Combinatorial Techniques for Mixed-size Placement

S. N. Adya and I. L. Markov
University of Michigan, Ann Arbor

While recent literature on circuit layout addresses large-scale standard-cell placement, the authors typically assume that all macros are fixed. Floorplanning techniques are very good at handling macros, but do not scale to hundreds of thousands of placeable objects. Therefore we combine floorplanning techniques with placement techniques to solve the more general placement problem. Our work shows how to place macros consistently with large numbers of small standard cells. Proposed techniques can also be used to guide circuit designers who prefer to place macros by hand.

We address the computational difficulty of layout problems involving large macros and numerous small logic cells at the same time. Proposed algorithms are evaluated in the context of wirelength minimization because a computational method that is not scalable in optimizing wirelength is unlikely to be successful for more complex objectives (congestion, delay, power, etc.)

We propose several different design flows to place mixed-size placement instances. The first flow relies on an arbitrary black-box standard-cell placer to obtain an initial placement and then removes possible overlaps using a fixed-outline floorplanner. This results in valid placements for macros, which are considered fixed. Remaining standard cells are then placed by another call to the standard-cell placer. In the second flow a standard-cell placer generates an initial placement and a force-directed placer is used in the ECO mode to generate an overlap-free placement. Empirical evaluations on *ibm* benchmarks show that in most cases our proposed flows compare favorably with previously published mixed-size placers Kraftwerk, mixed-size floor-placer proposed at DATE 2003 and are competitive with mPG-MS.

Categories and Subject Descriptors: B.7.2 [**Hardware**]: Integrated Circuits—*Design Aids*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: VLSI, Placement, Floorplanning

1. INTRODUCTION

During the last few decades, academia and industry have invested considerable effort in research on Physical Design for VLSI [Sherwani 1999]. Through the integration of multiple optimization techniques, design methods and high-performance CAD software for integrated circuits (ICs) were developed. However, the growing size of ICs lead to frequent changes to common design flows. Recently, design reuse was introduced as a way to (i)

Author's address: S.N. Adya and I.L. Markov, University of Michigan, EECS Department, 1301 Beal Ave., Ann Arbor, MI 48109-2122. Email {sadya,imarkov}@eecs.umich.edu

tame the complexity of circuit design for deep sub-micron technologies, and (ii) improve time-to-market. This trend is further accelerated with the use of hardware description languages and high-level synthesis. Indeed, several current industrial initiatives provide infrastructure and training for the reuse of Intellectual Property (IP), and also facilitate business models based on IP reuse.

Reuse of design IP is important for multi-million-gate ASICs and considered an integral part of the System-On-Chip (SoC) design style, that is critical for graphics cards, communication chips, etc. Design IP blocks may implement algorithms or signal transforms, and may contain "canned" table look-ups or embedded RAM.

During Physical Design, IP design blocks appear as black-box *macros*, i.e., blocks of logic with known function and a known geometric and electrical properties, but no structural description of their inner workings. Such macros may or may not have flexible geometries, but in any case are considered parts of design. In classical Physical Design flows a circuit is first partitioned, then floorplanned, and finally, standard-cell placement is performed in each partition. This was necessary primarily because older placers, e.g., those based on Simulated Annealing, did not scale very well. However the scalability of min-cut placers dramatically improved after the multi-level partitioning breakthrough in 1997 [Alpert et al. 1997; Karypis et al. 1997; Caldwell et al. 2000a]. In addition to having near-linear runtime, placers based on recursive bisection *perform circuit partitioning* and, if the cut-lines are allowed to move, also *perform floorplanning*. Yet, macro-placement is not supported in these placers, mainly because large macros, that contain more than several percent of layout area, introduce considerable discreteness in the solution space and may be difficult to handle within standard recursive min-cut bisection.

Reusing black-box macros in Physical Design still remains a challenge and existing commercial tools often require help from human designers. For example, the Cadence QPlace manual [Cadence] mentions that the addition of macros may slow down otherwise fairly efficient placement of standard cells and the results may be inferior to what human designers can achieve. Cadence Silicon Ensemble (SEDSM) recommends the following flow for circuits with macros.

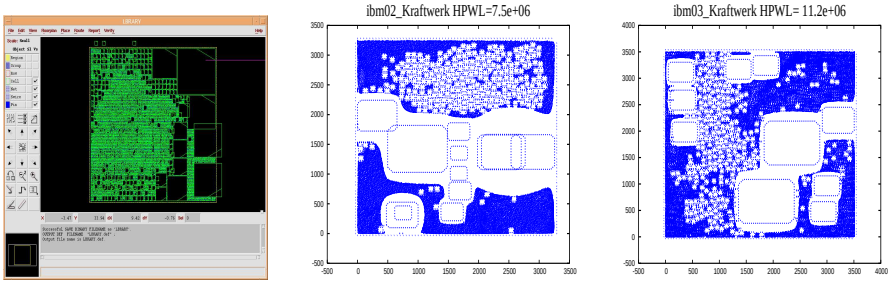
— Do block placement to place macros. Macros may have overlaps and may not fit in layout area.

— Human designer manually removes any overlaps between macros.

— Macros are now considered fixed.

— QPlace is called to place standard cells.

Figure 1 (A) shows the placement of the ibm02 design (see Section 5), produced with the Cadence SEDSM flow recommended for circuits with large number of macros. As seen there is a large amount of overlaps between macros and the designer is expected to remove these overlaps manually. If the design is given directly to SEDSM placer, QPlace, a legal placement is produced, but the run-times and solution quality suffer. However, a new version of SEDSM is currently in beta-testing and implements a different macro-placement flow, achieving better results [Varadrajan and DeLendonck 2002]. Recently acquired by Cadence, Silicon Perspective developed the First Encounter tool which performs System-on-Chip Physical-Prototyping and Hierarchical Physical Design. First Encounter allows full-chip physical prototyping and emphasizes early floorplanning. Figure 1 (B) shows the placement of the same ibm02 design produced by the force-directed placer Kraftwerk



(A) (B) (C)

Fig. 1. **Figure (A) shows a design with 19601 cells (ibm02) design placed by Cadence SEDSM recommended flow for designs with large number of macros. There are overlaps between macros which the designer is expected to remove manually. Figure (B) shows the Kraftwerk placement for the same design. Again there are significant overlaps, which have to be removed. Figure (C) shows the Kraftwerk placement for a design with 23136 cells (ibm03). The overlaps between macros is much smaller than (B) and can probably be removed by simple techniques.**

[Eisenmann and Johannes 1998]. This placement also has a large amount of overlaps between macros. Figure 1 (C) shows a Kraftwerk placement for design ibm03. As seen there are no signifi cant overlaps between macros in this placement and relatively simple techniques should be able to legalize such a placement. However, Kraftwerk does not always produce such non-overlapping placements and more sophisticated legalization techniques are required.

In addition to Physical Design with IP blocks, mixed-size placement techniques are relevant in the context of Physical Synthesis, where layout starts before the netlist is fully synthesized. While our work does not address synthesis, the proposed techniques may be useful in Physical Synthesis tools that operate at the chip level.

Previous published work on mixed-size placement can be broadly classifi ed into two approaches, continuous optimization techniques and combinatorial optimization techniques. Continuous optimization techniques like force-directed approaches work well with less constrained designs having relatively large amount of white-space [Eisenmann and Johannes 1998; Mo et al. 2000]. On the other hand combinatorial techniques are particularly promising on constrained designs with less white-space [Nag and Chaudhary 1999]. Published works [Nag and Chaudhary 1999; Vijayan 1991] focus on overlap removal for macros only and did not consider mixed-size placement. An entirely different approach is pursued in [Chang et al. 2003]. Their placer mPG-MS is based on an earlier tool mPG, which recursively clusters the netlist to build a hierarchy. The top-level netlist of approximately 500 clusters is placed using Simulated Annealing (SA), and then the placement is gradually refi ned by unclustering the netlist and improving the placement of smaller clusters by SA. mPG-MS contributes a structure of bins, in which large and small blocks are placed during course placement. The coarse placement is necessarily overlap-free for big objects, but small objects must be further re-placed by a detail placer. A signifi cant effort is expended to check for overlap during refi nement and legalize possible violations. A recent work [Choi and Bazargan 2003] also deals with the mixed-size placement problem. The authors of [Choi and Bazargan 2003] proposed a mixed-size placement flow which

combines a hierarchical simulated annealing based floorplanner with partitioning based placement techniques to handles mixed-size placement problem. Their method starts with a netlist and a fixed-floorplan area. At each level of partitioning, "large" hard macros are extracted from the netlist and the rest of the standard-cells and small macros are partitioned into a number of "soft" modules using a min-cut partitioner. The mixed hard/soft modules are floorplanned using a slicing floorplanner. This is performed recursively until the modules contain less than 30 gates. The method employs area migration techniques to satisfy the fixed-outline constraints. However, this method does not produce completely legal placements with large overlaps remaining between macros. The reader is referred to the book [Sarrafzadeh et al. 2002] for a detailed background discussion of mixed-size placement.

The main contribution of our work is a methodology to place designs with numerous macros by combining floorplanning and standard-cell techniques. The proposed design flow is as follows:

— An arbitrary black-box (no access to source code required) standard-cell placer generates an initial placement.

— To remove overlaps between macros, a physical clustering algorithm constructs a fixed-outline floorplanning instance.

— A fixed-outline floorplanner [Adya and Markov 2001], generates valid locations of macros.

— With macros considered fixed, the black-box standard-cell placer is called again to place small cells.

This design flow provides a somewhat new "killer-application" for the many floorplanning techniques developed in the last five years, e.g., [Lin and Chang 2001]. Indeed, we do not insist on using a particular floorplan representation, but rather emphasize floorplanning as a step in large-scale placement with macros.

We also propose a second design flow combining a black box standard-cell placer and a force-directed placer. The proposed design flow is as follows:

— An arbitrary black-box (no access to source code required) standard-cell placer generates an initial placement.

— A force-directed placer [Eisenmann and Johannes 1998] is used in ECO mode to remove the overlaps while changing the initial placement minimally.

Depending upon the design requirements and characteristics of the design, either of our flows can be used to produce high quality placements of mixed-size designs.

We notice that existing academic placers Capo [Caldwell et al. 2000a], Dragon 2000 [M. Wang and Sarrafzadeh 2000], Feng Shui[Yildiz and Madden 2001] and Spade [Dutt 2000] cannot process movable macros. In fact, *all macros are removed* in the placement benchmarks described in [M. Wang and Sarrafzadeh 2000] (produced from the ISPD 98 circuit benchmarks), and all cells are artificially made 1-by-1. Therefore, we derived new placement benchmarks from the original ISPD 98 circuits, preserving macros and the areas of all cells. Having converted the benchmarks into Cadence LEF/DEF format, we compared the performance of our methods to Cadence commercial placer, QPlace, Kraftwerk, mPG-MS and and the mixed-size placement flow proposed in [Choi and Bazargan 2003].

The remaining part of the paper is organized as follows. Section 2 covers previous work relevant to force-directed placement and fixed-outline floorplanning. Two new design flows for macro placement are proposed in Sections 3 and 4. Section 5 presents empirical validation of our work, and future directions are discussed in Section 6. Section 7 concludes our work.

2. PREVIOUS WORK

In this section we outline the relevant background for our study. We briefly describe the top-down recursive bisection based placement framework, a generic force-directed placement and floorplanning algorithm, and a fixed-outline floorplanning algorithm. All these algorithms are used in our proposed mixed-size placement flows.

2.1 Top-down, Recursive Bisection Based 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, reformulating constraints, and cutting the netlist hypergraph [Caldwell et al. 2000a]. Such a netlist decomposition is typically done with the min-cut objective. Each hypergraph partitioning instance is induced from a rectangular region, or block in the layout. A block corresponds to (i) placement region with allowed locations, (ii) a collection of cells to be placed in this region, (iii) all nets incident to the modules in the region, and (iv) fixed-terminals which are cells outside the region. The top-down placement process can be viewed as a sequence of passes where each pass examines all blocks and if required, divides them into two smaller blocks using min-cut partitioning. In our work we use the top-down recursive bisection based placer Capo [Caldwell et al. 2000a]. Capo uniformly distributes the available whitespace [Caldwell et al. 2003] around the core. However, if non-uniform distribution is required, fake unconnected filler cells [Adya et al. 2003] can be used.

2.2 Kraftwerk: Generic Global Placement and Floorplanning

A force-directed method for global placement was introduced in [Eisenmann and Johannes 1998]. Their global placer is called Kraftwerk. In addition to the well known wirelength dependent forces, Kraftwerk uses additional forces to reduce cell-overlaps and to consider the placement area. The wirelength dependent quadratic objective function to minimize is described as follows. Let n be the number of movable cells in the circuit and (x_i, y_i), the coordinates of cell i. A placement of the circuit can be described by the $2n$-dimensional vector $\vec{p} = (x_1, .., x_i, ..x_n, y_1, .., y_i, .., y_n)^T$. The circuit connectivity is modeled as a graph. Cells are modeled as vertices and nets are modeled as edges. Hyperedges are modeled as cliques. The cost of an edge is modeled as the squared Euclidean distance between its adjacent vertices multiplied with the weights of the edges. The squared Euclidean distance between cells i and j is $(x_i - x_j)^2 + (y_i - y_j)^2$. The objective function sums up the cost of all edges and can be written in matrix notation as

$$\frac{1}{2}\vec{p}^T C\vec{p} + \vec{d}^T\vec{p} + const$$

This objective function is minimized by solving the linear equation system

$$C\vec{p} + \vec{d} = 0$$

Additional constant forces are introduced in [Eisenmann and Johannes 1998] to distribute the cells more evenly in the layout region.

$$C\vec{p} + \vec{d} + \vec{e} = 0$$

The force vector $\vec{e}$ contains additional forces working on each cell in the x and y direction. These additional forces try to move the cells from high density regions to low density regions in the layout, thus attempting to reduce the overlaps. The algorithm described in [Eisenmann and Johannes 1998] is iterative which determines the additional forces according to the current placement. In each iteration the forces acting on the cells are assumed constant and are used to calculate a new placement. The new placement is base for the next iteration step and so on. Each step of the algorithm is called a placement transformation. The transformation step can be applied to fully overlapping placements as well as nearly legal placements. Thus, the algorithm renders itself very elegantly to ECO style placement requirements.

It is argued in [Eisenmann and Johannes 1998] that their algorithm is able to handle large mixed-size placement problems without treating macros and standard cells differently. However, from our experiments, we conclude that if applied from scratch on constrained mixed-size designs with less whitespace, this algorithm frequently produces placements with large overlaps.

2.3 Fixed-outline Floorplanning

A fixed-outline floorplanner was proposed in [Adya and Markov 2001; 2003]. We describe the work briefly here.

A typical floorplanning formulation includes a set of blocks, that may represent circuit partitions in applications. Each block is characterized by *area* (typically fixed) and *shape-type*, e.g., fixed rectangle, rectangle with varying aspect ratio, etc. Multiple aspect ratios can be implied by an IP block available in several shapes as well as by a hierarchical partitioning-driven design flow for ASICs [Sherwani 1999; Kahng 2000] where *only the number of standard cells* in a block (and thus the total area) is known in advance. A solution to such a problem, i.e., a *floorplan*, specifies a selection of block shapes and overlap-free placements of blocks. Classical floorplanning minimizes a linear combination of area and wirelength. Among measures of circuit wirelength, the popularity of Half-Perimeter Wirelength (HPWL) function is due to its simplicity and relative accuracy before routing is performed. The HPWL objective gained relevance with the advent of multi-layer over-the-cell routing, where more nets are routed with shortest paths [Kahng 2000]. In floorplanners based on Simulated Annealing (e.g., with the Sequence-Pair representation [Murata et al. 1996]) the typical choice of moves is straightforward.

As pointed out in [Kahng 2000; Caldwell et al. 2000a], modern hierarchical ASIC design flows based on multi-layer over-the-cell routing naturally imply **fixed-die** placement and floorplanning, rather than the older **variable-die** style [Sherwani 1999], associated with channel routing, two layers of metal and feedthroughs. In such a flow, each top-down step may start with a floorplan of prescribed aspect ratio and with blocks of bounded (but not fixed) aspect ratios. The modern floorplanning formulation proposed in [Kahng 2000] is an inside-out version of the classical outline-free floorplanning formulation — the aspect ratio of the floorplan is fixed, but the aspect ratios of the blocks may vary.

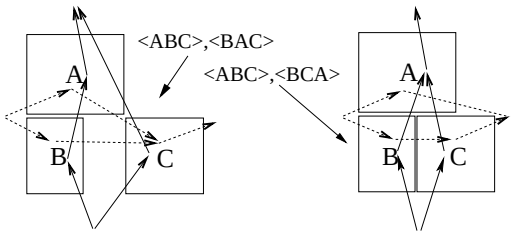


Fig. 2. **Two sequence-pairs with edges of the horizontal (dashed) and vertical (solid) constraint graphs.**

2.3.1 *Sequence-Pair Floorplan Representation.* An overwhelming majority of floorplanners rely on the Simulated Annealing framework [Sherwani 1999] but differ by internal floorplan representations.

The sequence-pair representation for classical floorplans consists of two permutations (orderings) of the N blocks [Murata et al. 1996]. The two permutations capture geometric relations between each pair of blocks. Recall that since blocks cannot overlap, one of them must be to the left or below from the other, or both. In sequence-pair

$$(<\ldots,a,\ldots,b,\ldots>,<\ldots,a,\ldots,b,\ldots>) \Rightarrow a \text{ is to the left of } b \quad (1)$$

$$(<\ldots,a,\ldots,b,\ldots>,<\ldots,b,\ldots,a,\ldots>) \Rightarrow a \text{ is above } b \quad (2)$$

In other words, every two blocks constrain each other in either vertical or horizontal direction. The sequence-pair representation is shift-invariant since it only encodes pairwise relative placements. Therefore, placements produced from sequence-pairs must be aligned to given horizontal and vertical axes, e.g., $x = 0$ and $y = 0$.

The original work on sequence-pair [Murata et al. 1996] proposed an algorithm to compute placements from a sequence-pair by constructing horizontal (H) and vertical (V) constraint graphs. The H and V graphs have $N+2$ vertices each — one for each of N block, plus two additional vertices :"the source" and "the sink". For every pair of blocks a and b there is a directed edge $a \rightarrow b$ in the H graph if a is to the left from b according to the sequence-pair (Formula 1). Similarly there is a directed edge $a \rightarrow b$ in the V graph if a is above b according to the sequence-pair (Formula 2) — exactly one of the two cases must take place. Vertices that do not have outgoing edges are connected to the sink, and vertices that do not have incoming edges are connected to the source. Both graphs are considered vertex-weighted, the weights in the H graph represent horizontal sizes of blocks, and the weights in the V graph represent vertical sizes of blocks. Sources and sinks have zero weights.

Block locations are the locations of block's lower left corners. The x locations are computed from the H graph, and y locations are computed from the V graph *independently*. Therefore, we will only discuss the computation of the x locations. One starts by assigning location $x = 0$ to the source. Then, the H graph is traversed in a topological order. To find the location of a vertex, one iterates over all incoming edges and maximizes the sum of the source location and source width. Figure 2 illustrates the algorithm on two examples. The worst-case and average-case complexity of this algorithm is $\Theta(n^2)$, since the two graphs, together, have a fixed $\Theta(n^2)$ number of edges, and topological traversals take linear time in the number of edges.

Sequence-pairs can be used to floorplan hard rectangular blocks by Simulated Annealing

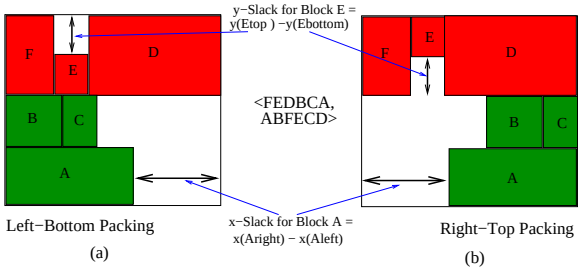


Fig. 3. **Slack Computation. In (a) the floorplan is evaluated left-to-right and bottom-to-top. In (b) the floorplan is evaluated right-to-left and top-to-bottom. The slacks for each block is the difference between its positions in the two evaluations.**

[Murata et al. 1996; Murata and Kuh 1998; Tang et al. 2000; Tang and Wong 2001]. The moves are (i) random swaps of blocks in one of the two sequence-pairs, and (ii) rotations of single blocks. Sequence-pairs are modified in constant time, but need to be re-evaluated after each move. No incremental evaluation algorithms have been reported, therefore, the annealer spends most of the time evaluating sequence-pairs.

The sequence-pair representation and necessary algorithms have been extended to handle fixed blocks [Murata and Kuh 1998] as well as arbitrary convex and concave rectilinear blocks [Fujuyoshi and Murata 1999]. Recently, the original $O(n^2)$-time evaluation algorithm [Murata et al. 1996], has been simplified and sped up to $O(n\log(n))$ in by Tang et al. [Tang et al. 2000], and then to $O(n\log(\log(n)))$ [Tang and Wong 2001]. Importantly, those algorithms do not change the semantics of evaluation — they only improve runtime, and lead to better solution quality by enabling a larger number of iterations during the same period of time. While O-trees [Pang et al. 2000] and corner block lists [Hong et al. 2000] can be evaluated in linear time, the difference in complexity is dwarfed by implementation variations and tuning, e.g., the annealing schedule. The implementation reported by Tang et al. [Tang and Wong 2001] seems to outperform most known implementations, suggesting that the sequence-pair is a competitive floorplan representation.

All three sequence-pair evaluation algorithms are based on the following theorem [Tang et al. 2000]: The x-span of the floorplan to which sequence pair (S_1, S_2) evaluates is equal to the length of the longest common weighted subsequence of S_1 and S_2, where weights are copied from block widths. An analogous statement about the y-span deals with the longest common subsequence of S_1^R and S_2 , where ^{R} denotes the "reversed" sequence and weights are copied from block heights. Moreover, the computations of x and y locations of all blocks can be integrated into the longest common subsequence computations.

2.3.2 *Floorplan Slacks.* The notion of slack can be used with any of above mentioned sequence pair evaluation algorithms and potentially other floorplan representations[Adya and Markov 2003]. Each block in a floorplan has two types of slacks: horizontal slack and vertical slack. Slack of a block in a floorplanning instance represents the distance (in a particular dimension) at which this block can be moved without changing the outline of the current floorplan. Blocks with zero slacks in a particular dimension must lie on critical paths in the relevant constraint graph.

We will base our discussion on the horizontal slack. The discussion on vertical slack is analogous. As shown in Figure 3, horizontal slacks can be computed with any floorplan

representation that can be evaluated left-to-right and right-to-left. Once the X-size of the floorplan is computed by packing left-to-right, one can re-pack it right-to-left. The horizontal slack of a block is the difference between the block's locations produced by those two packings. The floorplanner Parquet [Adya and Markov 2001] uses the Sequence-Pair representation because of the simplicity of the representation.

Once slacks for each block are known, they can be used in move selection. The rationale here is to reduce the floorplan size in a given dimension (X or Y) without impairing the hill-climbing abilities of Simulated Annealing. The new mechanism is combined with pairwise swaps and block rotations that are typically used in Sequence-Pair based annealers. If a move (such as pairwise swap) does not involve at least one block with zero slack in a given dimension, then the floorplan size in that dimension cannot decrease after the move. This is because such a move cannot improve critical paths or, equivalently, longest common subsequences [Tang et al. 2000; Tang and Wong 2001]. Therefore *move selection is biased towards blocks having zero slack in at least one dimension*. Of those blocks, the ones with large slack in the other dimension are often good candidates for single-block moves, such as rotations and gradual (discrete or continuous) changes of aspect ratio. Blocks with zero slack in both the directions, especially small blocks, are good candidates for a new type of move, in which a block is moved simultaneously in both sequence pairs to become a neighbor of another block (in both sequences, and, thus in placement). One possible heuristic is to move a critical block C next to a block L with as large a slack as possible, since large slacks imply that whitespace can be created around L.

2.3.3 *Handling Soft Blocks.* We can also use slack-based move types to change aspect ratios of soft blocks [Adya and Markov 2003]. During annealing, at regular intervals, a block with low (preferably zero) slack in one dimension and large slack in the other dimension are chosen. The height and the width of such a block is changed within allowable limits so that its size in the dimension of smaller slack is reduced (to increase the slack). Such moves are greedily applied to all soft blocks in the design.

2.3.4 *Wirelength Minimization.* In classical floorplanning, the global objective is to minimize wirelength and total area of the design. This implies multi-objective minimization. Typically, most simulated annealing based floorplanners use a linear combination of area and wirelength as an objective for the annealer.

Additional moves can be designed to improve the wirelength [Adya and Markov 2003]. For a given block a, we calculate, using analytical techniques, its "ideal" location that would minimize quadratic wirelength of its incident wires N. We determine the ideal location (x_a, y_a) of block a which minimizes the following function.

$$\sum_{N \in a} \sum_{v \in N} (x_v - x_a)^2 + (y_v - y_a)^2$$

The ideal location (x_a, y_a) of block a is simply the average of the position of all modules connected to block a. We then identify the block b closest to the ideal location. This is done by expanding a circle centered at the ideal location and identifying the closest block b. We then attempt to move block a in the sequence pair so that in both sequences it is located next to b. As explained earlier, we evaluate the four possible ways to do that, and choose the best. Thus an attempt is made to move a close to its ideal location to minimize quadratic wirelength.

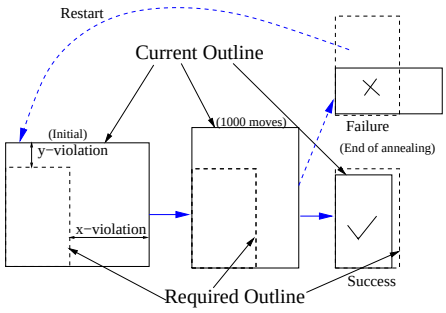


Fig. 4. **Snap-shots from fixed-outline floorplanning. The number of annealing moves is fixed, but if the evolving floorplan fits within the required fixed-outline, annealing is stopped earlier. If at the end of annealing the fixed-outline constraints are not satisfied, it is considered a failure and a fresh attempt is made.**

2.3.5 *Fixed-outline Floorplanning.* Fixed-outline floorplanning can be performed using Simulated Annealing, taking advantage of new types of moves that are based on the notion of *floorplan slack* [Adya and Markov 2001]. The following notation will be used in the floorplanning formulations. For a given collection of blocks with total area A and given *maximum percent of white-space* γ, we construct a fixed outline with aspect ratio $\alpha \geq 1$.[1]

$$H_* = \sqrt{(1+\gamma)A\alpha} \quad W_* = \sqrt{(1+\gamma)A/\alpha}$$

Aside from driving the annealer by area minimization, we can consider the following objective functions: (i) the sum of the excessive length and width of the floorplan, (ii) the greater of the two. Denoting the current height and width of the floorplan by H and W, we define these functions as

$$\textbf{(i)}\ \max\{H-H_*,0\}+\max\{W-W_*,0\} \quad \textbf{(ii)}\ \max\{H-H_*,W-W_*\}$$

The choice of these functions is explained by the fact that the fixed-outline constraint is satisfied when each of those functions takes value 0 or less. For this reason we cannot consider the product of fixed outline violations.

Figure 4 shows the evolution of the fixed-outline floorplan during Simulated Annealing with slack-based moves. The scheme works as follows. At regular time intervals (during area-minimizing Simulated Annealing) the current aspect ratio is compared to the aspect ratio of the desired outline. If the two are sufficiently different, then the slack-based moves described earlier are applied to bias the current aspect ratio in the needed direction. For example, if the width needs to be reduced, then choose the blocks in the floorplan with smallest slack in the X dimension and insert them above or below the blocks with largest slack in the Y dimension. These moves have better chances of reducing the area and improving the aspect ratio of the current floorplan at the same time. Using such repeated corrections, the structure of the floorplan is biased towards the aspect ratio of the fixed outline.

While a number of works on floorplanning discuss floorplan constraints, the results in [Adya and Markov 2003][Adya and Markov 2001] empirically demonstrate high ratios of

[1]The restriction of $\alpha \geq 1$ is imposed without loss of generality since our fborplanner can change orientations of individual blocks.

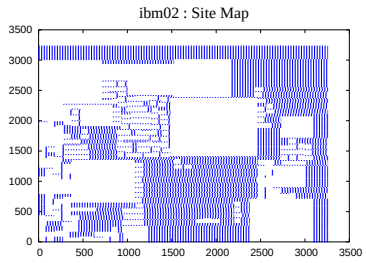


Fig. 5. **Map of cell sites for the ibm02 design with all the macros marked as fixed. Sites under the macros are removed.**

successes to failures in the flow from Figure 4.

3. MIXED-SIZE PLACEMENT FLOW 1

Our first proposed flow for mixed-size placement requires a black-box standard-cell placer that can place cells of equal height in rows that consist of cell sites, along the lines of the data-model implied by Cadence LEF/DEF. We also require that the placer can handle fixed cells/pins and can handle rows consisting of contiguous sub-rows. By removing cell sites from a sub-row and splitting the sub-row into two sub-rows, one can model the effect of fixed macros (because pins of fixed macros are fixed as well). For example, the site map in Figure 5 corresponds to the placement in Figure 10 (c). Our flow also uses a fixed-outline floorplanner described in Section 2.3. While our floorplanner uses the sequence-pair representation, a variety of other floorplan representations can be used.

3.1 Shredding Macro Cells

A hierarchical recursive bisection based placer has trouble handling mixed-size netlists [Sarrafzadeh et al. 2002] because of the large variations in the cell sizes. We get around this inherent problem by shredding all the macros to make the netlist more homogeneous in terms of cell sizes. The DOMINO detailed placer introduced the idea of shredding large cells to simplify placement [Doll et al. 1994]. To apply this technique in global placement, one must additionally handle cell orientations and remove cell overlaps (other than by left-to-right packing).

Our flow starts with a pre-processing step during which all macros are shredded into a number of smaller cells of minimal height. The number of these cells is determined by the area of the macro and the width of sub-cells. A macro shredded into sub-cells is shown in Figure 6. A sub-cell with row index i and column index j may be identified as $a_{i,j}$, and its immediate neighbors are $a_{i-1,j}$, $a_{i+1,j}$, $a_{i,j-1}$ and $a_{i,j+1}$. Fake two-pin nets are added between neighboring sub-cells to ensure that sub-cells are placed close to each other when wirelength is minimized. The number of fake nets added between each pair of sub-cells determine how strongly the sub-cells are tied to each other. We add three fake nets between each connected, neighboring sub-cells. The total number of faked wires depends on the width of sub-cells. A cleverly implemented placer could handle the faked wires implicitly, e.g., using net weights. In any case, a large-scale global placer with near-linear runtime (e.g., a fast min-cut placer) should be able to handle the increased number of wires. The Capo placer [Caldwell et al. 2000a] we use is scalable enough. The shredding procedure

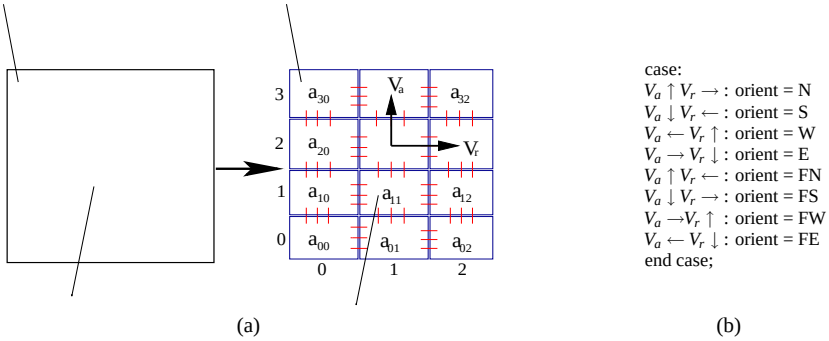


Fig. 6. **A macro is shredded into cells of minimal height, connected by fake wires. To find the orientation of the macro from locations of sub-cells, the relative locations of sub-cells $a_{i,j}$, $a_{i+1,j}$ and $a_{i,j+1}$ are analyzed for every eligible (i, j). Figure (b) shows the case analysis in terms of vectors V_a and V_r in final placement. "N","S","W","E" stand for "North", "South", "West" and "East" respectively. "F" stands for "Flipped". Any net connected to a macro pin is propagated to the respective sub-cell as shown.**

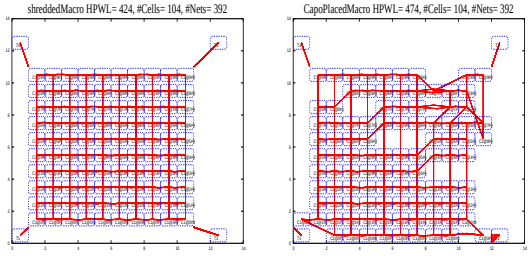


Fig. 7. **A design with only 1 macro and 4 terminals. Fig (a) shows the macro shredded into sub-cells. The sub-cells are placed at ideal locations. The fake nets connecting them form a regular grid structure. Fig (b) shows the shredded design placed by Capo followed by detailed placement. The sub-cells are placed close to each other and also maintain the initial grid structure(in (a)) on average.**

can be viewed as the equivalent of descending one level of hierarchy in a hierarchical design by flattening the macro. If the sub-cells of the macro are not placed close to each other in the placement of the flattened design then it implies that the macro was not formed properly, i.e. the clustering technique employed to form the macro did not work very well. Artifi cially shredding the macro makes the new placement problem more homogeneous and thus a fi nely tuned min-cut based placer can handle the shredded design better than handling the original placement problem with macros.

The shredded design with the fake nets is placed using the global placer Capo. The resulting placement does not immediately imply the locations of the original macros, because the macros are shredded. The center-location of a given macro is determined by averaging the locations of all sub-cells of that macro. Since the sub-cells of a macro are connected in a regular grid structure, a good placer will ensure that the sub-cells are placed

close to each other and in the original grid like structure. Determining the orientation of the macros which is globally consistent with the placement is very important. A top-down global placement methodology that handles large macros by fixing macros in partitions as soon as the macros become too large for the partition, has problems determining the orientation of individual macros. We developed a heuristic to determine the orientation of the macro using the initial placement information. The heuristic is based on the relative placement of each sub-cell with respect to its immediate neighbors. Namely, the placement of sub-cell $a_{i,j}$ is compared with the placements of $a_{i+1,j}$ and $a_{i,j+1}$. This is illustrated in Figure 6, where two vectors are computed for a given cell and then analyzed to produce one of eight possible orientation types. For each macro, a score table is maintained which records the number of sub-cells placed in a particular orientation. The orientation of the macro is chosen according to the highest score (if several orientations have comparably high scores, then we cannot conclude the orientation with certainty). The rationale is that the extra nets added while shredding will, in many cases, help the macro to approximately maintain its shape. Figure 7 shows an example design with only one macro and four terminals. Figure 7 (a) shows the macro shredded into sub-cells and connected in a regular grid-like fashion by fake wires. The sub-cells are placed at ideal locations. Figure 7 (b) is the shredded design placed by Capo followed by detailed placement. The sub-cells are placed close to each other and maintain the initial grid structure on average. From the placement of the shredded design we deduce that the macro is placed in the north orientation.

Thus, a crude placement (with orientations) is obtained by placing the shredded design. Since the standard cells were placed by using wirelength-minimization, highly connected cells will be close to each other, but macros may overlap with each other and may not be placed entirely inside the layout region. Figure 10 (a) shows the placement of the ibm02 circuit produced as explained above.

While our technique allows one to deduce the prevailing orientation of a macro or observe that there is no prevailing orientation, some macros may only be placeable in one orientation. Such a constraint can be ensured by tying the corners of the macro (i.e., the respective sub-cells after shredding) to the corners of the layout by strong (heavy) faked wires, as shown in Figures 8 (a) and (b). During the minimization of HPWL, e.g., by recursive min-cut bisection, the orientation of the macro will be preserved, and the quality of placement will not be affected. A formalization follows.

Lemma: Placements that minimize HPWL in the original design subject to orientation constraints are in a one-to-one correspondence with unconstrained placements that minimize HPWL, including the fake wires that tie the corners of macros to the corners of the layout region (assuming sufficiently strong wires).

The lemma can be proven along the lines of Figure 8, where five out of eight possible orientations of a macro tied to the four corners of the layout region are shown. Note that this result does not apply to quadratic placement, and in that case all tied macros will be attracted to the center of the layout.

3.1.1 *Better Placement of Regular Netlists* . Observe that placement of the shredded netlists calls for placement of grid-graphs embedded into random-logic netlists. However, we discovered that Capo 8.5 placer used in [Adya and Markov 2002] performs poorly on grid-graphs, as shown in Figure 9 which illustrates an optimal and a sub-optimal placement of a 10×10 grid with four fixed cells in the corners. This is hardly a surprise because generic standard-cell placers are known to perform badly on regular, data-path style de-

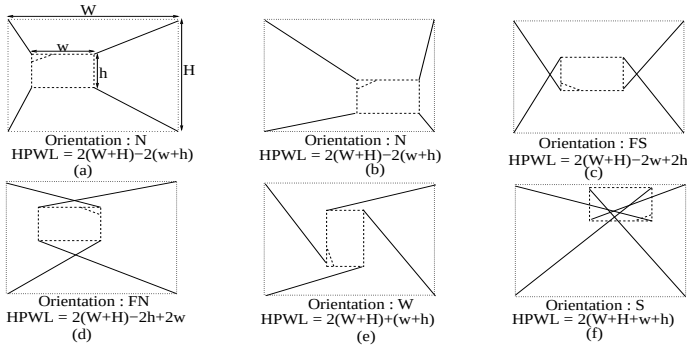


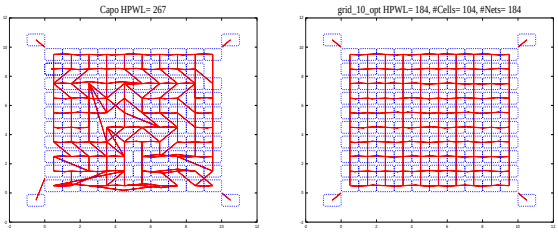
Fig. 8. **Five out of eight orientations of a macro whose corners are tied to the corners of the layout region; the orientation is N in (a) and (b). The [linear] length of faked wires depends only on the orientation and not on the location of the macro, as long as the macro is placed entirely within the layout region. The desired orientation (N in this example) is found by wirelength minimization.**

signs [Dally and Chang 2000]. Our improvements Capo allow it to better handle regular netlists without the loss of performance on random-logic netlists. These improvements are described below, and their implementation was contributed to Capo 8.6.

During each partitioning step with a vertical cut line, Capo 8.5 with default parameters uses a fairly large tolerance (of the order of 10-20%) in order to find better cuts. After a good cut is found, the geometric cut line is adjusted according to the sizes of partitions, with an equal distribution of whitespace among the partitions. However, *if no whitespace is available in the block*, this technique can cause cell overlaps. Namely, since cut-lines cannot cut through cell sites and since no "jagged" cut-lines are allowed, the set of partition balances that can be realized with a straight vertical cut-line and zero whitespace is fairly discrete. Capo 8.5 simply rounds the current balance to the closest realizable and sets the geometric cut-line accordingly. When whitespace is scarce, one of the resulting partitions may be overfull and the other may have artificially-created whitespace. Only a relatively small number of cell overlaps can be created this way, but they can be spread through the whole core area. When used in the MetaPlacer shell, Capo 8.5 removes overlaps after global placement by a simple and very fast greedy heuristic. However, this heuristic increases wirelength.

In an attempt to reduce the number of overlaps, we revised the partitioning process in Capo. When a placement block is partitioned with a vertical cut-line, at first the tolerance is fairly large. As described previously, this allows Capo to determine the location of the geometric cut-line by rounding to the nearest site. Furthermore, if the block has very little whitespace, we then repartition it with a small tolerance in an attempt to re-balance the current partitions according to the newly defined geometric cut-line.

Another modification we implemented is related to terminal propagation. Normally, if a projection of a terminal's location is too close to the expected cut-line, the terminal is ignored by Capo in an attempt to avoid excessively speculative decisions. The proximity threshold is defined in percent of the current block size, and this parameter is called "partition fuzziness". For example, suppose that the *y* location of a terminal is within 9% of the tentative location of the horizontal cut-line. Then, with partition fuzziness of 10%, this



Capo 8.5 Capo 8.6

Fig. 9. **Capo placements for designs with regular grid connectivity. Capo 8.5 produces sub-optimal placements. Capo 8.6 produces the optimal placement for this design. There are 4 terminals connected to the 4 corner cells to anchor the design.**

Circuit	#Nodes	#Nets	WS %	Optimal HPWL	Dragon HPWL	Kraftwerk HPWL	Capo default HPWL	Capo + repart HPWL
10x10	100	184	0	184	293	202	267	184
95x95	9025	17864	5	17884	39687	18302	21828	22764
100x100	10000	19804	0	19804	46066	20519	38352	21314
190x190	36100	71824	5	71864	175623	75384	90665	89814
200x200	40000	79604	0	79604	198182	82335	193167	100041

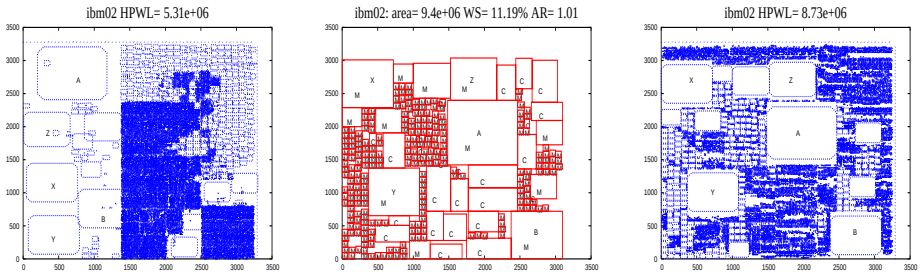
Table I. **Wirelength achieved by several placers on regular grids of varying size and with varying whitespace. While Kraftwerk produces small wirelength on $n \times n$ grids, it often fails to converge to a solution on random-logic netlists with embedded grids.**

Circuit	Original Netlist		Connectivity Based Clustering			Physical Clustering		
	#Nodes	#Nets	#Nodes	#Nets	Time	#Nodes	#Nets	Time
ibm01	12752	14111	196	**8732**	15.79s	282	**4689**	0.24s
ibm02	19601	19584	168	**14017**	42.26s	297	**6478**	0.39s
ibm03	23136	27401	272	**17069**	63.83s	332	**8723**	0.54s
ibm04	27507	31970	216	**20886**	84.05s	327	**9519**	0.58s
ibm05	29347	28446	28	**17280**	89.23s	37	**11630**	0.57s

Table II. **Two different clustering schemes to form floorplanning instances. Connectivity based greedy scheme groups highly connected objects together. Physical clustering groups objects, placed close to each other. In the clustered design, nets which connect only objects within a certain group are collapsed. The metric used to compare is the number of nets in the final clustered design. Physical clustering is more successful in reducing the number of nets crossing the groups compared to connectivity based clustering.**

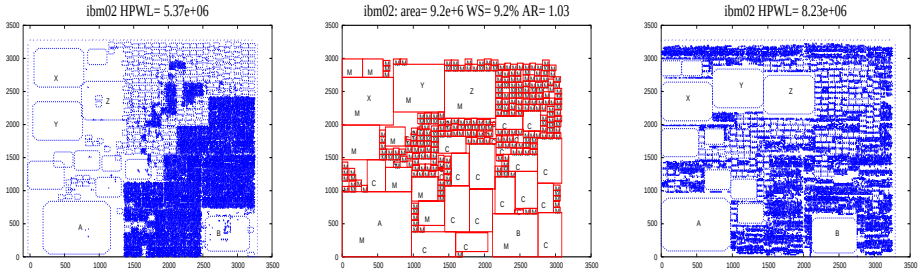
terminal will be ignored during partitioning. Our studies of Capo performance on grids suggest that partition fuzziness should be tuned up, particularly for small blocks. For example, if a placement block has only three cell rows, then possible tentative locations of horizontal cut-lines are relatively far from the center. In a neighboring block that has not been partitioned yet, all cells are "located" at the center of the block, causing all connected terminals to propagate into one partition in the current block. To avoid this, we increased partition fuzziness to 33%.

The two changes described above improve the performance of Capo on the grid designs with 0% whitespace by a factor of two. The results for performance of various placers on grid graphs [Adya et al. 2003][Adya et al. 2004] are reported in Table I.



(a) (b) (c)

Fig. 10. **Mixed-size placement Flow 1 explained in Section 3. Figure (a) is the placement (illegal) obtained after running Capo on ibm02 design with macros shredded into small cells. The locations of macros are determined by averaging the locations of sub-cells. Note that macro Z is not placed entirely within the layout region. Also, macro B overlaps with macro Y and standard cells. Figure (b) shows a possible final fixed-outline floorplan of the same design. The floorplanning is done from scratch and no attempt is made to preserve the original locations of macros. Macros are marked with M and clusters of standard cells with C. Aspect ratios of macros are fixed and those of cell clusters vary between 1/2 and 2. Observe that the vertical coordinates of three (A, X and Y) out of five large macros are similar to those in Figure (a). Figure(c) is the final placement of ibm02. The locations of all macros are taken from the floorplan in Figure (b).**



(a) (b) (c)

Fig. 11. **Same mixed-size placement flow as that described in figure 10. However, during the floorplanning stage (b), low-temperature annealing is used in an attempt to preserve the initial locations of macros obtained by placing the shredded netlist. The initial sequence-pair for floorplanning is constructed from the existing placement (a). As seen from (a), (b) and (c) positions of macros A,B,X,Y,Z are close to their original locations. However runtimes for the floorplanning stage increase because of larger number of tries required to satisfy the fixed-outline constraints when using low-temperature annealing.**

3.2 Physical Clustering

The crude placement obtained from the above step may have overlapping macros as well as macros placed outside the layout region (Figure 10 (a)). Such violations must be corrected without affecting the placement quality considerably. This can, in principle, be done by fi xed-outline floorplanning, but the number of movable objects is unrealistically large (every standard cell is movable). We therefore construct a fi xed-outline floorplanning

instance through physical clustering based on locations of standard cells. Cells that are placed together are merged into soft clusters (i.e., the aspect ratio may vary). This is done by gridding the layout region and putting all the standard cells that physically fall within a grid region into a cluster. We recommend computing the dimensions of the grid based on the number of standard cells and macros in the design. However, in our experiments we used a grid of size 6 x 6 in order to speed up floorplanning. This grid worked well for smaller benchmarks, but appeared too coarse for larger benchmarks. The original macros are not clustered to anything, and their aspect ratio is allowed to change just as in the original placement formulation. For each cluster, the nets connecting only blocks within the cluster are discarded.

Since the design has been initially placed with small wirelength, the generated clusters contain strongly connected cells. Alternatively, one could use connectivity-based clustering algorithms [Alpert et al. 1997; Karypis et al. 1997]. We compared the physical clustering scheme with one such simple greedy clustering scheme that consists of a series of passes. Each pass identifies pairs of connected vertices (the more connections between a pair of vertices, the more likely it is to be selected). Each pair is substituted with a cluster. The clustered vertices are removed, and all incident nets are connected to the new cluster. All nets whose pins are inside a single cluster are removed. Every such pass results in the reduction of the nodes in the netlist approximately by a factor of two. The next pass is applied to the clustered netlist, and passes are continued until the desired reduction in size is achieved.

We compare this greedy connectivity-based clustering with physical clustering by the number of nets assuming approximately equal number of clusters (which is somewhat more rigorous than comparing the nets-to-clusters ratio). The results in Table II suggest that the physical clustering scheme is more successful in reducing the number of nets because even for larger numbers of clustered nodes it has lower numbers of nets. Of course, one could use more involved clustering schemes based on connectivity such as those using multi-way min-cut partitioning. On the other hand, our physical clustering implicitly includes those algorithms. Another advantage is that our physical clustering based on the initial placement accounts for both netlist connectivity and the shapes of macros. Moreover, the initial placement can give the exact pin locations of the pins in the clustered cell. The initial placement is additionally used to construct an initial floorplan for Simulated Annealing. The blocks in this floorplan do not overlap, but may not fit into the desired outline. The initial placement run thus gives us information about macro locations, desired macro orientations and highly connected cells to be clustered.

3.3 Fixed-outline Floorplanning With Macros

The placement of macros obtained by placing the shredded netlist may have some overlaps and is in general not legal. We need to remove the overlaps between macros and ensure that they are all placed within the layout boundary. There are several possible options like using the approach in [Nag and Chaudhary 1999] for post-placement residual-overlap removal or force-directed approaches [Eisenmann and Johannes 1998]. However such approaches work well in less constrained designs with relatively large white-space.

As explained in Section 2.3.5, we use the fixed-outline floorplanner Parquet [Adya and Markov 2001] to satisfy the fixed-outline constraints imposed by fixed-die paradigm. This new version of Parquet is used to floorplan hard macros together with soft clusters of standard cells. The outline of the required floorplan is derived from the layout region

and is used as a constraint, with wirelength as the objective function. We configure the floorplanner to make multiple tries until it satisfies the fixed-outline constraint. In our experiments, the floorplanner typically succeeded on the first try, but the ratio of successes to failures may depend on the amount of whitespace in the design.

In our experiments the annealer found good floorplans where some macros were moved from their locations in the initial floorplan (see Figure 10 (b)). We therefore believe that closely following the initial floorplan is not necessary for wirelength minimization and that the necessary information from the initial placement is captured by the physical clustering. However, if other design concerns encourage the preservation of macro placements, one could use more incremental force-directed macro placers [Mo et al. 2000]. Alternatively, one could tie those macros with faked wires to faked pins placed in strategic locations. We tried a variant of our flow in which we employed low-temperature annealing in the floorplanning stage in an attempt to preserve the initial locations of macros. The initial sequence pair for floorplanning is constructed from the placed shredded design. Low-temperature annealing is employed with slack-based moves to satisfy the fixed-outline constraints. Snapshots of different stages of the flow with low temperature annealing are illustrated in Figure 11. Note that the relative and absolute positions of macros in Figure 11 (c) are close to the initial macro positions in Figure 11 (a).

There are a number of factors and choices at the floorplanning stage that can affect the final placement. We list them below:

— **Whitespace available in the design**. Fixed-outline constraints can be easily satisfied for designs with large amount of whitespace. However, for constrained designs the floorplanner can take a large amount of time in trying to satisfy the fixed-outline constraints.

— **Area assigned to the soft clustered blocks of standard cells**. The area assigned to the clustered block is the sum of the areas of the standard cells forming the cluster. However, for constrained designs with limited whitespace one can try to reduce the areas of these clustered macros to make it easier for the floorplanner to find a solution satisfying the outline constraints. However, floorplanning using sequence-pairs compacts blocks down and leftwards. Therefore, if you have a large amount of whitespace in the design, area-minimization will ensure that no macro is placed in the top-right corner. This may harm the solution quality if achieving minimum wirelength requires placing a particular macro in the top-right corner. Therefore, for large whitespace designs reducing the area of the clustered block can hurt wirelength optimization.

— **Try to preserve the original locations of macros**. One might want to preserve the initial locations of the macros provided by the placement of the shredded netlist. In this case the purpose of floorplanner is to only remove the overlaps.

We study the effect of these choices on the final placements in Section 5.2.1.

3.4 Final Standard-cell Placement with Fixed Macros

The final locations of the macros are taken from successful fixed-outline floorplans, and the macros are fixed in the original layout. All cell sites below the macros are removed, and cell rows may need to be split into sub-rows. This enables the standard-cell placer to place the remaining movable standard cells without overlaps with the macros. In our experiments, we used the Capo min-cut placer [Caldwell et al. 2000a], followed by two passes

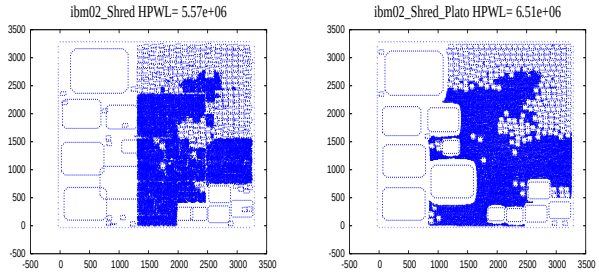


Fig. 12. **Mixed-size placement Flow 2 as that described in Section 4. Figure (a) shows the placement obtained after placing the shredded netlist. This is the same as in the first flow. Figure (b) shows the overlap free placement obtained after running Kraftwerk in ECO mode on the placement in (a). Sometimes Kraftwerk is not successful in removing all the overlaps.**

of window-based branch-and-bound placement.[2] Figure 10 (c) shows the final placement generated by our proposed flow for the ibm02 design.

Min-cut placers that uniformly distribute whitespace [Caldwell et al. 2000a][Caldwell et al. 2003] tend to produce excessive wirelength when large amounts of whitespace are present [Alpert et al. 2002]. In our flow, during the final placement of standard cells around the macros, the whitespace available in the designs (20% for ibm benchmarks) is distributed uniformly around the design with routability in mind. However, if the placement objective is only to minimize wirelength (as in this study), one can use more intelligent whitespace allocation techniques [Alpert et al. 2002]. As a variant of our original flow, in the final stage of our flow we add unconnected filler cells to the design to represent the excessive whitespace and reduce the whitespace available to the placer to 10%. Thus the standard cells are placed more compactly with the filler cells occupying vacant areas of the chip. The effect of filler cells on the final placement is studied in Section 5.2.1. However, we also point out that reducing HPWL may result in worsening the routability of a design.

4. MIXED-SIZE PLACEMENT FLOW 2

Our second proposed flow for mixed-size placement combines a black-box standard cell placer and a force-directed placer [Eisenmann and Johannes 1998]. The flow is described as follows.

4.1 Shredding Macro Cells

This step is identical to that in Section 3.1. The netlist is first pre-processed and all the macros in the design are decomposed into tightly connected smaller sub-cells of minimal height. For each macro the sub-cells are connected in a grid-like fashion. The shredded design with fake sub-cells and nets is placed using the Capo placer. The locations and orientation of the macros is determined from the placement of the shredded netlist. This initial placement can have overlaps between macros. The second step attempts to make the placement overlap-free. The first step entails placing a random logic netlist with embedded

[2]As the detailed placement step, we apply branch-and-bound end-case placers [Caldwell et al. 2000b] using sliding windows.

grid graphs. Standard cell placers Capo, Dragon and Cadence QPlace have no difficulties in placing these netlists. However the force-directed placer Kraftwerk often fails to converge to a solution on such netlists.

4.2 Overlap Removal Using Force-Directed Techniques

We employ the ECO capabilities of the force-directed placer Kraftwerk [Eisenmann and Johannes 1998] to remove the overlaps from the initial placement while disturbing the initial placement as little as possible. As explained in Section 2.2, Kraftwerk can take an initial placement and work in an ECO mode. In the ECO mode, Kraftwerk starts from the given placement and introduces additional forces according to density deviations arising because of the existing overlaps. The additional forces move the surroundings slightly in order to remove the overlaps. The algorithm tries to preserve the relative placement of cells. If the overlaps in the initial placements are small the additional forces are small resulting in small changes for the placement. However, if the overlaps in the initial placement are large this procedure can result in large changes to the initial placement and in some cases may also produce placement with large overlaps. Kraftwerk stops its placement iterations once the placement density in each region is below a certain threshold. However, this may result in small overlaps between the macros. From our experiments we conclude that the percentage of these overlaps with respect to the total layout area is fairly small. Alternatively one could employ other techniques [Mo et al. 2000] [Nag and Chaudhary 1999] [Vijayan 1991] to attempt and remove the overlaps. Figure 12 shows the placement obtained after running this flow on design ibm02. As seen the final placement corresponds very accurately to the initial seed placement.

5. RESULTS

Our proposed flow is implemented in C++ and compiled by `g++ 2.95.4 -O3`. Runtimes are measured on a 2 GHz PC/Intel system running Linux. We compare our results against QPlace v.5.1.67 from Cadence, whose runtimes are measured on a 500 MHz Sun Blade-100 system running Solaris. We also compare our results against force-directed placer, Kraftwerk [Eisenmann and Johannes 1998] and mPG-MS [Chang et al. 2003]. Runtimes for Kraftwerk are measured on a 2 GHz PC/Intel system and for mPG-MS are observed on a 750 MHz Sun Blade-1000 system running Solaris.

5.1 Benchmarks

The benchmarks used in our experiments are derived from the ISPD-98 (IBM) circuit benchmarks [Alpert 1998]. We converted the netlists into the Bookshelf placement format [Caldwell et al.], added placement-related information and made the new benchmarks available on the Web. [3] The original descriptions specify cell areas, but not their dimensions. Since in the `ibm` benchmarks, all areas are divisible by 16, we define rows of height 16. Cell sites in all rows have width 1. Cell widths were computed by dividing cell areas by row height (16). When the width of a cell exceeded a threshold number of sites (100 in our case), we upgraded such a cell to the status of a multi-row macro with aspect ratio 1. The height of such a macro is computed by rounding the square-root of the area to the closest integer multiple of row height (16). The width is computed by dividing cell area by cell height and rounding the result to the closest integer number of cell sites. All designs have

[3] at `http://vlsicad.eecs.umich.edu/BK/ISPD02bench/`

Circuit	#Nodes	#Nets	#Macros	A_m^b	ΣA_m	$A_m^b : A_m^s : A_c^s$
ibm01	12752	14111	246	6.37%	67.13%	8416:252:1
ibm02	19601	19584	280	**11.36%**	76.89%	30042:240:1
ibm03	23136	27401	290	**10.75%**	70.75%	33088:240:1
ibm04	27507	31970	608	**9.15%**	59.82%	26593:240:1
ibm05	29347	28446	0	N/A	N/A	N/A
ibm06	32498	34826	178	3.95%	72.90%	36347:175:1
ibm07	45926	48117	507	4.75%	52.56%	17578:240:1
ibm08	51309	50513	309	**12.10%**	67.35%	50880:240:1
ibm09	53395	60902	253	5.42%	52.42%	29707:240:1
ibm10	69429	75196	786	4.79%	81.37%	71299:252:1
ibm11	70558	81454	373	4.47%	49.76%	29707:240:1
ibm12	71076	77240	651	6.42%	73.00%	74256:152:1
ibm13	84199	99666	424	4.22%	47.64%	33088:240:1
ibm14	147605	152772	614	1.98%	26.72%	17860:144:1
ibm15	161570	186608	393	**10.99%**	43.34%	125562:240:1
ibm16	183484	190048	458	1.89%	48.71%	31093:252:1
ibm17	185495	189581	760	0.94%	23.78%	12441:252:1
ibm18	210613	201920	285	0.96%	11.96%	10152:243:1

Table III. **Benchmark characteristics. Column A_m^b shows the area of the largest macro in the design as % of the total cell area. Column ΣA_m shows the total area of the macros as % of the total cell area. Column $A_m^b : A_m^s : A_c^s$ shows the ratios of areas of the biggest macro to the smallest macro to the smallest standard-cell in the design.**

a whitespace of 20% and their pads (marked in the original IBM netlists) were randomly placed near the perimeter of the core area. We converted the newly created benchmarks to the Cadence LEF/DEF format and applied Cadence's standard-cell placer QPlace to them.

5.2 Flow 1

Statistics for the new benchmarks are given in Table III, together with performance results of our Flow 1 with the Capo placer [Caldwell et al. 2000a] and Parquet fixed-outline floorplanner [Adya and Markov 2001]. [4] We detail runtimes of each step in our proposed design flow. The performance of the industry placer QPlace is given in the same table for comparison. Our flow improves wirelength by 10-50% on most benchmarks.

The complexity of the problem increases with the number of macros and their relative size. According to Table IV, the benchmarks with relatively large macros (`ibm02`, `ibm03`, `ibm04`, `ibm08` and `ibm15`) are difficult for QPlace.[5]

In our Flow 1 the bottleneck is the fixed-outline floorplanning stage, namely in the wirelength computation that is performed independently for every move within the Simulated Annealing framework. While the number of nets in large netlists is typically proportional to the number of cells, many of those nets are not internal to physical clusters which serve as blocks during fixed-outline floorplanning. In other words, physical clustering reduces the number of movable objects much more than the number of nets. Since the relative white-space in the designs that we created was fairly small (20%), the fixed-outline floorplanner take more time to satisfy the fixed-outline constraints. For less constrained designs with more white-space the run-times for the floorplanning stage can be significantly improved.

For benchmarks `ibm01`, `ibm17` and `ibm18`, QPlace results are superior to our flow in terms of runtime. We believe that this is because the macros in these benchmarks are

[4] The C++ source code of Parquet is available on the Web at `http://vlsicad.eecs.umich.edu/BK/parquet/`

[5] We hope that extending QPlace with our proposed techniques can improve results for some circuits.

Ckt	QPlace (ver. 5.1.67) A		Flow 1 = Capo+ Parquet + Capo (High-Temperature Annealing) %Final Overlap=0 B (Uniform WS)						Improved Flow 1 = Capo + Parquet + Capo (Low-Temperature Annealing) %Final Overlap=0 C (Uniform WS)			D (Filler Cells+Uniform WS)		
	Final WL (e6)	Total Time	Final WL (e6)	Total Time	Shred Place Time	FP Time	FP # Tries	Final Place Time	Final WL (e6)	Total Time	#FP Tries	Final WL (e6)	Total Time	#FP Tries
ibm01	3.41	5m	3.66	15m	4m	10m	1	1m	3.36	13m	1	3.05	20m	4
ibm02	10.27	1hr13m	8.73	21m	8m	11m	1	2m	8.23	4hr0m	15	6.83	11m	1
ibm03	19.61	2hr23m	12.50	27m	7m	17m	1	3m	11.53	22m	1	10.38	59m	6
ibm04	37.31	6hr31m	12.99	31m	10m	18m	1	3m	11.93	25m	1	10.11	15m	1
ibm05	12.09	8m	11.20	5m	-	-	-	5m	11.20	5m	-	11.1	5m	-
ibm06	16.03	1hr16m	9.96	24m	10m	9m	1	5m	9.63	19m	1	9.94	18m	1
ibm07	22.04	1hr4m	16.37	47m	16m	23m	1	8m	15.80	39m	1	15.25	25m	1
ibm08	24.69	2hr36m	19.57	48m	16m	24m	1	8m	18.85	1hr51m	3	17.91	29m	1
ibm09	36.28	4hr10m	19.45	3hr23m	21m	2hr53m	5	9m	17.52	2hr58m	6	19.88	29m	1
ibm10	61.07	6hr21m	59.73	3hr21m	54m	2hr11m	1	16m	53.58	8hr10m	3	45.46	1hr56m	1
ibm11	42.73	3hr34m	29.43	1hr24m	28m	41m	1	15m	26.47	1hr9m	1	29.4	45m	1
ibm12	71.69	7hr5m	59.16	2hr31m	42m	1hr33m	1	16m	55.12	1hr59m	1	51.1	1hr35m	1
ibm13	59.80	5hr20m	36.08	3hr24m	35m	2hr31m	2	18m	33.56	1hr28m	1	37.73	53m	1
ibm14	52.06	1hr32m	58.49	3hr21m	53m	1hr52m	1	36m	52.67	5hr33m	2	50.26	2hr35m	1
ibm15	126.26	15hr49m	69.05	3hr51m	65m	1hr56m	1	50m	64.69	4hr24m	2	65.0	3hr15m	1
ibm16	216.16	44hr46m	93.45	4hr14m	1hr53m	1hr29m	1	52m	83.14	9hr40m	4	82.9	2hr42m	2
ibm17	89.13	1hr36m	98.23	6hr1m	1hr40m	3hr18m	1	1hr3m	91.50	4hr9m	1	89.17	3hr8m	1
ibm18	58.11	1hr32m	53.70	3hr0m	1hr13m	50m	1	57m	54.11	6hr37m	5	51.84	2hr7m	1

Table IV. **Our Flow 1 (Capo+Parquet+Capo) versus the industry placer QPlace v. 5.1.67. Run-times for QPlace are measured on a 500 MHz Sun Blade 100 system with UltraSPARC-IIe processor; for Capo and Parquet on a 2 GHz Pentium Xeon. Parquet runtime includes all attempts to satisfy the given outline constraints, the number of attempts is shown as well.** `ibm05` **does not require a floorplanning stage (no macros). In Tables IV (C) and (D) we report results for Flow 1, with the floorplanner running in low-temperature annealing mode to preserve initial macro locations. Comparing with results from Table B, we note that for some benchmarks the runtimes increase because of larger number of attempts required to satisfy the fixed-outline constraints. The solution quality for most benchmarks improves with this flow. Table IV (C) shows results for the final placement of standard cells with uniform whitespace distribution. Table IV (D) shows the results for final placement of standard cells along with filler cells for better whitespace handling. Also, in Table IV (D), the areas of clustered macros of standard cells during floorplanning stage is reduced by 10% to make it easier for the floorplanner to satisfy the fixed-outline constraints. Our comparison to QPlace is mostly a sanity check because QPlace is not designed to solve mixed-size placement instances.**

relatively small, and a standard-cell placer may handle them well enough. On the other hand, `ibm17` and `ibm18` are big enough to expose the coarseness of the 6x6 grid used in our experiments. Aside from increasing the grid size, it is possible to extend Capo to handle small macros, and thus entirely avoid running a floorplanner on those benchmarks.

5.2.1 *Sensitivities in Flow 1.* We study the various sensitivities in our flows. In our original Flow 1 the information about the initial locations of macros is not useful and placing the clustered netlist serves as a means to generate high quality clusters. Also, fixed-outline floorplanning stage is a bottleneck in terms of runtime. The results in Table IV B are for the flow in which the floorplanning is done from scratch with random initial solution and no attempt is made to preserve the initial positions of macros. We tried a variant of this flow to in an attempt to maintain the initial macro locations obtained by placing the shredded netlist. We do this by forming a sequence pair from the illegal placement obtained from Step 1 of the flow and then employing low-temperature annealing. The results for

Ckt	Flow 2 = Capo + Kraftwerk ECO			mPG [Chang et al. 2003]		Kraftwerk [Eisenmann and Johannes 1998]			[Choi and Bazargan 2003]		
	A			B		C			D		
	Final WL(e6)	Total Time	% Overlap	Final WL(e6)	Total Time	Final WL(e6)	Total Time	% Overlap	Final WL(e6)	Total Time	% Overlap
ibm01	2.92	5m	0.87	3.01	18m	3.01	2m	3.5	2.78	2m	6.31
ibm02	6.5	11m	0.19	7.42	32m	7.58	9m	6.4	6.64	3m	5.52
ibm03	9.63	14m	0.96	11.2	32m	11.4	10m	1.29	10.4	5m	7.62
ibm04	11.2	14m	0.47	10.5	42m	12.1	12m	1.31	11.7	6m	7.85
ibm05	11.2	5m	0.00	10.9	36m	12.9	3m	0.03	13.1	5m	1.72
ibm06	7.9	17m	0.41	9.2	45m	10.2	12m	0.9	9.89	9m	5.46
ibm07	13.6	55m	0.56	13.7	1hr8m	17.1	19m	0.75	16.2	14m	7.04
ibm08	17.2	22m	0.73	16.4	1hr22m	18.2	21m	7.43	18.0	15m	8.06
ibm09	17.8	31m	0.70	18.6	1hr24m	19.1	28m	0.49	20.2	16m	7.91
ibm10	47.5	1hr8m	2.1	43.6	2hr52m	51.5	35m	7.1	42.3	27m	4.29
ibm11	25.1	41m	0.67	26.5	1hr52m	26.6	36m	0.82	27.6	23m	9.69
ibm12	47.5	51m	0.38	44.3	1hr33m	52.6	43m	10.8	48.0	27m	7.06
ibm13	33.4	1hr8m	0.80	37.7	1hr31m	35.9	55m	0.69	35.4	33m	10.11
ibm14	47.9	1hr57m	0.43	43.5	4hr36m	47.4	1hr14m	0.4	-	-	-
ibm15	66.8	2hr2m	0.34	65.5	6hr25m	73.7	1hr33m	0.6	-	-	-
ibm16	86.7	2hr46m	0.45	72.4	7hr16m	82.4	1hr34m	13.1	-	-	-
ibm17	87.6	2hr16m	0.38	78.5	10hr6m	92.2	1hr47m	0.35	-	-	-
ibm18	57.2	2hr38m	0.08	50.7	7hr17m	54.9	1hr50m	0.09	-	-	-

Table V. **Results for our Flow 2. On average the results are better than Flow 1 in Table IV. However some overlaps remain in the final placement. The % overlaps is shown. We compare our results with mPG, Kraftwerk and a mixed-size placement flow. Runtimes for Flow 2(A) and Kraftwerk (C) are observed on 2 GHz Linux/Pentium machine. Runtimes for mPG (B) are observed on Sun Blade 1000 workstations running at 750 MHz. Runtimes for the mixed-size placement Flow (D) are observed on 900MHz Linux/Pentium machine.**

this flow is presented in Table IV C. For some benchmarks (`ibm02`, `ibm09` and `ibm10`), the floorplanner requires more tries to satisfy the fixed-outline constraints because in the low-temperature annealing mode it is trying to massage an existing solution and the hill climbing capabilities of simulated annealing do not work as efficiently. As a result the total runtimes for these designs increase. The final HPWL for most designs improve, but not significantly. We conclude that it is useful to preserve the initial positions of macros, especially in less area-constrained designs.

We try another variant of the low-temperature annealing flow in an attempt to reduce the floorplanning overhead. When forming soft clusters of standard cells using physical clustering, we reduce the area of the clustered soft block by 10%. Thus the area of the clustered block is 0.9 * (sum of areas of sub-cells). This helps the fixed-outline floorplanner to find a solution that satisfies fixed-outline constraints faster. However, reducing the area of the clustered cells might affect the optimization in some cases. Step 4 of the flow fixes the macro locations to the ones provided by the floorplanner and replaces standard-cells around the macros. The standard cells are placed around the macros and the whitespace in the design is allocated uniformly around the chip. However we can improve the wirelength of the design by improved whitespace allocation. We introduce unconnected filler cells in the design to represent the excessive whitespace and reduce the whitespace available to the placer to 10%. The design is then replaced with the macros being fixed. Thus the standard cells are placed more compactly, resulting in improved wirelength. The results for this variant of the flow are presented in Table IV D. As seen, wirelength and runtimes improve for most designs.

5.3 Flow 2

Table V shows the results for our Flow 2 which places the shredded netlist using Capo to create an initial placement and then uses Kraftwerk in ECO mode to remove the overlaps while changing the initial seed placement as little as possible. For these results we also report the overlap remaining in the placement as a percent of the layout area. This is because, the Flow 2 does not produce completely overlap free placements. The results for this flow are on average better than our Flow 1, but Flow 1 is guaranteed to produce completely overlap free placements. We also compare our results with placements generated by Kraftwerk [Eisenmann and Johannes 1998] from scratch. As seen in Table V, Kraftwerk frequently produces placements with large overlaps. Our proposed Flow 2 produces much better placements in terms of wirelength and overlaps than Kraftwerk run from scratch. We also compare our results to mPG [Chang et al. 2003] and the mixed-size placement flow [Choi and Bazargan 2003]. Note that the flow in [Choi and Bazargan 2003] produces placements with large overlaps.

The problem with Flow 1 is that some of its steps ignore information produced by previous steps. The macro locations generated by placing the shredded netlist of step 1 are discarded. Also the soft macro locations obtained by the floorplanning stage are discarded in the final placement. Flow 2, which uses force-directed techniques to legalize placements obtained from step 1 overcomes this problem. An up-coming work [Khatkhate et al. 2004] studies legalization of such mixed-size placements with minimal movement from the original locations. However, the methods proposed in [Khatkhate et al. 2004] produce a placement that is packed to the left side.

6. ONGOING WORK

We are currently working on tight integration of floorplanning and placement techniques to handle mixed-size designs. We have integrated the fixed-outline floorplanner Parquet [Adya and Markov 2003] [Adya and Markov 2001] with the top-down placer Capo [Caldwell et al. 2000a] based on recursive bisection, seeking to improve scalability when handling mixed-size designs. Capo's framework is briefly described in Section 2.1. We modify this framework to handle mixed-size designs as follows. The large macros are initially treated as normal placeable cells and the placement block is processed in the regular fashion. Fixed-outline floorplanning is employed to place the macros at legal locations inside the placement block if atleast one of the following conditions is satisfied.

— The placement block has atleast one large macro whose height/width is greater than a certain fraction (in our case 1/4) of the block's height/width.

— The total area of the macros in the placement block is greater than a certain threshold (80% in our case) of the total cell area and the number of macros in the placement block is less than 100.

In order to use the floorplanner, a floorplanning instance is formed by clustering the standard cells with highest connectivity in a bottom-up fashion as explained in Section 3.2. The macros identified before are not clustered. The fixed-outline constraints are derived from the placement block's dimensions. If the fixed-outline floorplanning is successful the macros are fixed at legal locations provided by the floorplanner, the sites are removed below the fixed macros and the macros are removed from the block. From now on the placement block is processed as a normal placement block which has only standard-cells.

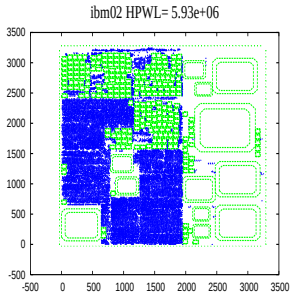


Fig. 13. **ibm02 placement by our integrated placement and floorplanning flow. The macros are marked by double lines.**

In theory, this proposed flow is a correct-by-construction approach and will produce a legal placement assuming that fixed-outline floorplanning succeeds in all cases. However, in our current implementation the fixed-outline floorplanning sometimes fails to find a legal solution satisfying the fixed-outline constraints. We believe that this is due to the difficulties in integrating recursive bisection and fixed-outline floorplanning. Figure 13 shows the placement of ibm02 design obtained using this strategy. We present preliminary results for this flow in Figure VI. Comparing with Tables IV and V, we see that, in most cases, this flow produces better HPWL placements and requires less run-time. In some cases final placements have overlaps. Since these overlaps tend to be extremely small, they can be removed by techniques from [Khatkhate et al. 2004]. However, [Khatkhate et al. 2004] requires left-packing of the placement which we would like to avoid [Adya et al. 2003], because from our experience it is likely to cause routability problems. The fact that [Khatkhate et al. 2004] also uses the recursive partitioning approach and achieves lower wirelength than reported in this paper suggests that more work is needed on mixed-size placement.

7. CONCLUSIONS

Modern SoC designs entail placement instances with numerous design IP blocks. Handling such layout problems has become important, and our work addresses this problem. Floorplanning techniques handle designs with macros, but do not scale to a hundred thousand standard cells. On the other hand, standard-cell placers handle large numbers of small, fixed-height cells, but do not handle macros very well. Therefore, we attempt to combine the strengths of both techniques.

We propose two design flows to place macro cells consistently with large numbers of standard cells. The first flow uses a combination of techniques from standard-cell placement and fixed-outline floorplanning. In particular, a number of existing placers can be used without source code modifications. Our proposed Flow 1 can be summarized as follows:

— Shred the original netlist. Use a standard-cell placer to generate an initial placement.
— Construct a floorplanning instance using a physical clustering algorithm.
— Generate valid locations of macros with an improved fixed-outline floorplanner.
— Fix the macros and place the remaining standard cells.

Ckt	Flow 3(Capo)		
	WL(e6)	Time	%Overlap
ibm01	2.96	1.8m	0
ibm02	5.93	4.6m	0
ibm03	9.74	6.8m	0
ibm04	10.6	8.0m	0
ibm05	11.0	4.6m	0
ibm06	7.35	7.1m	0
ibm07	12.5	17.4m	0.0002
ibm08	15.6	2hr26m	0
ibm09	17.1	12m	0.007
ibm10	37.4	2hr13m	0.006
ibm11	23.6	16m	0.0001
ibm12	44.5	58.8m	0.14
ibm13	31.0	22m	0
ibm14	44.5	38.5m	0.005
ibm15	61.1	56.9m	0
ibm16	70.5	60.2m	0.03
ibm17	80.6	45.9m	0.01
ibm18	50.9	44.6m	0.0005

Table VI. **Results for integrated placement and floorplanning flow. We report HPWL, run-times and the final overlap between macros as a percentage of the total layout area. Run-times are observed on 2 GHz Linux/Pentium machine. Designs ibm08 and ibm10 take relatively longer times because of multiple floorplanning attempts to satisfy fixed-outline constraints during the placement flow.**

This flow can be modified to include a human designer who uses the initial placement as a hint when manually placing macros. Alternatively, variants of this flow can better preserve the initial placement.

Our proposed Flow 2 combines a standard cell placer with force-directed techniques and can be summarized as follows:

— Shred the original netlist. Use a standard-cell placer to generate an initial placement.

— Use a force-directed placer in ECO mode to remove the overlaps while trying to minimize the change in existing placement.

Our Flow 1 produces completely overlap-free placements with reasonably good wirelengths. Our Flow 2 produces high quality wirelength placements with potentially some overlaps. However, these overlaps are generally very small and can be removed by simple techniques. Either of our flows can be applied for mixed-size design placement depending upon the requirements and characteristics of the design. Our empirical results for mixed-size placement are significantly better than those produced by the Cadence placer QPlace. Our results also compare favorably to those in [Chang et al. 2003] and [Choi and Bazargan 2003]. It should be noted however, that the multi-level techniques in [Chang et al. 2003] are very different from those used by other researchers and can, in principle, be combined with ours or even applied to placements produced by our methods.

Our experiments show that the proposed flows scale up to at least a thousand macros in addition to hundreds of thousands standard cells. However, in Flow 1, floorplanning instances with a thousand blocks is a bottleneck and may be improved further. Our on-going work focuses on techniques for incremental wirelength computation as well as multi-level techniques for floorplanning that can handle greater numbers of macros. We have not explicitly considered timing and congestion, but the significant improvements in wirelength obtained suggest that those metrics can also improve. Moreover, if an objective function

can be quickly computed (e.g., circuit delay without false paths can be computed by static timing analysis in linear time), its optimization can be quickly added to simulated annealing that we use for floorplanning. Alternatively, one could use a previously reported force-directed macro block placer [Mo et al. 2000] that handles congestion. Congestion and timing can also be addressed at the second call to a black-box placer, assuming that the placer has relevant functionalities [M. Wang and Sarrafzadeh 2000; Kahng et al. 2002]. Our focus on half-perimeter wirelength is also due to our belief that any large-scale layout tool that cannot successfully optimize wirelength is not going to successfully optimize more complex objectives. Our work can be considered a first step in this direction.

REFERENCES

ADYA, S. N. AND MARKOV, I. L. 2001. Fixed-outline floorplanning through better local search. In *Proceedings of the International Conference on Computer Design*. IEEE, Austin, 328–334.

ADYA, S. N. AND MARKOV, I. L. 2002. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of the International Symposium on Physical Design*. ACM, San Diego, 12–17.

ADYA, S. N. AND MARKOV, I. L. 2003. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Transactions on VLSI Systems 11*, 6 (Dec.), 1120–1135.

ADYA, S. N., MARKOV, I. L., AND VILLARRUBIA, P. G. 2003. On whitespace and stability in mixed-size placement and physical synthesis. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 311–318.

ADYA, S. N., YILDIZ, M., MARKOV, I. L., VILLARRUBIA, P. G., PARAKH, P. N., AND MADDEN, P. H. 2003. Benchmarking for large-scale placement and beyond. In *Proceedings of the International Symposium on Physical Design*. ACM, Monterey, 95–103.

ADYA, S. N., YILDIZ, M., MARKOV, I. L., VILLARRUBIA, P. G., PARAKH, P. N., AND MADDEN, P. H. 2004. Benchmarking for large-scale placement and beyond. *To appear in IEEE Transactions on CAD*.

ALPERT, C. J. 1998. The ISPD98 circuit benchmark suite. In *Proceedings of the International Symposium on Physical Design*, `http://vlsicad.cs.ucla.edu/~cheese/ispd98.html` . ACM, Monterey, 80–85.

ALPERT, C. J., HUANG, J., AND KAHNG, A. B. 1997. Multilevel circuit partitioning. In *Proceedings of the Design Automation Conference*. ACM, Anaheim, 530–533.

ALPERT, C. J., NAM, G. J., AND VILLARRUBIA, P. G. 2002. Free space management for cut-based placement. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 746–751.

CADENCE. Openbook documentation for QPlace version 5.1.67, 10/27/2000.

CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. VLSI CAD Bookshelf. In `http://vlsicad.eecs.umich.edu/BK` .

CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. 2000a. Can recursive bisection alone produce routable placements? In *Proceedings of the Design Automation Conference*. ACM, Los Angeles, 477–482.

CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. 2000b. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Transactions on CAD 19*, 11, 1304–1314.

CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. 2003. Hierarchical whitespace allocation in top-down placement. *IEEE Transactions on CAD 22*, 11 (Nov.), 716–724.

CHANG, C., CONG, J., AND YUAN, X. 2003. Multi-level placement for large-scale mixed-size ic designs. In *Proceedings of the ASPDAC*. IEEE, KitaKyushu/Japan, 325–330.

CHOI, W. AND BAZARGAN, K. 2003. Hierarchical global floorplacement using simulated annealing and network flow area migration. In *Proceedings of the DATE*. IEEE, Munich, 1104–1105.

DALLY, W. J. AND CHANG, A. 2000. The role of custom design in asic chips. In *Proceedings of the Design Automation Conference*. ACM, Los Angeles, 643–647.

DOLL, K., JOHANNES, F. M., AND ANTREICH, K. J. 1994. Iterative placement improvement by network flow methods. *IEEE Transactions on CAD 13*, 10 (Oct.), 1189–1200.

DUTT, S. 2000. Effective partition-driven placement with simultaneous level processing and a global net view. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 254–259.

EISENMANN, H. AND JOHANNES, F. M. 1998. Generic global placement and floorplanning. In *Proceedings of the Design Automation Conference*. ACM, San Francisco, 269–274.

FUJUYOSHI, K. AND MURATA, H. 1999. Arbitrary convex and concave rectilinear block packing using sequence pair. In *Proceedings of the International Symposium on Physical Design*. ACM, Monterey, 103–110.

HONG, X., HUANG, G., CAI, Y., GU, J., DONG, S., CHENG, C.-K., AND GU, J. 2000. Corner Block List : An effective and efficient topological representation of non-slicing floorplan. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 8–13.

KAHNG, A. B. 2000. Classical floorplanning harmful? In *Proceedings of the International Symposium on Physical Design*. ACM, San Diego, 207–213.

KAHNG, A. B., MANTIK, S., AND MARKOV, I. L. 2002. Min-max placement for large-scale timing optimization. In *Proceedings of the International Symposium on Physical Design*. ACM, San Diego, 143–148.

KARYPIS, G., AGARWAL, R., KUMAR, V., AND SHEKHAR, S. 1997. Multilevel hypergraph partitioning: Applications in vlsi design. In *Proceedings of the Design Automation Conference*. ACM, Anaheim, 526–529.

KHATKHATE, A., AGNIHOTRI, A. R., YILDIZ, M. C., AND MADDEN, P. H. 2004. Recursive bisection based mixed block placement. In *To appear in the Proceedings of the International Symposium on Physical Design*. ACM, Arizona.

LIN, J. AND CHANG, Y. 2001. TCG : A transitive closure graph based representation for non-slicing floorplans. In *Proceedings of the Design Automation Conference*. ACM, Las Vegas, 764–769.

M. WANG, X. Y. AND SARRAFZADEH, M. 2000. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 260–263.

MO, F., TABBARA, A., AND BRAYTON, R. K. 2000. A force-directed macro-cell placer. In *Proceedings of the International Conference on Computer Aided Design*. ACM, San Jose, 404–407.

MURATA, H., FUJIYOSHI, K., NAKATAKE, S., AND KAJITANI, Y. 1996. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Transactions on CAD 15*, 12, 1518–1524.

MURATA, H. AND KUH, E. S. 1998. Sequence-pair based placement methods for hard/soft/pre-placed modules. In *Proceedings of the International Symposium on Physical Design*. ACM, Monterey, 167–172.

NAG, S. AND CHAUDHARY, K. 1999. Post-placement residual-overlap removal with minimal movement. In *Proceedings of the DATE*. IEEE, Munich, 581–586.

PANG, Y., CHENG, C., AND YOSHIMURA, T. 2000. An enhanced perturbing algorithm for floorplan design using the o-tree representation. In *Proceedings of the International Symposium on Physical Design*. ACM, San Diego, 168–173.

SARRAFZADEH, M., WANG, M., AND YANG, X. 2002. *Modern Placement Techniques*. Kluwer.

SHERWANI, N. 1999. *Algorithms for VLSI Physical Design Automation*. Kluwer.

TANG, X., TIAN, R., AND WONG, D. F. 2000. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of the DATE*. IEEE, Paris, 106–111.

TANG, X. AND WONG, D. F. 2001. FAST-SP: A fast algorithm for block placement based on sequence pair. In *Proceedings of the ASPDAC*. IEEE, Yokohama, 521–526.

VARADRAJAN, R. AND DELENDONCK, G. 2002. Personal communication.

VIJAYAN, G. 1991. Overlap elimination in floorplans. In *Proceedings of the VLSI Design*. IEEE, 157–162.

YILDIZ, M. C. AND MADDEN, P. H. 2001. Improved cut sequences for partitioning based placement. In *Proceedings of the Design Automation Conference*. ACM, Las Vegas, 776–779.