Unification of Partitioning, Placement and Floorplanning

Saurabh N. Adya
Synplicity Inc.
600 W. California Ave.
Sunnyvale, CA 95054
saurabh@synplicity.com

Shubhyant Chaturvedi
Advanced Micro Devices
Austin, TX, 78741
shubhyant.chaturvedi@amd.com

Jarrod A. Roy, David A. Papa, Igor L. Markov
University of Michigan
EECS Department
Ann Arbor, MI 48109-2122
{royj,iamyou,imarkov}@umich.edu

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, and improvements by over 10% per paper are still common. 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 synthesis.

1. INTRODUCTION

The amount of embedded memory used on a chip is expected to grow dramatically in the next few years [24], from around 50% of the die area today to 70% by 2005, and 90% by 2011. This growth is mostly fueled by chips for high-bandwidth communication, portable multi-media, 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 [24]. 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 challenging than conventional standard-cell layout. While commercial layout tools have considerably improved in the last two years, the locations of large blocks are still typically determined by manual floorplanning. Perhaps, the most obvious challenge is the minimization of wirelength, which also affects routability. The optimization of wirelength is the most prevalent approach to placement and floorplanning, and it enables other optimizations through the use of net weights and bounds [12, 14]. Moreover, some form of wirelength optimization appears necessary — a recent study [23] from Intel shows that 51% of dynamic power in currently-shipped microprocessors is consumed when driving signals over interconnects, including local and global wires.

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 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 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 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 implementation Capo 9.0 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 so as 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 is completely new and is the first to incorporate 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

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
Can handle 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.

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 [6, 22] which relies on greedy legalization of cell macro locations through left (or right) packing. Such strategies typically produce unroutable standard-cell placements [29, 5], and careful re-distribution of whitespace shown in [29] to improve routability may be less effective with large circuit blocks present, due to the fragmentation of layout. More 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. The Appendix introduces new mixed-size placement benchmarks which are used for empirical validation in Section 4. Section 5 concludes our paper.

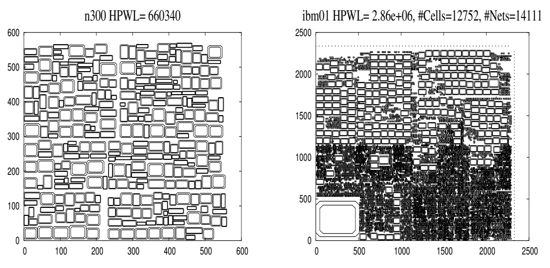
2. RELEVANT PREVIOUS WORK

As pointed out in [18, 9, 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 [9]. 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 [9], 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 [11] 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* [19] 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 [29] in a mode where it spreads whitespace according to congestion. This significantly increases wirelength, but produces more routable placements. As of August 2004, FengShui [22] does not have such a mode and shifts all cells to the left (or right), typically yielding unroutable placements.



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

Figure 1: Layout styles. Standard-cell layout is shown in Figure 4(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 [18], which accentuates the minimization of wirelength, reminding of placement.

The floorplanner Parquet [3, 10] performs fixed-outline floorplanning with rectangular modules by combining Simulated Annealing with a new mechanism for move selection, based on the notion of *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 necessarily relocate a block with zero horizontal slack (similarly for height). Such moves are performed at regular time intervals during conventional Simulated Annealing to bias the aspect ratio of current floorplan to that of the desired outline. When the temperature schedule runs out, the final

Circuit	Capo 9.0 -feedback		Dragon 3.01 -fd		FengShui 2.5	
	routed WL	Viol	routed WL	Viol	routed WL	Viol
ibm01e	839802	0	871052	53	time-out	1351
ibm01h	860067	147	832928	0	time-out	1736
ibm02e	2239345	0	2198366	200	2202910	0
ibm02h	2162938	0	2215116	0	time-out	1722
ibm07e	4620754	0	4249798	0	time-out	85
ibm07h	4861456	25	4643654	0	time-out	649
ibm08e	4750574	0	4681110	0	4609964	0
ibm08h	4882005	0	4530017	0	time-out	133116

Table 2: Routing results on IBM-Dragon V2 benchmarks with a 24-hour time-out. Cadence WarpRoute typically completes on placements produced by Dragon and Capo, sometimes with a small number of violations. WarpRoute often fails on FengShui placements.

floorplan may not satisfy the outline. Parquet empirically achieves high ratios of successes to failures on fixed-outline floorplanning instances with 15% whitespace [3]. Parquet also supports soft blocks.

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, 13, 22, 28], and we survey relevant algorithms below.

The force-directed algorithm Kraftwerk [15] 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 [5] Kraftwerk was outperformed by several min-cut tools. Another potential shortcoming of this analytical algorithm is having no provisions for optimizing orientations of large modules — a clearly discrete optimization problem.

MMP [28] 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 cutline after each min-cut optimization. As described, the algorithm assumes pre-determined orientations of 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 placements 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 [13] 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 [22] advocates a two-stage approach to mixed-size placement. First, the min-cut placer FengShui [6] 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 [6], 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 [22] (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 [22] 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 [22] contain numerous modules of medium size). Another concern about packed placements is the harmful effect of such a strategy on routability, explicitly shown in [29]. Overall, the work in [22] 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.

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

```
 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)
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        Floorplan new merged bin
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 2: Our floorplacement algorithm. Bold-faced lines 3-10 are different from traditional min-cut placement.

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 is split into two bins of similar sizes, and during this process the cut-line is adjusted according to actial 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 [8]. 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 [22] 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, congestion and delay estimates that may have been performed early during top-down placement [7], and (2) avoid the need to legalize a placement with overlapping macros. In particular, we are unconvinced that existing legalization algorithms are robust enough to handle a wide variety of module shapes and sizes in realistic netlists (see Figure 5). We also anticipate difficulty ensuring routability while shifting macros and standard cells at the same time.

While resorting 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 2.

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 [26]. 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 [20]. 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], because standard cells will be placed later anyway. 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, the locations of all large modules are returned to the top-down placer and 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 using a row-based legalizer from the GSRC bookshelf [10]. Detailed placement uses branch-and-bound placement in sliding windows [8], but does not move the macros. Figure 1(b) shows a sample placement produced by our tool.

3.2 Practical Issues

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 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 blocks in bins (that trigger floorplanning) are deliberately conservative.

- At least one module does not fit into a potential child bin.
- The sum of the larger dimension of the largest module and the smaller dimension of the second largest module exceeds the smaller dimension of a potential child bin.
- There are ≤ 30 large modules in the bin, but their total area exceeds 80% of the total area of cells and modules in the bin.

In our experience, these tests are good enough to ensure that at most one level of backtracking (block-merging) is required to prevent overlaps between large modules.

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

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

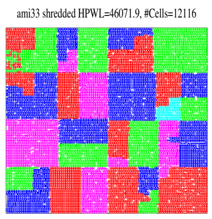
wide cells get assigned to narrow columns, they may end up overlapping with macros. Since such overlaps are relatively rare, they can be resolved by simple legalization with minimal movement, e.g., Cadence Qplace in the ECO mode. One can also identify contiguous site sequences (*sub-rows*) that are shorter than existing wide cells and mark them as used when creating a new placement bin.

3.3 Wirelength-driven Floorplanning

Pure block-based designs. 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 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 1(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 recusrive bisection 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.

Free-shape rectilinear floorplanning. Some circuit modules, such as embedded memories and pre-designed datapaths, have fixed rectangular shapes. However, when only the area of a module is estimated, but its shape is unknown, there is often no *a priori* reason to limit its shape to rectangles. Such limitations may be justified by added efficiency in handling rectangular blocks, but can handicap interconnect optimization. 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 [21] and [27] 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 [17] solves a specific floorplanning formulation proposed in [18], 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 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.

Below 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 3(a) where modules are color-coded. The plot is



Type→	Rectangular	Free-shape	
Circuit ↓	Parquet 2.0 HPWL	Capo 9.0 HPWL	Avg% Impr
ami33	76987	46072	40.1
ami49	895560	469476	47.5
n50	202240	87957	56.5
n100	350593	157548	55.0

Figure 3: Figure on left shows 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.

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, everything is accomplished without Simulated Annealing. Figure 3(b) reports the improvement in runtime and wirelength over traditional 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.

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 3 compares our proposed floorplacer with the annealing-based tool Parquet using GSRC floorplanning benchmarks [10]. Comparisons of other floorplanners to Parquet can be found 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 resorts to 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 9.0, we use the new benchmarks introduced in the Appendix. We compare our approach with Cadence Qplace (part

Circuit	#Blocks	Parquet		Capo (Mixed-size)		
		HPWL	Time sec	HPWL	Time sec	# Min-cut Levels
n10	10	5.58	0.27	5.57	0.37	0
n30	30	17.38	2.35	16.93	1.89	**1**
n50	50	20.77	8.16	20.34	5.3	**1**
n100	100	34.53	50.12	32.39	**10.5**	**2**
n200	200	62.28	240.61	**56.82**	**27.42**	**3**
n300	300	75.69	433.92	**63.62**	**25.21**	**3**

Table 3: Floorplannning 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.

Circuit	SEUltra - Qplace(v5.4.126)					Capo9.0 -feedback					FengShui 2.6 06/17/04				
	Place		Route			Place		Route			Place		Route		
	HPWL (e8)	Time (min)	WL (e8)	Time (min)	Viol	HPWL (e8)	Time (min)	WL (e8)	Time (min)	Viol	HPWL (e8)	Time (min)	WL (e8)	Time (min)	Viol
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/OC	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/OC	25	OC	OC	OC

Table 4: Routing results on Faraday benchmarks. Routed WL is in database units. *OC* means that a large number of cells and macros were placed outside the core area. Best results are bold-faced. 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. Capo is used in the -feedback mode, which is several times slower than the default mode. Also note that Capo performs local annealing-based floorplanning.

of SEUltra) and FengShui 2.6, using Cadence WarpRoute for routing in all cases. The results are presented in Table 4. For SEUltra, we use the Cadence-recommended flow for placing mixed-size designs as explained in the Appendix. The placements produced by Capo are generally routable on all benchmarks, sometimes with a small number of violations. For the Capo results in Table 5 legalization by Qplace ECO was not needed, but may be necessary rarely. 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 5. Only with considerable effort Qplace ECO legalized these placements, but WarpRoute did not 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.

We derive a new set of benchmarks termed `IBM-MSwPins` from the `IBM-MS` placement benchmarks. Aspect ratios of large modules are chosen randomly between 0.5 and 2.0. Pins of all cells and large modules are distributed evenly through the periphery. To determine pin locations for individual cells and large modules, we first perform placement with all pins centered. For every net, we determine its center by averaging the locations of incident cells. Then, for each cell and large module, pins are ordered on the periphery by the centers of their incident nets. The new `IBM-MSwPins` benchmarks are available in the public domain [4].

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 [13] and FengShui 2.6 [22] using the two sets of IBM mixed-size benchmarks. Relative performance is reported in Table 6. Detailed results for the newer `IBM-MSwPins` benchmarks are presented in Table 7. 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.

On the older `IBM-MS` benchmarks, placements produced by our floorplacer Capo 9.0 (with option -feedback) are on average *12.09%* better than Cadence SEUltra, *19.61%* better than the C-P-C flow, *14.56%* better than Capo-Kraftwerk ECO flow, *13.99%* better than mPG-MS and *8.09%* worse than FengShui 2.6. Using the best of two runs of Capo 9.0 improves solution quality by *1.66%*. On the newer `IBM-MSwPins` benchmarks, in terms of HPWL, on average, the placements produced by our floorplacer are *13.74%* better than Cadence SEUltra, *19.59%* better than the Capo-Parquet-Capo flow, *17.83%* better than Capo-Kraftwerk ECO flow and *5.14%* worse than FengShui 2.6. Choosing the best of two Capo 9.0 runs results in a *1.55%* 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 2, 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 6, 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.

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 5: Faraday benchmarks 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.

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

Benchmark Suite	SEUltra v5.1.67 (2002)	SEUltra v5.4.126 (2004)	Capo+ Parquet+ Capo[2]	Capo+ Kraftwerk ECO[2]	mPG [13]	FengShui[22] v2.6 06/17/04	Capo v9.0 -feedback	Capo v9.0 -feedback best-of-2
IBM-MS (ISPD 2002)	92.71%	12.09%	19.61%	14.56%	13.99%	-8.09%	0%	-1.66%
IBM-MS wPins (new)	-	13.74%	19.59%	17.83%	-	-5.14%	0%	-1.55%

Table 6: Average mixed-size placement results on two suites of mixed-size benchmarks derived from ISPD-1998 circuit netlists. IBM-MS is the original suite of benchmarks released in ISPD-02. `IBM-MSwPins` **is the new suite of benchmarks with non-trivial macro aspect ratios and pins spread around the periphery of all cells and macros. A positive percentage indicates an approach produces placements with that much greater HPWL than Capo 9.0 on average. FengShui 2.6 placements are packed to an edge of the core, and in practical applications they may have to be spread to ensure routability. Since Capo placements are already spread, the difference in wirelengths will be reduced.**

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 V		Capo v9.0 -feedback Linux/Pentium,2.4GHz VI	
	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	2.67	4
ibm02	7.17	31	7.91	12	6.84	13	0.25	6.05	5	5.54	9
ibm03	9.06	28	10.08	57	9.45	13	0.18	8.77	6	8.67	13
ibm04	10.28	31	11.01	12	10.09	15	0.74	8.38	7	9.79	18
ibm05	11.55	24	11.03	5	11.46	5	0	9.94	8	10.82	8
ibm06	8.33	32	8.70	19	9.22	19	0.25	6.99	9	7.35	12
ibm07	13.79	41	14.34	22	14.34	57	0.24	11.37	12	12.30	25
ibm08	17.36	50	17.01	26	17.63	22	1.80	13.51	15	16.02	36
ibm09	16.91	56	19.53	29	21.04	32	0.35	14.12	14	15.51	31
ibm10	43.71	86	53.34	119	49.52	72	4.34	41.96	22	34.98	59
ibm11	24.98	71	25.51	43	25.48	42	0.76	21.19	21	22.31	36
ibm12	46.38	87	54.82	97	61.48	53	0.63	40.84	22	40.78	42
ibm13	33.06	91	34.30	54	32.37	73	0.12	25.45	25	28.70	65
ibm14	45.74	148	48.66	145	47.63	117	0.07	39.93	52	40.97	71
ibm15	68.63	206	70.68	208	62.63	124	0.09	51.96	67	59.19	116
ibm16	75.94	248	75.27	154	78.47	166	2.03	62.77	70	67.00	115
ibm17	92.41	288	87.81	204	85.40	132	0.13	69.38	79	78.78	94
ibm18	57.04	190	54.66	115	57.47	162	0.02	45.59	87	50.39	85
Avg	13.74%		19.59%		17.83%			-5.14%		0%	

Table 7: Mixed-size placement results on the new `IBM-MSwPins` **mixed-size benchmarks. A positive percentage in the last row indicates an approach produces placements with that much greater HPWL than Capo 9.0 on average. Cadence SEUltra places designs ibm09 and ibm14 illegally with overlaps between macros or macros outside the core area. FengShui 2.6 returns core placements packed to core edges.**

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 boundary can be lowered even further.

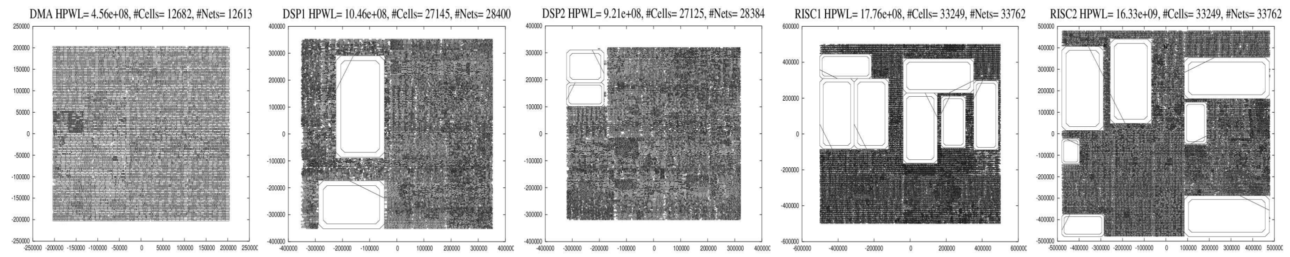
A floorplacer of the type described in out work can place objects with very different semantics — standard cells, macros, datapaths, memories, etc. Extensions to free-shape floorplanning can be used with unsynthesized modules to better estimate global interconnect delays before synthesis. However, to 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.

Appendix: Embedded-Memory Benchmarks

The Faraday Corporation recently released three circuits [16], originally intended for comparisons between structured and conventional ASICs. We apply to these benchmarks a standard ASIC design flow to generate five mixed-size designs. Faraday benchmarks include three commonly-used functional blocks: **(I) a 16-bit DSP**, **(II) a 32-bit RISC CPU**, and **(III) a DMA controller** — see Table 5 for statistics. Other details on these benchmarks, such as the EDA tools recommended by Faraday and tool settings, can be found in [16]. To minimize the impact of routing on the results of the accounted placement approaches, we avoid clock-tree generation and power routing in our flows. However, both clock-trees and power rails can be built on our benchmarks. Below we describe our ASIC flow for generating the mixed-size benchmarks from the original netlists.

Faraday benchmarks come with behavioral Verilog descriptions, timing constraints and scripts for synthesis. We use Artisan's 0.13 micron libraries in IBM technology for synthesizing these designs under worst-case process conditions, with the same timing constraints as specified in the Faraday design documents. Synopsys Design Compiler (v2003.03-2) is used for synthesis, and Artisan Memory Generator to instantiate embedded memories. Artisan limits the size of its SRAM memories to a minimum word-length of 128 for dual-port memories and to 256 for single-port memories. This requires a change in the behavioral descriptions of Faraday designs to account for larger word-lengths. In a variation of the original design, we built register files in place of memories for smaller word-lengths and thus came up with two flavors each for DSP and RISC — one uses only memories and the other uses both memories and register files.

The gate-level netlists obtained after synthesis are taken through the automatic place and route (APR) flow using Cadence Silicon Ensemble Ultra (v5.4.126). We follow the Cadence recommended flow for placing mixed-size designs and first use the "Qplace No-



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

Figure 4: Faraday benchmarks placed by the Capo 9.0 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. The discrepancies in wirelength versus Table 4 for the same benchmarks represent variability in Capo results. To show block orientations, north-west corners of memories are marked with diagonal lines.

config Block" command to place embedded memories. Then the locations of embedded memories are fixed, the affected cell sites are removed, and the remaining standard cells are placed using the command "Qplace Noconfig". To find good locations of I/O pads, we perform concurrent pin and cell placement in Qplace. This improves routability compared to a random I/O placement during floorplanning. When routing with Cadence WarpRoute, we found that Artisan memory pins are not aligned to the same routing grid as pins in random logic. Fixing this required manual intervention. The new Faraday-MS benchmarks are the first mixed-size placement benchmarks in the public domain [4] to provide non-square modules with realistic pin offsets and routing information.

6. 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", to appear in *ACM Trans. on Design Autom. of Elec. Sys.*, 2004. 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 and I.L. Markov, `http://vlsicad.eecs.umich.edu/BK/ICCAD04bench`

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

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

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

[8] 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.

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

[10] A. E. Caldwell, A. B. Kahng, I. L. Markov, "VLSI CAD Bookshelf" `http://vlsicad.eecs.umich.edu/BK`

[11] 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.

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

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

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

[15] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *DAC* 1988, p. 269-274.

[16] http://www.faraday-tech.com/StructuredASIC/download.html

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

[18] A. B. Kahng, "Classical Floorplanning Harmful?", *ISPD* 2000, pp. 207-213.

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

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

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

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

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

[24] 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.

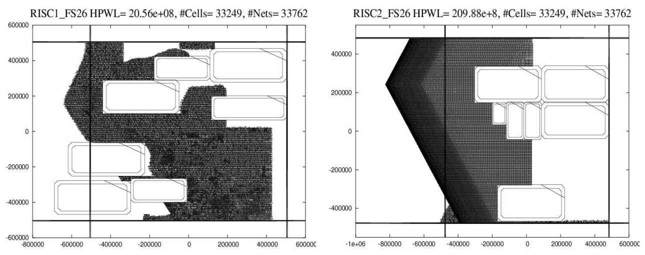
[25] D. Sherlekar, "Design Considerations for Regular Fabrics," *ISPD* 2004, pp. 97-102.

[26] J. Vygen, "Algorithms for Large-Scale Flat Placement", *DAC* 1997, pp. 746-751.

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

[28] 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.

[29] 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.



(a) RISC1 (b) RISC2

Figure 5: Faraday benchmarks RISC1 and RISC2 placed by FengShui 2.6. All large modules have default orientations. FengShui places many standard cells beyond the left boundaries of core regions, shown by thin vertical lines. FengShui 2.5 exhibits similar behavior on DSP1 and DSP2 benchmarks, which the authors attribute to bugs fixed in FengShui 2.6.