines this point by backtracking. Backtracking does incur some overhead in failed floorplan runs, but this overhead is tolerable because merged bins take considerably longer to floorplan. Furthermore, this overhead can be moderated somewhat by careful prediction, as will

be described later.

For a given bin, a floorplanning instance is constructed as follows. All connections between modules in the bin and other modules are propagated to *fixed terminals* at the periphery of the bin. Similar terminal propagation schemes are commonly used in some analytical placers [31]. As the bin may contain numerous standard cells, we reduce the number of movable objects by conglomerating standard cells into soft placeable blocks. This is accomplished by a simple bottom-up connectivity-based clustering [23]. The existing large modules in the bin are usually kept out of this clustering. To further simplify floorplanning, we artificially downsize soft blocks consisting of standard cells, as in [1]. The clustered netlist is then passed to the randomized fixed-outline floorplanner Parquet, which sizes soft blocks and optimizes block orientations. We allow at most five attempts to find a non-overlapping placement of modules within the bin. If the floorplanner is successful, we attempt to shift all the floorplanned modules simultaneously within the bin boundary to further improve wirelength. After suitable locations are found, the locations of all large modules are returned to the top-down placer and are considered fixed. The rows below those modules are fractured and their sites are removed, i.e., the modules are treated as fixed obstacles. At this point, min-cut placement resumes with a bin that has no large modules in it, but has somewhat non-uniform row structure. When min-cut placement is finished, large modules do not overlap by construction, but small cells sometimes overlap in few places (typically below 0.01% by area). Those overlaps are quickly detected and removed with local changes. Detailed placement uses branch-and-bound placement in sliding windows [9], but does not move the macros. Figures 3, 5 and 9 show sample placements produced by our tool.

Since our floorplacer includes a state-of-the-art floorplanner [3], it can natively handle pure block-based designs. Unlike most algorithms designed for mixed-size placement, it can pack blocks into a tight outline, optimize block orientations and tune aspect ratios of soft blocks. Indeed, when the number of blocks is very small, our algorithm applies floorplanning right away. However, when given a larger design, it may start with partitioning and then call fixed-outline floorplanning for separate bins. This is demonstrated in Figure 3(a) which shows the block-based design `n300` placed using our floorplacer. The cuts made by the min-cut partitioner are clearly seen making the resulting floorplan globally slicing, but locally non-slicing. Since recursive bisec-

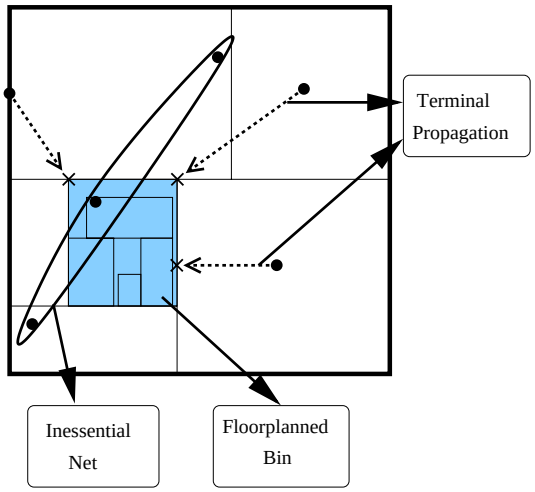


Figure 6: Terminal propagation during the floorplanning of a placement bin. The shaded placement bin is being floorplanned because of large/many macros. Dotted lines depict the external connections to objects inside the bin, being propagated as terminals to the placement bin boundaries. An inessential net for floorplanning is also shown. The shown 3-pin inessential net has no consequence on the floorplanning decision for HPWL minimization and is removed when forming the floorplanning problem.

tion scales well and is more successful at minimizing wirelength than annealing-based floorplanning, the proposed approach is scalable and effective at minimizing wirelength. This expectation is fully confirmed by empirical results in Section 4.

3.2 Improved Construction of Floorplanning Instances

For realistic designs, during floorplanning, the number of wires between modules is generally very large compared to the number of modules. Packing oriented floorplan representations such as Sequence Pair and B* trees suffer from the lack of incremental nature of the floorplan evaluation algorithm. Thus, wirelength for each prospective move during simulated annealing has to be calculated from scratch. In such a framework, the runtime is dominated by the evaluation of wirelength for a move rather than the computation of a new floorplan during each move [15]. Therefore we outline two ways to improve the runtime of the floorplanning stage of our proposed mixed-size placement flow by reducing the number of wires and pins generated for the floorplanning instance.

Inessential nets. Figure 6 shows the terminal propagation performed before floorplanning a bin

within the proposed min-cut floorplacement flow. All connections between modules in the bin and other modules are propagated to fixed terminals *at the periphery* of the bin. Similar terminal propagation schemes are commonly used in some analytical placers [31]. If the bounding box of any multi-pin net covers the bounding box of the entire bin, then this net will have no consequence on the HPWL minimization during floorplanning. Such a net is treated as *inessential* and ignored when forming the floorplanning problem. A sample three-pin inessential net is shown in Figure 6.

Net conglomeration. As the placement bin may contain numerous standard cells, we reduce the number of movable objects by conglomerating standard cells into soft placeable blocks. This is accomplished by a simple bottom-up connectivity-based clustering [23]. The existing large modules (macros) in the bin are usually kept out of this clustering.

Figure 7 shows an efficient algorithm for removing duplicate two-pin nets when forming the floorplanning problem. For each node, a list of adjacent nodes is built for all of its two-pin connections. Care is taken to only traverse each net once. Thus if two nodes *a* and *b* are connected by several two-pin nets and *a* is processed before *b*, *a* will have multiple entries of *b* in its adjacency list and *b* will have 0 entries of *a* in its adjacency list. After building the two-pin adjacency lists, the lists are sorted to quickly locate duplicate connections. One new weighted two-pin connection is created for each pair of nodes on a two-pin connection with the weight determined by the total weight of the of two-pin nets that connected them in the original netlist.

3.3 Legalization of Macro Overlap

Although our floorplanning conditions have been designed and refined so as to prevent floorplanning failure and backtracking, in rare cases overlaps exist after placement when whitespace is low or in the presence of fixed obstacles. In these cases we must remove the overlaps and produce a legal placement.

Given the techniques we use to fracture rows beneath macros so that macros and standard cells will not overlap by construction, we first remove overlap between macros, rebuild the subrow structures, move any standard cells that were displaced due to the movement of macros to the nearest subrow, and remove any remaining standard cell overlap as normal.

```
 Input:  nodes:  list of nodes
 Input:  origNets:  list of original nets
 Output:  newNets:  list of conglomerated nets
 Variables:  seenNets:  list of seen nets
 Variables:  list node.nodeConn for each node in nodes
 Initialize seenNets with false for each net
 Initialize node.nodeConn for each node in nodes with NULL

1   foreach (node in nodes)
2     foreach (net in node.nets)
3       if(!seenNets[net])
4         if(net.degree > 2)
5           add net to newNets
6         else
7           foreach (nextNode in net.nodes)
8             if(nextNode.idx != node.idx)
9               add nextNode to node.nodeConn
10          end
11        seenNets[net] = true
12    end
13  end
14  foreach (node in nodes)
15    sort node.nodeConn
16    gather multiple two-pin connections between the same
        nodes by a linear traversal of node.nodeConn
17    make a new single weighted connection ``net''
        for each set of multiple two-pin connections
18    add net to newNets
19  end
```

Figure 7: Net conglomeration pseudocode. In this code, we remove redundant two-pin nets between the blocks in a clustered floorplanning instance and replace them with a single weighted net to increase speed. We do not consider nets that have more than two pins because redundant two pin nets are by far the most plentiful.

Standard cells and macros may be packed tightly to each other, so moving macros around without care can cause the displacement of many standard cells, eventually increasing HPWL. To try and prevent this, the first stage of our overlap remover tries to move macros around as little as possible. First, one identifies all movable macros that overlap with either other macros or fixed objects. For each macro, we examine eight possible moves for alleviating each of its overlap conditions (up, down, left, right and the four diagonal directions). Each move is constructed to be just enough to remove the overlap in question. We proceed in a greedy fashion by making the move that clears as much overlap as possible while considering move distance as the secondary

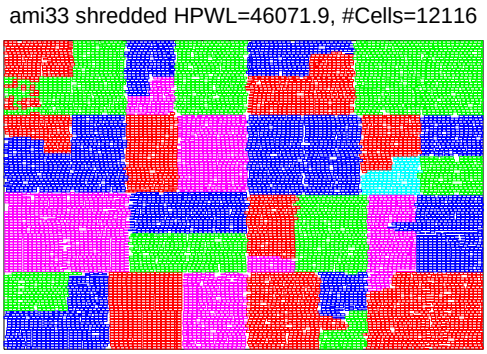
tie breaker. Moves are performed in this manner until all overlap is removed or there are no moves available that reduce overlap.

To allow hill-climbing, each block can increase overlap at most once. Such a move produces an obvious move the next time around that will just move the macro back to its original position, so we disallow these moves for one turn so that actual progress can be made if possible. Note that we should never have more overlap than what we started with at the termination of this phase because any moves that increased overlap will be greedily undone if they do not help leave a local minimum.

In the majority of cases, this first phase is able to remove all overlaps between macros and displace very few standard cells. Since it is a fairly simple, greedy method, there are situations where it does fail. One such situation is in the presence of many fixed objects. If a macro were to become placed in the middle of an area with many fixed objects such that it could not find a close legal location, the first phase will usually not be able to remove the overlaps. For these cases, a greedy technique is employed. For each macro that overlaps with other macros or fixed objects, the macro is moved to the rectilinearly closest location free fom overlaps (without regard to any standard cells that may already be there). Larger macros are processed before smaller macros.

3.4 Free-Shape Rectilinear Floorplanning

During mixed-size placement, soft blocks arise by clustering standard cells. Some circuit modules, however, such as embedded memories and pre-designed datapaths, have fixed rectangular shapes. A third category of shape constraints occurs when only the area of a module is estimated, but its shape is unknown and is free to change – there is often no *a priori* reason to limit its shape to a rectangle. Non-rectangular floorplanning has been popular in several design contexts, and existing work can be classified by whether the floorplanner is allowed to change the shape type of modules. To this end, the work in [24] and [33] represents simple non-rectangular shapes with Sequence Pairs (SP) and Bounded Slicing Grids (BSG) to pack such modules using the popular annealing-based framework. In contrast, the work in [19] solves a specific floorplanning formulation proposed in [20], which assumes desired locations of given rectangular modules and seeks to re-shape the modules so as to avoid overlaps. The proposed algorithm is an incremental detailed



Type→	Rectangular	Free-shape	
Circuit ↓	Parquet 2.0 HPWL	Capo 9.2 HPWL	Avg% Impr
ami33	76987	46072	40.1
ami49	895560	469476	47.5
n50	202240	87957	56.5
n100	350593	157548	55.0

Figure 8: The image on the left depicts a free-shape floorplan of the `ami33` benchmark. Our floorplacer determines both locations and shapes of individual modules to minimize wirelength. Traditional rectangular floorplanning with Parquet is compared to our free-shape non-rectangular floorplacement on the right.

floorplanner that tends to generate fairly complicated shapes, but does not account for interconnect. Below we extend our global free-shape floorplanner to generate both locations and shapes of soft modules so as to minimize interconnect. Empirically, most of the modules are shaped as rectangles, but L-,T- and U-shapes are sometimes created when this helps reducing interconnect. Our algorithm is also capable of pin placement.

We rely on techniques proposed in [2], where each large module is pre-processed into a grid of fake cells and heavy fake nets. Signal pins of a module are propagated to respective fake cells. However, in our context there is no need to shred fixed-shape blocks because they are already handled by our floorplacer. Thus, we only shred soft blocks. As in [2], heavy weights on fake nets ensure that shreds of the same module stay together during min-wirelength placement. However, since we now allow non-rectangular shapes, there is no need to average locations of fake cells and determine the prevailing orientation as in [2]. We simply accept module shapes assumed by fake grids during placement. Because of the relative rigidity of fake grids and because we rely on min-cut placement, most modules assume rectangular shapes, which is convenient from many perspectives. Other shapes are generated only when this reduces interconnect, and they remain relatively simple. This is demonstrated in Figure 8 where modules are color-coded. The plot is produced by our floorplacer using fake-net weights of 500. An additional benefit of our approach is its scalability, e.g., if no hard blocks are present, the entire layout is completed without Simulated Annealing. Figure 8 reports the improvement in runtime and wirelength over traditional

Floorplanning conditions used in Capo 9.2	
N, n:	The numbers of large modules and movable objects in a given bin.
$A(m)$:	The area of the m largest modules in a given bin, $m \leq n$.
C:	The capacity of a given bin.
Test 1.	At least one large module does not fit into a potential child bin.
Test 2.	$N \leq 30$ and $A(N) < 0.80 * A(n)$ and $A(n) > 0.6 * C$.
Test 3.	$N \leq 15$ and $A(N) < 0.95 * A(n)$ and $A(n) > 0.6 * C$.
Test 4.	$A(50) < 0.85 * C$.
Test 5.	$A(10) < 0.60 * C$.
Test 6.	$A(1) < 0.30 * C$ and $N = 1$.
Test 7.	$N = n = 1$.

Table 3: Floorplanning conditions used in Capo 9.2. Test 1 is the most fundamental, for if a bin meeting test 1 were not floorplanned, a failure would be guaranteed at the next level. Tests 2-6 detect bins dominated by large macros. Test 7 is a base case where only one module exists, but it is large.

rectangular floorplanning with Parquet on a mix of MCNC and GSRC floorplanning benchmarks. For larger designs, wirelength is reduced by more than 50%. We expect that this new type of free-shape floorplanning can be useful before logic synthesis to determine relative locations of large modules and enable early estimates of signal delays in global interconnect.

3.5 Practical Issues and Implementation Details

Empirical boundary between placement and floorplanning. By identifying the characteristics of placement bins for which our algorithm calls floorplanning, one can tabulate the empirical boundary between placement and floorplanning. Formulating such *ad hoc* thresholds in terms of dimensions of the largest module in the bin, etc allows one to avoid unnecessary backtracking and decrease the overhead of floorplanning calls that fail to satisfy the fixed outline constraint because they are issued too late. In practice, issuing floorplanning calls too early (i.e., on larger bins) increases final wirelength and sometimes runtime. To improve wirelength, our *ad hoc* tests for large modules in bins (that trigger floorplanning) are deliberately conservative. For the purpose of this discussion a large module is defined as one with height larger than the height of one row. Tests currently used in Capo 9.2 are listed in Table 3.

These conditions were derived by closely monitoring the legality of floorplanning and min-cut placement solutions. When a partitioned bin yields an illegal placement solution it is clear that the

Benchmark	#Macros	Time (s)	%Capo time in floorplaning	#FP runs	#Failed	Max blocks in FP instance	Max blocks in failed FP instance
ibm01	246	66.38	25	67	4	22	9
ibm02	271	95.23	16	42	2	53	6
ibm03	290	109.23	13	48	5	108	24
ibm04	295	179.40	19	62	3	47	7
ibm05	0	0.00	0	0	0	0	0
ibm06	178	58.09	6	56	0	48	0
ibm07	291	183.65	12	84	4	111	3
ibm08	301	271.15	16	62	2	91	2
ibm09	253	245.03	13	65	2	64	4
ibm10	786	805.53	23	90	5	166	6
ibm11	373	250.72	10	59	2	139	1
ibm12	651	413.40	12	125	3	51	11
ibm13	424	321.03	10	84	5	55	4
ibm14	614	576.15	10	199	12	57	14
ibm15	393	1495.33	18	24	3	163	32
ibm16	458	398.65	5	145	7	47	3
ibm17	760	304.63	3	150	8	57	13
ibm18	285	171.09	3	127	4	24	10

Table 4: Floorplanning statistics for the `IBM-MSwPins` mixed-size benchmarks.

bin should have been floorplanned and a condition should be derived. When a call to floorplanning fails to satisfy the fixed outline constraint the placer has to backtrack by merging two child bins and floorplanning the parent. To avoid paying this penalty, a condition should be derived to allow for floorplanning the parent bin and prevent the failure all together.

We refine these conditions to prevent floorplanning failure by visual inspection of a plot of the resulting parent bin and formulating a condition describing its composition. An example of such a plot is shown in Figure 5. Floorplanned bins are outlined with rounded rectangles. Nested rectangles indicate a failed floorplan run, followed by backtracking and floorplanning of the larger parent bin.

In our experience, these tests are strong enough to ensure that at most one level of backtracking is required to prevent overlaps between large modules. The effectiveness of these conditions is demonstrated in Table 4. On average, there are fewer than ten failures per run, and the number of failures does not grow with benchmark size.

Side-effect: Narrow vertical slivers between large modules. Adjacent large modules placed

Circuit	#Blocks	Parquet 4.0		Capo 9.2		
		HPWL	Time sec	HPWL	Time sec	# Min-cut Levels
n10	10	5.75	0.13	5.78	0.49	0
n30	30	16.88	1.21	16.75	1.35	**1**
n50	50	20.97	3.40	20.02	3.09	**1**
n100	100	33.74	13.29	**31.23**	**5.27**	**2**
n200	200	63.77	60.45	**55.52**	**23.93**	**3**
n300	300	75.63	127.68	**63.94**	**32.32**	**3**

Table 5: Floorplanning versus floorplacement. The last column "Levels" lists the number of min-cut levels executed before the first floorplanning step. All data are averaged over 10 independent runs.

by the fixed-outline floorplanner may have tall, narrow columns of empty sites between them. Fitting small cells in such slivers may be non-trivial, e.g., consider a column with four sites and a collection of cells that take two or three sites each. In this case, every three-site cell implies the loss of one site, but this loss is difficult to estimate during balanced min-cut partitioning. Therefore, a traditional min-cut placer that assigns cells to bins based only on site area, may create cell overlaps in such cases. When wide cells get assigned to narrow columns, they may end up overlapping with macros. Since such overlaps are relatively rare, they can be resolved by simple legalization with minimal movement, e.g., Cadence Qplace in the ECO mode. One can also identify contiguous site sequences (*sub-rows*) that are shorter than existing wide cells and mark them as used when creating a new placement bin.

4 Empirical Validation

In earlier sections we demonstrate the effectiveness of our proposed floorplacer in large-scale congestion-driven standard cell placement and free-shape floorplacement. Below we validate our tool on designs with hard blocks and on mixed-size placement instances.

4.1 Results on Floorplanning Instances

Table 5 compares our proposed floorplacer with the annealing-based tool Parquet using GSRC floorplanning benchmarks [12]. Comparisons of other floorplanners to Parquet can be found

Circuit	# Nodes	# Nets	# IOs	Row-Util %	# Macros	% M Area
DMA	11734	13256	948	95.43	0	0
DSP1	26299	28447	844	90.66	2	21.98
RISC1	32615	34034	627	93.94	7	41.99
DSP2	26279	28431	844	90.05	2	6.96
RISC2	32615	34034	627	94.09	7	37.31

Table 6: Faraday benchmarks [4] synthesized and laid out with a standard ASIC flow using IBM Artisan 0.13μm libraries. *%M Area* represents the area of embedded memories in percent of the total cell area. Sample placements of these benchmarks by Capo 9.2 can be found in Figure 9.

Circuit	SEUltra - Qplace(v5.4.126)					Capo 9.2 -feedback					FengShui 2.6 06/17/04				
	Place		Route			Place		Route			Place		Route		
	HPWL (e8)	Time (min)	WL (e8)	Time (min)	V	HPWL (e8)	Time (min)	WL (e8)	Time (min)	V	HPWL (e8)	Time (min)	WL (e8)	Time (min)	V
DMA	4.79	1	6.37	3	0	**4.41**	2	**5.74**	3	0	4.60	6	6.33	3	0
DSP1	10.54	5	12.77	5	0	**9.82**	24	**11.76**	5	1	10.75	14	14.17	8	0
RISC1	16.72	7	21.69	11	3	**15.75**	21	**21.50**	16	0	19.98	30	OC	OC	OC
DSP2	9.98	4	12.09	6	0	**9.23**	9	**11.12**	5	0	9.28	10	11.66	6	0
RISC2	**15.63**	8	**20.74**	30	333	16.30	19	21.38	11	5	209.8	25	OC	OC	OC

Table 7: Routing results on the Faraday benchmarks [4]. Routed WL is in database units. *V* stands for routing violations. *OC* indicates that many cells and macros were placed outside the core area. Routing was performed by Cadence WarpRoute in all cases. All routing as well as Qplace runs are performed on a 750MHz Sun Blade workstation with 2GB RAM running Solaris. Capo and FengShui runs are on a somewhat faster 2.4GHz Linux workstation with 1GB RAM.

in recent literature on floorplanning. We first convert the benchmarks to the GSRC bookshelf format for *placement* using an internal converter and generate square fixed-die layouts with 20% whitespace. Since area minimization is not an objective as long as we fit within the fixed-outline constraints, we only report half-perimeter wirelength (HPWL) and runtimes. For the smallest three benchmarks `n10`, `n30` and `n50` the two approaches perform similarly, as the floorplacer relies on floorplanning. However, the larger the designs, the more partitioning calls are made by the floorplacer. This results in faster and more powerful interconnect optimization compared to the annealing-based Parquet tool. The improvements should be even more pronounced for larger block-based designs.

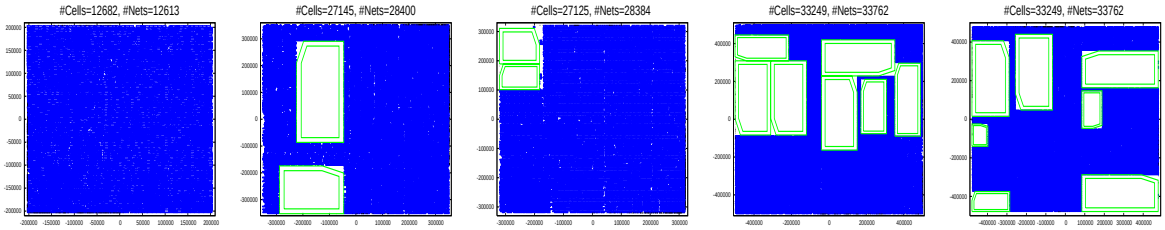
4.2 Validation in Mixed-Size Placement

Faraday Benchmarks. To validate the routability of placements produced by Capo 9.2, we use the new Faraday mixed-size benchmarks introduced in [4]. Characteristics of these benchmarks are listed in Table 6. We compare our approach with Cadence Qplace (part of SEUltra) and FengShui 2.6, using Cadence WarpRoute for routing in all cases. The results are presented in Table 7. FengShui 2.6 was used rather than 5.0 because of observed crashes on Faraday benchmarks. For SEUltra, we use the Cadence-recommended flow for placing mixed-size designs. The placements produced by Capo are generally routable on all benchmarks, sometimes with a small number of violations. For the Capo results in Table 7 legalization by Qplace ECO was not needed, but may be necessary rarely. Sample Capo placements are depicted in Figure 9. FengShui 2.6 produces legal placements of benchmarks DMA, DSP1 and DSP2, but places many cells in RISC1 and RISC2 outside the core area as shown in Figure 10. Only with considerable effort did Qplace ECO legalize these placements, but WarpRoute was unable to complete.

IBM Netlists. The IBM Mixed-Size (`IBM-MS`) placement benchmarks released at ISPD 2002 [2] are derived from the well-known netlists made public by IBM in 1998. These benchmarks have been consistently used in the recent literature on mixed-size placement, but have two important drawbacks: (i) all large modules are square, (ii) all pins in such modules are in the center. Therefore these benchmarks give no incentive to optimize block orientations and cannot be extended with routing information. To this end, the majority of published mixed-size placers do not attempt to optimize module orientations. While the `IBM-MS` benchmarks served well to compare entry-level mixed-size placers, we seek more realistic evaluation.

In [4], a new set of benchmarks derived from the `IBM-MS` placement benchmarks called `IBM-MSwPins` were introduced and are available in the public domain [5]. Aspect ratios of large modules were chosen randomly between 0.5 and 2.0 and pins of all cells and large modules were distributed evenly through the periphery [4]. To determine pin locations for individual cells and large modules, placement was performed with all pins centered. For every net, the center of the net was determined by averaging the locations of incident cells. Lastly, for each cell and large module, pins were ordered on the periphery by the centers of their incident nets[4].[2]

[2] Although benchmark construction is not within the scope of this paper, it has been verified that these benchmarks do not heavily favor one placement algorithm over another. The benchmarks have been randomized and the same trends as



(a) DMA (b) DSP1 (c) DSP2 (d) RISC1 (e) RISC2

Figure 9: The Faraday benchmarks placed by the Capo 9.2 floorplacer. Note that Capo tends to align large blocks, which may simplify the routing in their vicinity, as well as the routing of busses connecting those blocks. To show block orientations, north-west corners of memories are removed.

We compare our proposed floorplacement approach to Cadence Qplace (part of SEUltra), a Capo-Parquet-Capo methodology [1], Capo followed by an incremental run of Kraftwerk (data from [2]), mPG-MS [16] and FengShui 2.6 [25] using the two sets of IBM mixed-size benchmarks. Relative performance is reported in Table 8. Detailed results for the newer `IBM-MSwPins` benchmarks are presented in Table 9. Given that some tools are only available on the Sun Solaris platform and others only on Intel-compatible Linux workstations, runtimes are not directly comparable. However, we list the hardware platform for each tool. For SEUltra, we use the Cadence-recommended flow for mixed-size designs, which produces completely legal placements, unlike those reported in [2] for the 2002 version of the same tool. Also note that the wirelengths achieved by the latest version of SEUltra are much better than those reported in [2]. Clearly, Cadence tools have greatly improved since 2002.

Benchmark Suite	SEUltra v5.1.67 (2002)	SEUltra v5.4.126 (2004)	Capo+ Parquet+ Capo[2]	Capo+ Kraftwerk ECO[2]	mPG [16]	FengShui[25] v2.6 06/17/04	Capo v9.2 -feedback	Capo v9.2 -feedback best-of-2
IBM-MS (ISPD02)	100.02%	16.52%	24.37%	19.08%	18.47%	-4.47%	0%	-1.03%
IBM-MSwPins (ICCAD04)	-	20.66%	26.80%	24.97%	-	0.56%	0%	-1.23%

Table 8: Averaged placement results on common mixed-size benchmarks. The `IBM-MSwPins` benchmarks have non-trivial macro aspect ratios and pins spread around the periphery of cells and macros [4]. A positive percentage indicates a loss relative to Capo 9.2.

For placements produced by our floorplacer Capo 9.2, Capo was run with the -feedback option (which is several times slower than default) and used the HMetis partitioner [23] in place of Capo's

noted here in experimental results have been observed.

Circuit	Cadence SEUltra Block-Place+QPlace Sun-Blade1000 750MHz I		Capo+Parquet+Capo [2] (low-temp. annealing) Linux/Pentium 2GHz II		Capo+Kraftwerk ECO [2] Linux/Pentium 2GHz III			FengShui v2.6 06/17/04 Linux/Pentium 2.4GHz IV		Capo v9.2 -feedback Linux/Pentium 2.4GHz V	
	HPWL (e6)	Time (min)	HPWL (e6)	Time (min)	HPWL (e6)	Time (min)	% Overlap	HPWL (e6)	Time (min)	HPWL (e6)	Time (min)
ibm01	3.25	12	3.23	18	2.96	5	1.22	2.56	3	2.56	6
ibm02	7.17	31	7.91	12	6.84	13	0.25	6.05	5	5.20	10
ibm03	9.06	28	10.08	57	9.45	13	0.18	8.77	6	8.47	15
ibm04	10.28	31	11.01	12	10.09	15	0.74	8.38	7	8.56	15
ibm05	11.55	24	11.03	5	11.46	5	0	9.94	8	10.27	13
ibm06	8.33	32	8.70	19	9.22	19	0.25	6.99	9	6.93	19
ibm07	13.79	41	14.34	22	14.34	57	0.24	11.37	12	11.98	26
ibm08	17.36	50	17.01	26	17.63	22	1.80	13.51	15	13.81	32
ibm09	[16.91]	56	19.53	29	21.04	32	0.35	14.12	14	14.61	31
ibm10	43.71	86	53.34	119	49.52	72	4.34	41.96	22	33.13	52
ibm11	24.98	71	25.51	43	25.48	42	0.76	21.19	21	21.65	45
ibm12	46.38	87	54.82	97	61.48	53	0.63	40.84	22	38.81	58
ibm13	33.06	91	34.30	54	32.37	73	0.12	25.45	25	27.33	59
ibm14	[45.74]	148	48.66	145	47.63	117	0.07	39.93	52	39.86	110
ibm15	68.63	206	70.68	208	62.63	124	0.09	51.96	67	57.96	135
ibm16	75.94	248	75.27	154	78.47	166	2.03	62.77	70	62.65	152
ibm17	92.41	288	87.81	204	85.40	132	0.13	69.38	79	73.43	173
ibm18	57.04	190	54.66	115	57.47	162	0.02	45.59	87	46.55	154
Avg	20.66%		26.80%		24.97%			0.56%		0%	

Table 9: Mixed-size placement results on the `IBM-MSwPins` mixed-size benchmarks. A positive percentage in the last row indicates a loss to Capo 9.2. Cadence SEUltra places designs ibm09 and ibm14 illegally with overlaps between macros or macros outside the core area.

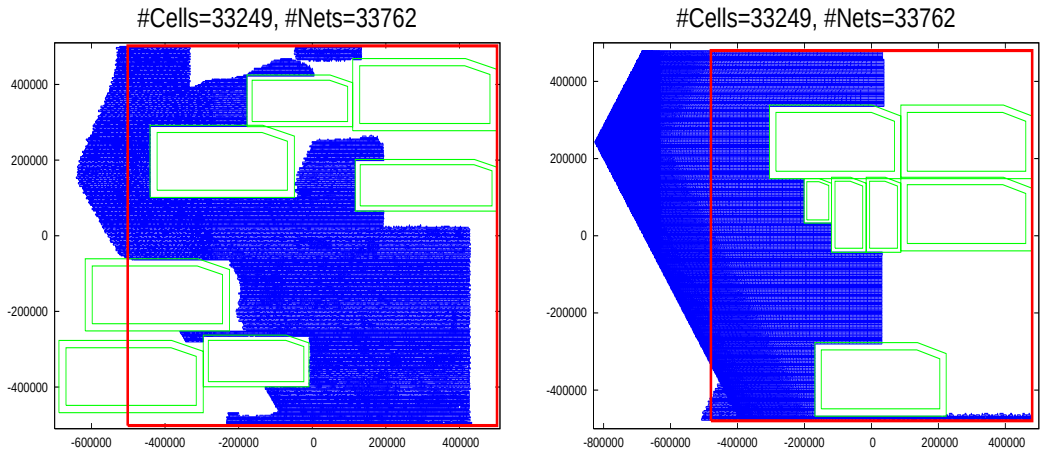
MLPart partitioner. Currently, using HMetis instead of MLPart improves Capo's results by 2% on average. On the older `IBM-MS` benchmarks, Capo 9.2 placements, in terms of HPWL, are on average *16.52%* better than Cadence SEUltra, *24.37%* better than the Capo-Parquet-Capo flow, *19.08%* better than Capo-Kraftwerk ECO flow, *18.47%* better than mPG-MS and *4.47%* worse than FengShui 2.6. FengShui 2.6 was used rather than 5.0 because of observed crashes on Faraday benchmarks. Using the best of two runs of Capo 9.2 improves solution quality by *1.03%*. On the newer `IBM-MSwPins` benchmarks, on average, the placements produced by our floorplacer are *20.66%* better than Cadence SEUltra, *26.80%* better than the Capo-Parquet-Capo flow, *24.97%* better than Capo-Kraftwerk ECO flow and *0.56%* better than FengShui 2.6. Choosing the best of two Capo 9.2 runs results in a *1.23%* improvement. Note that FengShui shifts all cells to the left (or right) edge of the chip, thus lowering wirelength compared to a placement spread around the core area. However, according to Table 7, this strategy is not always successful in the presence of large modules. Comparing results of FengShui 2.6 on two sets of benchmarks in Table 8, we conclude that the relative advantage of FengShui 2.6 decreases in the presence of rectangular

blocks with non-trivial pin offsets, as it does not optimize module orientations.

5 Conclusions

Our work originates from the realization that min-cut placers implicitly perform floorplanning, in addition to partitioning. Therefore, separate partitioning and floorplanning steps traditionally used in VLSI design can be subsumed by a min-cut placer. Such a unification can lead to simpler, more consistent, more controllable and more successful EDA tools and tool chains. For example, while the field of floorplanning has been very active in academia for twenty years, there are relatively few successful commercial floorplanners. While this is partly due to integration difficulties and to the fact that experienced designers perform floorplanning by hand, our results suggest that common floorplanners based purely on Simulated Annealing tend to produce very sub-optimal solutions. To a large extent this is not a matter of EDA tools' lacking intangible designer intuition, but rather the poor quality of existing algorithms with respect to closed-form optimization objectives. Interconnect optimization is also handicapped by the popular limitation that all modules be laid out as rectangles. To this end, our work shows that unifying partitioning, floorplanning and placement in a single algorithm leads to better layouts and facilitates new layout optimizations, such as free-shape floorplanning that simultaneously determines the locations and shapes of modules so as to optimize interconnect. Empirical validation uses a unified *floorplacer* tool, that can be used as a partitioner, a large-scale cell placer, a floorplanner and a mixed-size placer. Our implementation scales well, is competitive with the state of the art in all of its areas of applicability, and in some cases produces better wirelengths than any previously reported methods.

We show that for sufficiently large floorplanning and mixed-size placement instances, min-cut techniques are more successful in minimizing wirelength than simulated annealing. However, for small layout instances with modules of different sizes, the use of annealing seems required to pack modules well. In the process of tuning the performance of our implementation, we empirically tabulate the boundary between placement and floorplanning by identifying more successful optimizations in various cases. A representative threshold for floorplanning is currently at 30 blocks, which means that the use of flat annealing on larger instances is not justified. In the future, as floorplanners improve at satisfying fixed-outline constraints while minimizing wirelength, this



(a) RISC1 (b) RISC2

Figure 10: The Faraday benchmarks RISC1 and RISC2 placed by FengShui 2.6. All large modules have default orientations. FengShui places many standard cells beyond the left boundary of core region (outlined in red). FengShui 2.5 exhibits similar behavior on DSP1 and DSP2 benchmarks, which the authors attribute to bugs fixed in FengShui 2.6.

boundary can be lowered even further.

A floorplacer of the type described in our work can place objects with very different semantics — standard cells, macros, datapaths, memories, etc. Extensions to free-shape floorplanning can be used with unsynthesized modules to better estimate global interconnect delays before synthesis. However, to fully exploit these novel capabilities, new VLSI methodologies are required. Our hope is that such future methodologies and methodology studies will confirm the potential of floorplacement.

References

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

[2] S.N. Adya and I.L. Markov, “Combinatorial Techniques for Mixed-size Placement”, *ACM Trans. on Design Autom. of Elec. Sys.*, vol. 10, no. 5, January 2005. Also see *ISPD 2002*, pp. 12-17.

[3] S.N. Adya and I.L. Markov,“Fixed-outline Floorplanning: Enabling Hierarchical Design”, *IEEE Trans. on VLSI* 11(6), pp. 1120-1135, December 2003. Also see *ICCD 2001*, pp. 328-334.

[4] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov, “Unification of Partitioning, Floorplanning and Placement”, *Intl. Conf. Computer-Aided Design* (ICCAD), San Jose, CA, November 2004, pp. 550-557.

[5] S.N. Adya, S. Chaturvedi and I.L. Markov,
“ICCAD’04 Mixed-size Placement Benchmarks”, in *GSRC Bookshelf*,
`http://vlsicad.eecs.umich.edu/BK/ICCAD04bench`

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

[7] A. Agnihotri et al., “Fractional Cut: Improved recursive bisection placement”,
ICCAD, 2003, pp. 307-310.

[8] U. Brenner and A. Rohe, “An effective congestion driven placement framework”, *ISPD*, 2002, pp. 6-11.

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

[10] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, “On Wirelength Estimations for Row-Based Placement”, *IEEE Trans. on CAD* 18(9), (1999), pp. 1265-1278.

[11] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?” *DAC* 2000, pp. 477-82.

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

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

[14] M. R. Casu and L. Macchiarulo, “Floorplanning for Throughput”, *ISPD* 2004, pp. 62-69.

[15] H. H. Chan, S. N. Adya and I. L. Markov, “Are Floorplan Representations Useful in Digital Design?”, to appear in *Proc. Intl. Symposium on Physical Design (ISPD)*, San Fransisco, April 2005.

[16] C.-C. Chang, J. Cong, and X. Yuan, “Multi-level Placement for Large-Scale Mixed-Size IC Designs,” *ASPDAC* 2003, pp. 325-330.

[17] J. Cong et al., “Microarchitecture Evaluation With Physical Planning” *DAC*, 2003, pp. 32-35.

[18] H. Eisenmann and F. M. Johannes, “Generic Global Placement and Floorplanning”, *DAC* 1988, pp. 269-274.

[19] Y. Feng, D. P. Mehta and H. Yang, “Constrained Floorplanning Using Network Flows,” *IEEE Trans. on CAD*, April 2004.

[20] A. B. Kahng, “Classical Floorplanning Harmful?”, *ISPD* 2000, pp. 207-213.

[21] A. B. Kahng, S. Mantik and I. L. Markov, “Min-max Placement For Large-scale Timing Optimization”, in *Proc. ACM/IEEE Intl. Symp. on Physical Design (ISPD)*, pp. 143-148, April 2002.

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

[23] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, “Multilevel Hypergraph Partitioning: Applications in VLSI Domain”, *DAC* 1997, pp. 526-629.

[24] M. Z.-W. Kang and W. W.-M. Dai, “Topology Constrained Rectilinear Block Packing For Layout Reuse”, *ISPD 1998*, pp. 179-186.

[25] A. Khatkhate et al., “Recursive Bisection Based Mixed Block Placement”, *ISPD* 2004, pp. 84-89.

[26] N. Magen, A. Kolodny, U. Weiser and N. Shamir, “Interconnect Power Dissipation in a Microprocessor”, *SLIP* 2004, pp. 7-13.

[27] A. Ranjan, K. Bazargan and M. Sarrafzadeh, “Floorplanner 1000 Times Faster: A Good Predictor and Constructor”, in *System-Level Interconnection Prediction (SLIP)*, pp. 115-120, 1999.

[28] A. Ranjan, K. Bazargan and M. Sarrafzadeh, “Fast Hierarchical Floorplanning with Congestion and Timing Control”, *IEEE International Conference on Computer Design (ICCD)*, pp. 357-362, September 2000.

[29] D. Keitel-Schulz and N. Wehn, “Embedded DRAM development: Technology, physical design, and application issues”, *IEEE Design & Test* 18(3), pp. 7-15, May 2001.

[30] D. Sherlekar, “Design Considerations for Regular Fabrics,” *ISPD* 2004, pp. 97-102.

[31] J. Vygen, “Algorithms for Large-Scale Flat Placement”, *DAC* 1997, pp. 746-751.

[32] E. Wein and J. Benkoski, “ Hard macros will revolutionize SoC design”, *EE Times*, August 20, 2004. `http://www.eetimes.com/news/design/showArticle.jhtml?articleID=26807055`

[33] J. Xu, P.-N. Guo and C.-K. Cheng, “Rectilinear Block Placement Using Sequence Pair”, *ISPD* 1998, pp. 173-178.

[34] H. Yu, X. Hong and Y. Cai “MMP: A Novel Placement Algorithm for Combined Macro-Block and Standard Cell Layout Design” *ASPDAC*, 2000, pp. 271-276.

[35] X. Yang, B.-K. Choi and M. Sarrafzadeh, “Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement”, *IEEE Trans. on CAD* 22 (4), pp. 410-419, April 2003.