Consistent Placement of Macro-Blocks Using Floorplanning and Standard-Cell Placement

Saurabh N. Adya and Igor L. Markov
University of Michigan, EECS Department, Ann Arbor, MI 48109-2122
{sadya,imarkov}@eecs.umich.edu

ABSTRACT

While a number of recent works address large-scale standard-cell placement, they 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 in a design flow that solves the more general placement problem. Our work shows how to place macros consistently with large numbers of small standard cells. Our techniques can also be used to guide circuit designers who prefer to place macros by hand.

The proposed 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. Empirical evaluation on ibm benchmarks shows, in most cases, wirelength improvements of 10%-50% compared to Cadence QPlace, as well as runtime improvements.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – Layout, Placement and Routing

General Terms

Algorithms, Design

1. INTRODUCTION

During the last few decades, academia and industry have invested considerable effort in research on Physical Design for VLSI [15]. Through the integration of multiple optimization techniques, design methods and high-performance CAD software for integrated circuits (ICs) were developed. However, the ever-increasing size of ICs lead to frequent changes to common design flows. Recently, design reuse was introduced as a way to (i) tame the complexity of circuit design for deep submicron 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, 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 logics with known function, geometric and electrical properties, but no structural description of their inner workings. Such macros may or may not be flexible, but in any case are considered parts of design. Unfortunately, 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 [4] 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. In classical Physical Design flows a circuit is first partitioned, then floorplanned, and finally, standard-cell placement is applied to the partitions. 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 [5] after the multi-level partitioning breakthrough in 1997 [2, 11]. In addition to having near-linear runtime, placers based on recursive bisection *perform circuit partitioning* and, if the cutlines are allowed to move, also *perform floorplaning*. 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.

The main contribution of this paper 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 [1], improved to minimize wirelength, 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., [12]. Indeed, we do not insist on using a particular floorplan representation, but rather emphasize floorplanning as a step in large-scale placement with macros.

ISPD'02, April 7-10, 2002, San Diego, California, USA.

We notice that existing academic placers Capo [5], Dragon 2000 [18], Feng Shui[19] and Spade [9] cannot process movable macros. In fact, *all macros are removed* in the placement benchmarks described in [18] (produced from the ISPD 98 circuit benchmarks), and all cells are artificially made 1-by-1. Therefore, we derived new placement benchmarks from the same circuits, preserving macros and the areas of all cells. Using those benchmarks, we compared the performance of our methods to a major commercial placer.

The remaining part of the paper is organized as follows. Section 2 covers previous work relevant to fixed-outline floorplanning. It also introduces our contributions in wirelength minimization and the handling of soft blocks. A new design flow for macro placement is proposed in Section 3. Section 4 presents empirical validation of our work, and future directions are discussed in Section 5.

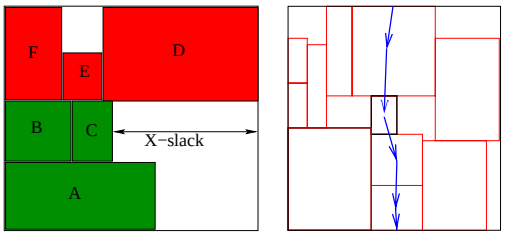

Figure 1: In figure (a) X-slack of blocks B and C is shown by the solid arrow. In figure (b) blocks with zero Y-slack are shown. They lie on a "critical path" marked with arrows.

2. FIXED-OUTLINE FLOORPLANNING

2.1 Previous Work

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 [15, 10] 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 [10]. In floorplanners based on Simulated Annealing (e.g., with the Sequence Pair representation [14]) the typical choice of moves is fairly straightforward.

As pointed out in [10, 5], 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 [15], 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 [10] 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.

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* [1]. As illustrated in Figure 1 (a), 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 floorplan. Blocks with zero slack in the Y dimension are shown in Figure 1 (b). Such blocks must lie on critical paths in the relevant constraint graph.

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 slack of a given block in a given dimension is the difference between the block's locations produced by those two packings. The floorplanner Parquet [1] uses the Sequence Pair representation [14] because of its simplicity.

Once slacks 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 pair-wise swaps and block rotations that are typically used in Sequence-Pair based annealers. Observe that 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 [16, 17]. 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 two zero slacks, 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 white space can be created around L.

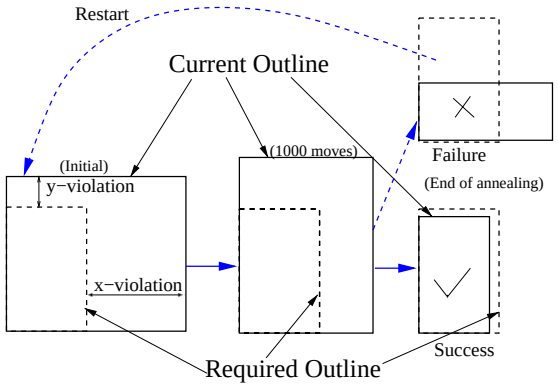

Figure 2: Snap-shots from fixed-outline floorplanning. The number of annealing moves is fixed, but if the evolving floorplan fits within the required fixed-outline, annealing is stopped earlier. If at the end of annealing the fixed-outline constraints are not satisfied, it is considered a failure and a fresh attempt is made.

Figure 2 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 [1] empirically demonstrate high ratios of successes to failures in the flow from Figure 2.

2.2 Wirelength Minimization

In fixed-outline floorplanning, the global objective is to minimize the wirelength of the design subject to fixed-outline constraints. In our floorplanner we use a linear combination of area and HPWL to evaluate annealer moves. The area term is normalized by the total area of all blocks, and the wirelength term is normalized by the current wirelength of the floorplan at every move.

Additional moves are designed to improve the wirelength. For a given block a, we calculate, using analytical techniques, its "ideal" location that would minimize quadratic wirelength of its incident wires.[1] We then identify the block b closest to that location and attempt to move a in the sequence pair so that in both sequences it is located next to b. We evaluate the four possible ways to do that, and choose the best. The following example illustrates moving a block close to another by manipulating the sequence pair.

Example: Consider the five-block sequence pair $<<a,b,c,d,e> <c,a,d,e,b>>$. We wish to move block **e** close to block **a** in the floorplan. This can be done in four ways:

- $<\mathbf{a},\mathbf{e},b,c,d>,<c,\mathbf{a},\mathbf{e},d,b>$ (e is to the right of a)
- $<\mathbf{e},\mathbf{a},b,c,d>,<c,\mathbf{e},\mathbf{a},d,b>$ (e is to the left of a)
- $<\mathbf{a},\mathbf{e},b,c,d>,<c,\mathbf{e},\mathbf{a},d,b>$ (e is below a)
- $<\mathbf{e},\mathbf{a},b,c,d>,<c,\mathbf{a},\mathbf{e},d,b>$ (e is above a)

Another type of move attempts to minimize both the floorplan size and wirelength objectives at the same time. Find a block b closest to the ideal location of the chosen block a such that the block b has large slack in at least one dimension. Depending on whether b has a large slack in the X-dimension or in the Y-dimension, we place a with a horizontal relation or a vertical constraint relative to b, respectively. Empirical measurements confirm that adding the proposed move types improves final floorplans.

2.3 Handling Soft Blocks

We also added slack-based move types to change aspect ratios of soft blocks. During annealing, at regular intervals, a block with low (preferably zero) slack in one dimension and high 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.

3. MACRO PLACEMENT FLOW

Our proposed flow 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 datamodel implied by Cadence LEF/DEF. We also require that the placer can handle fixed cells/pins and can handle rows consisting of contiguous subrows. By removing cell sites from a subrow and splitting the subrow into two subrows, one can model the effect of fixed macros (because pins of fixed macros are fixed as well). For example, the site map in Figure 3 corresponds to the placement in Figure 6 (c). Our flow also uses a fixed-outline floorplanner, of the type described above. A variety of floorplan representations can be used.

[1] Analytical techniques are used because they are fast and easy to implement.

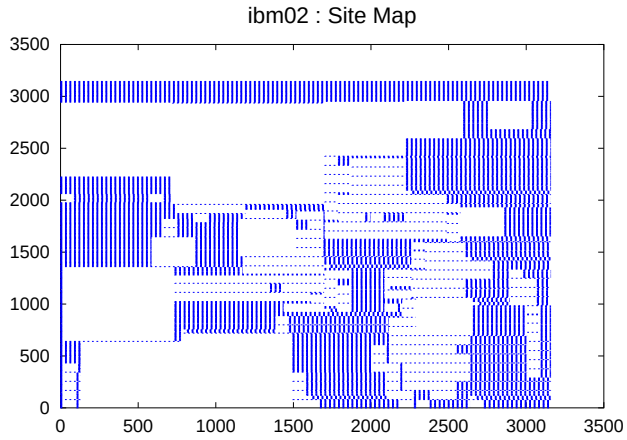

Figure 3: Map of cell sites for the ibm02 design with all the macros marked as fixed. Sites under the macros are removed.

3.1 Shredding Macro Cells

The DOMINO detailed placer [8] introduced the idea of shredding big cells to simplify placement. 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 at 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 4. 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. 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 [5] we use is scalable enough.

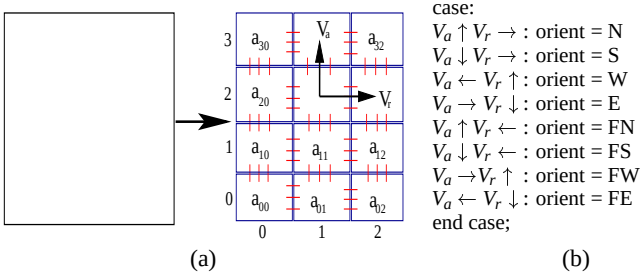

Figure 4: A macro is shredded into cells of minimal height, connected by fake wires. To find the orientation of the macro from locations of sub-cells, the relative locations of sub-cells $a_{i,j}$, $a_{i+1,j}$ and $a_{i,j+1}$ are analyzed for every eligible (i,j). Figure (b) shows the case analysis in terms of vectors V_a and V_r in final placement. "F" stands for "flipped".

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 all sub-cells of that macro. Additionally, we developed a heuristic to determine the orientation of the macro. The heuristic is

based on the relative placement of each 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 4, 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. 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 6 (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 5 (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 proof of the lemma is summarized in Figure 5. 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.

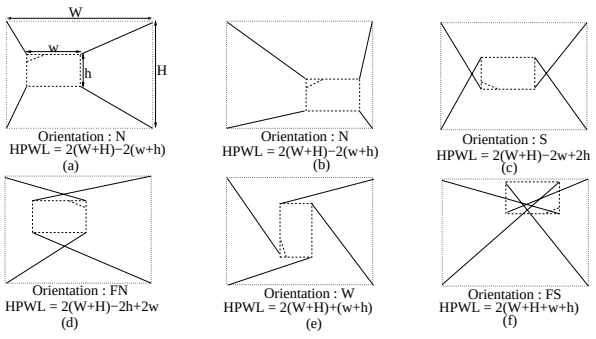

Figure 5: Five out of eight orientations of a macro whose corners are tied to the corners of the layout region; the orientation is N in (a) and (b). The [linear] length of faked wires depends only on the orientation and not on the location of the macro, as long as the macro is placed entirely within the layout region. The desired orientation (N in this example) is found by wirelength minimization.

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 6 (a)). Such violations must be corrected without increasing wirelength. This can, in principle, be done by fixed-outline floorplanning, but the number of movable objects is unrealistically high (every standard cell is movable). We therefore construct a fixed-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 [2, 11]. We believe that our physical clustering based on the initial placement accounts for both netlist connectivity and the shapes of macros. 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.

3.3 Fixed-outline Floorplanning With Macros

As explained in Section 2.2, we extend the fixed-outline floorplanner Parquet [1] to minimize wirelength and handle soft blocks. 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 6 (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 [13]. Alternatively, one could tie those macros with faked wires to faked pins placed in strategic locations.

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 [5], followed by two passes of window-based branch-and-bound placement.[2] Figure 6 (c) shows the final placement generated by our proposed flow for the ibm02 design.

4. RESULTS

Our proposed flow is implemented in C++ and compiled by `g++ 2.95.4 -O3`. Runtimes are measured on a 1GHz PC/Intel system runnning Linux. We compare our results against QPlace v.5.1.67 from Cadence, whose runtimes are measured on a 440 MHz Sun Ultra-5 system running Solaris. The Sun workstation has a larger cache, and the two hardware platforms are within a factor of two

[2]As the detailed placement step, we apply branch-and-bound end-case placers [6] using sliding windows.

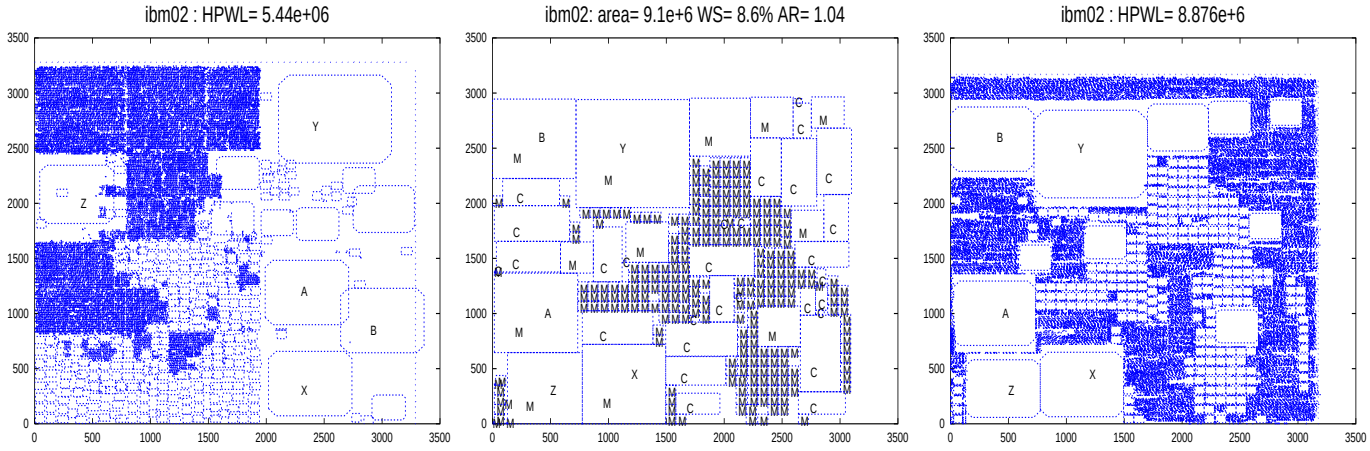

Figure 6: Figure (a) is the placement (illegal) obtained after running Capo on ibm02 design with macros shredded into small cells. The locations of macros are determined by averaging the locations of sub-cells. Note that macro B is not placed entirely within the layout region and overlaps with macro A. Macro Z overlaps with smaller macros and standard cells. Figure (b) shows a possible final fixed-outline floorplan of the same design. 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 four (A, X, Y and Z) 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).

by overall performance.

The benchmarks used in our experiments are derived from the ISPD-98 (IBM) circuit benchmarks [3]. We converted the netlists into the Bookshelf placement format [7], added placement-related information and made the new benchmarks available on the Web at `http://vlsicad.eecs.umich.edu/BK/ISPD02bench/` 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 a whitespace of 15% 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.

Statistics for the new benchmarks are given in Table 1, together with performance results of our flow with the Capo placer [5] and a version of the Parquet fixed-outline floorplanner [1] improved to minimize wirelength and handle soft blocks.[3] 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 and achieves better runtime.

The complexity of the problem increases with the number of macros and their relative size. Indeed, QPlace timed out after *twenty four hours* on benchmarks with many large macros, e.g., `ibm04` and `ibm09`. We consider this a failure. According to Table 1, the benchmarks with relatively large macros (`ibm02`, `ibm03`, `ibm08` and `ibm15`) are difficult for QPlace.[4]

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

[4]QPlace can be easily improved by implementing our techniques.

In our flow 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.

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 relatively small, and a standard-cell 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. CONCLUSIONS AND FUTURE WORK

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 take the best of both techniques.

Our work points out that the handling of movable macros in existing commercial placement tools can be considerably improved, especially in terms of scalability. We propose a design flow to place macro cells consistently with large numbers of standard cells. This 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. However, we had to improve the state of the art in fixed-outline floorplanning [1] by adding new moves to handle soft blocks and drive wirelength minimization more efficiently. Our proposed flow can be summarized as follows:

- Use a standard-cell placer to generate an initial placement.
- Construct a floorplanning instance using a physical clustering algorithm.

Circuit	# Nodes	# Nets	# Macros	Area of biggest macro	QPlace HPWL	QPlace Time (24h limit)	Our Flow = Capo + Parquet + Capo Final HPWL	Total time	ShredPlace time	Floorplan time	Floorplan #tries	Place time
ibm01	12752	14111	246	6.37%	4.01e6	**6m**	**3.96e6**	18m	6m	11m	1	1m
ibm02	19601	19584	280	**11.36%**	16.64e6	1hr19m	**8.37e6**	**31m**	10m	18m	1	3m
ibm03	23136	27401	290	**10.75%**	time-out	time-out	**12.16e6**	**42m**	12m	26m	1	4m
ibm04	27507	31970	608	9.15%	time-out	time-out	**13.48e6**	**47m**	12m	30m	1	5m
ibm05	29347	28446	0	-	11.70e6	**7m**	**11.51e6**	8m	-	-	-	8m
ibm06	32498	34826	178	3.95%	18.00e6	2hr4m	**10.25e6**	**56m**	11m	39m	3	6m
ibm07	45926	48117	507	4.75%	28.10e6	1hr45m	**15.75e6**	**58m**	18m	31m	1	9m
ibm08	51309	50513	309	**12.10%**	26.62e6	3hr47m	**21.18e6**	**1hr34m**	18m	1hr6m	1	10m
ibm09	53395	60902	253	5.42%	time-out	time-out	**19.59e6**	**1hr6m**	23m	30m	1	14m
ibm10	69429	75196	786	4.79%	time-out	time-out	**60.72e6**	**3hr49m**	55m	2hr40m	1	14m
ibm11	70558	81454	373	4.47%	time-out	time-out	**28.49e6**	**1hr46m**	32m	58m	1	16m
ibm12	71076	77240	651	6.42%	time-out	time-out	**51.74e6**	**11hr15m**	39m	10hr20m	4	16m
ibm13	84199	99666	424	4.22%	time-out	time-out	**39.39e6**	**2hr31m**	40m	1hr31m	1	20m
ibm14	147605	152772	614	1.98%	time-out	time-out	**56.19e6**	**4hr46m**	58m	3hr11m	1	37m
ibm15	161570	186608	393	**10.99%**	time-out	time-out	**70.48e6**	**3hr57m**	58m	2hr14m	1	45m
ibm16	183484	190048	458	1.89%	time-out	time-out	79.59e6	3hr13m	25m	1hr51m	1	57m
ibm17	185495	189581	760	0.94%	**89.34e6**	**2hr24m**	92.38e6	7hr23m	1hr30m	6hr0m	1	53m
ibm18	210613	201920	285	0.96%	56.43e6	**1hr29m**	**54.90e6**	5hr78m	1hr9m	3hr12m	2	57m

Table 1: Our flow (Capo+Parquet+Capo) versus the industry placer QPlace v. 5.1.67. Run-times for QPlace are measured on a 440 MHz Sun Ultra-5 system; for Capo and Parquet — on a 1 GHz Intel Pentium-III. Parquet runtime includes all attempts to satisfy the given outline constraints, the number of attempts is shown as well. We do not run Parquet on `ibm05` because `ibm05` has no macros.

- Generate valid locations of macros with an improved fixed-outline floorplanner.
- Fix the macros and place the remaining standard cells.

This flow can be modified to add 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 experiments show that the proposed flow scales up to at least a thousand macros in addition to hundreds of thousands standard cells. However, 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.

Finally, we point out that Cadence recommended flow that includes a separate macro placer may perform better on our benchmarks. However, the fact that QPlace *attempts* to place macros and often does a poor job suggests a possible improvement.

6. ACKNOWLEDGEMENTS

This work was supported by a gift from IBM, a DAC graduate scholarship and a grant from the DARPA/MARCO Gigascale Silicon Research Center.

7. REFERENCES

[1] S. N. Adya, I. L. Markov, “Fixed-outline Floorplanning Through Better Local Search”, *ICCD 2001*, pp. 328-334.

[2] C. J. Alpert, J.-H. Huang and A. B. Kahng,“Multilevel Circuit Partitioning”, *DAC 1997*, pp. 530-533.

[3] C. J. Alpert, ISPD-98 circuit benchmarks, “The ISPD98 Circuit Benchmark Suite”, http://vlsicad.cs.ucla.edu/~cheese/ispd98.html

[4] Cadence Inc, “Openbook documentation for QPlace”, QP version 5.1.67 10/27/2000.

[5] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?”, *DAC 2000*, pp. 477-482.

[6] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Optimal Partitioners and End-case Placers for Standard-cell Layout”, *IEEE Trans. on CAD, vol. 19, no. 11, 2000*, pp. 1304-1314

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

[8] K. Doll, F. M. Johannes and K. J. Antreich, “Iterative Placement Improvement By Network Flow Methods”. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.13, (no.10), Oct. 1994. pp. 1189-1200.

[9] S. Dutt, “Effective Partition-Driven Placement with Simultaneous Level Processing and a Global Net View”, *ICCAD 2000*, p. 254.

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

[11] G. Karypis, R. Agarwal, V. Kumar, and S. Shekhar, “Multilevel Hypergraph Partitioning: Applications in VLSI Design”, *DAC ’97*, pp. 526-529

[12] J. Lin and Y. Chang, “TCG: A Transitive Closure Graph Based Representation for Non-slicing Floorplans,” *DAC 2001*.

[13] Fan Mo, Abdallah Tabbara, Robert K. Brayton, “A Force-Directed Macro-Cell Placer”, *ICCAD2000*

[14] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence pair”, *IEEE Trans. on CAD*, vol 15(12), pp. 1518-1524, 1996

[15] N. Sherwani, “Algorithms for VLSI Physical Design Automation”, 3rd ed. Kluwer, 1999.

[16] X. Tang, R. Tian and and D. F. Wong, “Fast Evaluation of Sequence Pair in Block Placement by Longest Common Subsequence Computation”, *DATE 2000*, pp. 106-111.

[17] X. Tang and D. F. Wong, “FAST-SP: A Fast Algorithm for Block Placement Based on Sequence Pair”, *ASPDAC 2001*.

[18] M. Wang, X. Yang and M. Sarrafzadeh, “Dragon2000: Standard-cell Placement Tool for Large Industry Circuits”, *ICCAD 2000*.

[19] M. C. Yildiz and P. H. Madden, “Improved Cut Sequences for Partitioning Based Placement”, *DAC 2001*.