Handling Complexities in Modern Large-Scale Mixed-Size Placement*

Jackey Z. Yan Natarajan Viswanathan Chris Chu
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50010
{zijunyan, nataraj, cnchu}@iastate.edu

ABSTRACT

In this paper, we propose an effective algorithm flow to handle large-scale mixed-size placement. The basic idea is to use floorplanning to guide the placement of objects at the global level. The flow consists of four steps: 1) The objects in the original netlist are clustered into blocks; 2) Floorplanning is performed on the blocks; 3) The blocks are shifted within the chip region to further optimize the wirelength; 4) With big macro locations fixed, incremental placement is applied to place the remaining objects. There are several advantages of handling placement at the global level with a floorplanning technique. First, the problem size can be significantly reduced. Second, exact HPWL can be minimized. Third, precise object distribution can be achieved so that legalization only needs to handle minor overlaps among small objects in a block. Fourth, rotation and various placement constraints on macros can be handled. To demonstrate the effectiveness of this new flow, we implement a high-quality floorplan-guided placer called *FLOP*. We also construct the Modern Mixed-Size (MMS) placement benchmarks which can effectively represent the complexities of modern mixed-size designs and the challenges faced by modern mixed-size placers. Compared with state-of-the-art mixed-size placers and leading macro placers, experimental results show that *FLOP* achieves the best wirelength, and easily obtains legal solutions on all circuits.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuits, Design Aids**]: Placement and routing

General Terms

Algorithms, Design, Performance

Keywords

Floorplanning, Incremental Placement, Mixed-size Design

1. INTRODUCTION

*This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998.

DAC'09, July 26–31, 2009, San Francisco, California, USA.

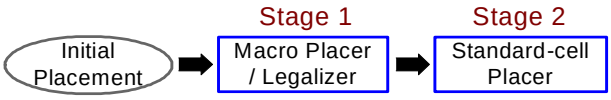

Figure 1: Previous two-stage approach.

In the nanometer scale era, placement has become an extremely challenging stage in modern VLSI designs. Millions of objects need to be placed legally within a chip region, while both the interconnection and object distribution have to be optimized simultaneously. As an early step of VLSI physical design flow, the quality of the placement solution has significant impacts on both routing and manufacturing. In modern System-on-Chip (SoC) designs, the usage of Intellectual Property (IP) and embedded memory blocks becomes more and more popular. As a result, a design usually contains tens or even hundreds of big macros. A design with big movable macros and numerous standard cells is known as mixed-size design, where the placement of big macros plays a key role. Due to the big size difference between big macros and standard cells, the placement of mixed-size designs is much more difficult than the standard-cell placement. Existing placement algorithms usually cannot generate a legal solution by themselves. They have to rely on a post-placement legalization process. However, legalizing big macros with wirelength minimization has been considered very hard to solve for a long time.

1.1 Previous Work

Most mixed-size placement algorithms place both the macros and the standard cells simultaneously. Examples are the annealing-based placer Dragon [1], the partitioning-based placer Capo [2], and the analytical placers FastPlace3 [3], APlace2 [4], Kraftwerk [5], mPL6 [6], and NTUplace3 [7]. The analytical placers are the state-of-the-art placement algorithms. They can produce the best result in the best runtime. But, the analytical approach has two problems. First, only an approximation (e.g., by log-sum-exp or quadratic function) of the Half-Perimeter Wirelength (HPWL) is minimized. Second, the distribution of objects is also approximated and that usually results in a large amount of overlaps. They have to rely on a legalization step to resolve the overlaps. For mixed-size designs, such legalization process is very difficult and is likely to significantly increase the HPWL.

Other researchers apply a two-stage approach as shown in Figure 1 to handle the mixed-size placement. An initial wirelength-driven placement is first generated. Then a macro placement or legalization algorithm is used to place only the macros, without considering the standard cells. After that, the macros are fixed, and the standard cells are re-placed in the remaining whitespace from scratch. As the macro placement is a crucial stage in this flow, people propose

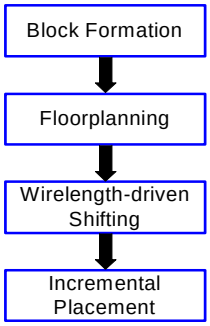

Figure 2: New algorithm flow for mixed-size placement.

different techniques to improve the quality of result (QoR). Based on the MP-tree representation, Chen et al. [8] used a packing-based algorithm to place the macros around the four corners of the chip region. In [9], a transitive closure graph (TCG) based technique was applied to enhance the quality of macro placement. One main problem with the above two approaches is that the initial placement is produced with large amount of overlaps. Thus, the initial solution may not provide good indications on the locations of objects. However, the following macro-placement stage still determines the macro locations by minimizing the displacement from the low-quality initial placement. Alternatively, Adya et al. [10] used an annealing-based floorplanner to directly minimize the HPWL among the macros and clustered standard cells at the macro-placement stage. But, they still have to rely on the illegal placement to determine the initial locations of macros and clusters. For all of the above two-stage approaches, after fixing the macros, the initial positions of standard cells have to be discarded to reduce the overlaps.

1.2 Our Contributions

To effectively handle the complexities of mixed-size placement, we present a new algorithm flow which efficiently integrates floorplanning and incremental placement algorithms. As floorplanners have a good capability of handling a small number of objects [2], we apply floorplanning on the clustered circuit to generate a global overlap-free layout, and use it to guide the subsequent placement algorithm. This new flow is as follows (see Fig. 2).

1. **Block Formation**: The purpose of the first step is to cut down the problem size. We define "small objects" as small macros and standard cells. The small objects are clustered into soft blocks, while each big macro is treated as a single hard block.

2. **Floorplanning**: In this step, a floorplanner is applied on the blocks to directly minimize the *exact* HPWL. Simultaneously, the objects are precisely distributed across the chip region to guarantee an overlap-free layout.

3. **Wirelength-driven Shifting**: In order to further optimize the HPWL, the blocks are shifted at the floorplan level. After shifting, big macros are fixed. The remaining movable objects are assumed to be at the center of the corresponding soft block.

4. **Incremental Placement**: Lastly, the placement algorithm will place the remaining objects. The initial positions of such objects provided by the previous step are used to guide the incremental placement.

Comparing this new methodology with the state-of-the-art analytical placers, we can see that it is superior in several aspects: 1) The exact HPWL is optimized in Steps 1–3; 2) The objects are more precisely distributed in Step 2; 3) Placement constraints and macro orientation optimization can be handled in Step 2. Compared with the previous two-stage approach, instead of starting from an illegal initial placement, we use the floorplanner to directly generate a global overlap-free layout among the big macros, *as well as between big macros and small objects*. In addition, the problem size has been significantly reduced by clustering. A good floorplanner should be able to produce a high-quality global layout for the subsequent incremental placer. Furthermore, the initial positions of the small objects are not discarded. We keep such information as a starting point of incremental placement. Since the big macros have already been fixed, the placer avoids the difficulty of legalizing the big macros.

Based on the new algorithm flow, we implement a robust, efficient and high-quality floorplan-guided placer called *FLOP*. It can effectively handle mixed-size placement with all movable objects including both macros and standard cells. *FLOP* can also optimize the macro orientation respecting to packing and wirelength optimization.

To show the effectiveness of *FLOP*, we derive the Modern Mixed-Size (MMS) placement benchmarks from the original ISPD05/06 Placement Benchmarks. These new circuits can represent the challenges of modern large-scale mixed-size placement.

The rest of this paper is organized as follows. Section 2 describes the overview of *FLOP*. Section 3 introduces the block formation and floorplanning algorithms. Section 4 presents the wirelength-driven shifting technique. Section 5 describes the incremental placement algorithm. Section 6 describes the MMS benchmarks. Section 7 presents the experimental results. Finally this paper ends with the conclusion and future work.

2. OVERVIEW OF FLOP

FLOP follows the same algorithm flow as shown in Figure 2.

The block formation is based on the result of recursive partitioning of the original circuit. After partitioning, small objects in each partition are clustered into a soft block and each big macro becomes a single hard block.

In the floorplanning step, *FLOP* adopts a min-cut based fixed-outline floorplanner similar to *DeFer* [11]. In *DeFer*, a hierarchy of the blocks needs to be derived using recursive partitioning. Because such a hierarchy has already been generated during the block formation step, it will be passed down and will not be generated again. Another way to look at the flow of *FLOP* is that the block formation step is merged into the floorplanning step as the first stage of *DeFer*.

We formulate the wirelength-driven shifting problem as a linear programming (LP) problem. Therefore, we can find the *optimal* block position in terms of the HPWL minimization among the blocks. In the LP-based shifting we only ignore the local netlist among small objects within each soft block.

Because analytical placers have the best capability in placing a large number of small objects, we use an analytical placer as the engine in the incremental placement step.

3. BLOCK FORMATION AND FLOORPLANNING

A high-quality and non-stochastic fixed-outline floorplanner *DeFer* was presented in [11]. It has been shown that, compared with other fixed-outline floorplanners, *DeFer* achieves the best success rate, the best wirelength and the best runtime on average.

Here is a brief description of the algorithm flow of *DeFer*: Firstly the original circuit is partitioned into several subcircuits, each of which contains at most 10 objects. After that, a high-level slicing tree structure is built up. Secondly, for each subcircuit an associated shape curve is generated to represent all possible slicing layouts within the subcircuit. Thirdly, the shape curves are combined from

bottom-up following the high-level slicing tree. In the final shape curve at the root the points within the fixed outline are chosen for further HPWL optimization. At the end *DeFer* outputs a final layout.

In *FLOP*, we use *DeFer* in the floorplanning step. To make it more robust and efficient for mixed-size placement, we propose some new techniques and strategies, which are described in Sections 3.1–3.3.

3.1 Usage of Exact Net Model

We use the exact net model in [12] to improve the HPWL in partitioning. By applying this net model in partitioning, the cut value becomes exactly the same as the placed HPWL, so that the partitioner can directly minimize the HPWL instead of interconnections between two partitions. In *FLOP* at the first β levels of the high-level slicing tree ($\beta = 3$ by default), we apply two cuts on the original partition. One is horizontal cut, and another is vertical cut. We compare these two cuts and pick the one with less cost, i.e. HPWL.

However, for a vertical/horizontal cut, the cut value returned by the net model is only equal the horizontal/vertical component of HPWL. So for two cuts with different directions, it is incorrect to decide a better cut direction based on the two cut values generated by these two cuts. The authors in [12] avoided such comparison by fixing the cut direction based on the dimension of the partition region. Nevertheless, this may potentially lose the better cut direction. Here we propose a simple heuristic to solve the cut value comparison between the cuts from two different directions.

Suppose K is the total number of nets in one partition that we are going to cut. For the horizontal cut (H-cut), $L^x_{H_i}/L^y_{H_i}$ is the horizontal/vertical component of the HPWL of net i, the same as $L^x_{V_i}$ and $L^y_{V_i}$ for the vertical cut (V-cut). So the total HPWL of the K nets in this partition are:

$$\text{For H-cut}: \quad L_H = \sum_{i=1}^{K} L^x_{H_i} + \sum_{i=1}^{K} L^y_{H_i}$$
$$\text{For V-cut}: \quad L_V = \sum_{i=1}^{K} L^x_{V_i} + \sum_{i=1}^{K} L^y_{V_i}$$

Thus, the correct way to make the comparison between H-cut and V-cut should be:

$$\text{if } L_H \geq L_V \Rightarrow \text{V-cut is better}$$
$$\text{if } L_H < L_V \Rightarrow \text{H-cut is better}$$

As the net model only returns $\sum_{i=1}^{K} L^y_{H_i}$ for H-cut, and $\sum_{i=1}^{K} L^x_{V_i}$ for V-cut, we need find a way to estimate $\sum_{i=1}^{K} L^x_{H_i}$ and $\sum_{i=1}^{K} L^y_{V_i}$. Let the aspect ratio (i.e. height/width) of the partition region be γ. When K is very big, based on statistics we can have:

$$\frac{\sum_{i=1}^{K} L^y_{H_i}}{\sum_{i=1}^{K} L^x_{H_i}} \approx \frac{\sum_{i=1}^{K} L^y_{V_i}}{\sum_{i=1}^{K} L^x_{V_i}} \approx \gamma$$

Thus,

$$\text{if } L^y_H \geq L^x_V \cdot \gamma \Rightarrow \text{V-cut is better}$$
$$\text{if } L^y_H < L^x_V \cdot \gamma \Rightarrow \text{H-cut is better}$$

Two reasons prevent us from applying the net model in lower levels ($> \beta$): 1) As partitioning goes on, K becomes smaller and smaller, which makes the approximation of $\sum_{i=1}^{K} L^x_{H_i}$ and $\sum_{i=1}^{K} L^y_{V_i}$ inaccurate; 2) Using the net model, we restrict the combine direction in the Generalized Slicing Tree [11], which hurts the packing quality. To make a trade-off we only apply the net model in the first β levels.

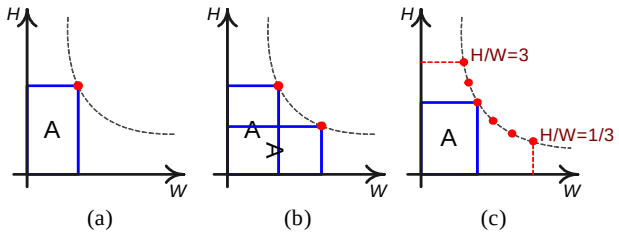

Figure 3: Generation of shape curves for blocks.

3.2 Block Formation

As mentioned earlier, since *DeFer* starts with a min-cut partitioning, *FLOP* merges the block formation step into the floorplanning step. After the original circuit is partitioned into multiple subcircuits, in each subcircuit we treat a big macro as a hard block, and cluster all small objects into a soft block.

However, in *DeFer* the partitioning will not stop until each subcircuit contains less than or equal to 10 objects. If the same stopping criteria is used in *FLOP*, then most subcircuits will contain at most 10 standard cells, which means by clustering we can only cut down the problem size by at most 90%. Nevertheless, for a typical placement problem with millions of objects, the resulted circuit size is still too big for the floorplanning algorithm. So here we propose a more suitable stopping criteria. Let A_o be the total area of all objects in the design. In one partition there are N_p objects of which the total area is A_p, α is the area bound ($\alpha = 0.15\%$ by default). We will stop cutting this partition, if *either* one of the following conditions is satisfied: 1) $\frac{A_p}{A_o} \leq \alpha$; 2) $N_p \leq 10$.

3.3 Generation of Shape Curve for Blocks

To capture the shape of the blocks, we generate an associated shape curve for each block. For the hard block if a macro cannot be rotated, only one point representing the user-specified rotation is generated (see Fig. 3 (a)). Otherwise two points representing two different rotations are generated (see Fig. 3 (b)). For the soft block we bound its aspect ratio from $1/3$ to 3, and sample multiple points on the shape curve to represent its shape (see Fig. 3 (c)). Considering the target density constraint in the placement, we add some white space in each soft block. In some sense, we "inflate" the soft block based on the target density.

$$A'_{s_i} = \frac{A_{s_i}}{TD} \times (max((TD - 0.93), 0) \times 0.5 + 1) \qquad (1)$$

In Equation 1, for soft block i, A'_{s_i} is the "inflated area", A_{s_i} is the total area of objects within soft block i, and TD is the target density. Based on this formula, if the target density is more than 93%, we add some white space into the soft block. The purpose is to leave some space for the analytical placer to place the small objects.

4. WIRELENGTH-DRIVEN SHIFTING

In *FLOP* the wirelength-driven shifting process is formulated as a linear programming (LP) problem, which is the same as in [13]. We use the contour structure [14] to derive the horizontal and vertical non-overlapping constraints among the blocks.

The LP-based shifting is an essential part in *FLOP*. In terms of the HPWL minimization it can find the optimal position for each block, and basically provides a globally optimized layout for the analytical placer. Since the LP-based shifting optimizes the HPWL at the floorplan level, it only ignores the local nets among the small objects within each soft block. The smaller the soft block is, the less nets it ignores, and the better the HPWL we will get at last. However, if the

soft blocks become too small, numerous nets will be considered in the shifting. This would slow down the whole algorithm. Because of this, in the partition stopping criteria we set an area bound α, so that the soft blocks would not become too small. On the other hand, we only need the shifting step to generate a globally good layout. Regarding the local nets within the soft blocks, the following analytical placer can handle them very efficiently and effectively.

5. INCREMENTAL PLACEMENT

As mentioned before, the output of the wirelength-driven shifting step is a layout with legal, non-overlapping locations for the big macros. These big macros are then fixed in place to prevent further movement during any subsequent steps. But, there are multiple "soft blocks" in the layout, each containing numerous "small objects" (i.e., small macros and standard cells). The shifting step assigns these small objects to the center of the corresponding soft block. In this respect, the placement step has two key tasks: 1) Spread the small objects over the placement region and obtain a final overlap free placement among all objects; 2) Use the initial locations of the small objects as obtained by the shifting step.

To satisfy these two tasks, we use an efficient analytical incremental placement algorithm (see Algorithm 1).

Algorithm 1 Analytical Incremental Placement

```
Phase 0: Physical and Netlist based clustering
    initial_objects ← number_of_small_objects
    set locations of small objects to center of their soft blocks
    while number_of_clusters > target_number_of_clusters do
        cluster netlist using Best-choice clustering [15]
        use physical locations of small objects in clustering score
        set cluster_location ← center of gravity of the objects within cluster
    end while
end
Phase 1: Coarse global placement
    generate "fixing forces" for clusters based on their initial locations
    solve initial quadratic program (QP)
    repeat
        perform Cell Shifting [3] on coarse-grain clusters
        add spreading forces to QP formulation
        solve the quadratic program
    until placement is roughly even
    repeat
        perform Iterative Local Refinement [3] on coarse-grain clusters
    until placement is quite even
    uncluster movable macro-blocks
    legalize and fix movable macro-blocks
end
Phase 2: Refinement of fine-grain clusters
    while number_of_clusters < 0.5*number_of_small_objects do
        uncluster netlist
    end while
    perform Iterative Local Refinement on fine-grain clusters
end
Phase 3: Refinement of flat netlist
    while number_of_clusters < number_of_small_objects do
        uncluster netlist
    end while
    perform Iterative Local Refinement on flat netlist
end
Phase 4: Legalization and detailed placement
    Legalize the standard cells in the presence of fixed macros
    Perform detailed placement [16] to further improve wirelength
end
```

6. MMS BENCHMARKS

The only publicly available benchmarks for mixed-size designs are ISPD02 and ICCAD04 IBM-MS [10, 17] that are derived from ISPD98 Placement Benchmarks. As pointed out in [18], these circuits can no longer be representative of modern VLSI physical design. To continue driving the progress of physical design for the academic community, two suites of placement benchmarks [18, 19] have been released recently. They are directly derived from modern industrial ASICs design. Unfortunately, however, in the original circuits most macros have been fixed due to the difficulty of handling movable macros for the existing placers. The authors in [8, 9] freed all fixed objects in ISPD06 benchmarks and created new mixed-size placement circuits. But seven out of eight circuits *do not* have any fixed I/O objects, which is not realistic in the real designs. In order to recover the complexities of modern mixed-size designs, we modify the original ISPD05/06 benchmarks and derive the Modern Mixed-Size (MMS) placement benchmarks (see Table 1). Essentially, we make the following changes on the original circuits.

I. Macros are freed from the original positions. In the GSRC Bookshelf format that the original benchmarks use, both fixed macros and fixed I/O objects are treated as fixed objects. There is no extra specification to differentiate them. So we have to distinguish them only based on the size differences. Basically, if the area of one fixed object is more than $\lambda\times$ the average area of the whole circuit, we will recognize it as a macro. Otherwise, it is a fixed I/O object. Because for each circuit the average area is different, we need to use a different λ (see the last column in Table 1) to decide a reasonable number and suitable threshold size for the macros. There is one exception: in both circuits *bigblue2* and *bigblue4*, there is one macro that does not connect with any other objects. If this macro is freed, it may cause some trouble for quadratic-based analytical placers. So we keep it fixed. Since this macro is also very small compared with other macros, it would not affect the circuit property.

II. The sizes of all I/O objects are set to zero. In MMS benchmarks there are two types of I/Os: *perimeter I/Os* around the chip boundary and *area-array I/Os* spreading across the chip region. Generally, the area-array I/Os are allowed to be overlapped with other movable objects in the design. But existing placers treat all fixed I/Os as fixed objects, so that their algorithms *internally* do not allow such overlaps during the legalization. Since the macros have already been freed in MMS benchmarks, the placers should ignore the overlaps between fixed I/O objects and movable objects, and concentrate on the legalization of movable objects. As we cannot change the code of other placers, one simple way to enforce this is to set the sizes of all I/O objects to zero.

The target density constraints are the same as the original circuits. The same scoring function [1] is used to calculate the scaled HPWL. However, since the macros are movable in the MMS circuits, we need to modify the script used in [19] to get the correct "scaled_overflow_factor". The modification being: Any movable macro that has a width or height greater than the bin dimension used for scaled overflow calculation, is now treated as a fixed macro during scaled overflow calculation. Note that, this was the method employed by the original script on *newblue1*, which is the only design that has big movable macros in the original circuits. It is required to treat big movable macros as fixed, otherwise we will get an incorrect picture of the placement density.

We have discussed the MMS benchmarks setup with the authors in [18, 19]. To keep the original circuit properties as much as possible, the above changes are the best we can do without accessing the original industrial data of the circuits. The MMS benchmarks are publicly available at [20].

7. EXPERIMENTAL RESULTS

All experiments were performed on a Linux machine with AMD Opteron 2.6 GHz CPU and 8GB memory. We use *hMetis2.0* [21] as the partitioner and *QSopt* [22] as the LP solver. The seed of *hMetis2.0* is set to 5. Essentially, we set up four experiments.

[1] scaled_HPWL = HPWL * (1 + scaled_overflow_factor)

Circuit	#Objects	#Movable Objects	#Standard Cells	#Macros	#Fixed I/O Objects	#Net	#Net Pins	Target Density%	λ
adaptec1	211447	210967	210904	63	480	221142	944053	100	70
adaptec2	255023	254584	254457	127	439	266009	1069482	100	160
adaptec3	451650	450985	450927	58	665	466758	1875039	100	650
adaptec4	496054	494785	494716	69	1260	515951	1912420	100	460
bigblue1	278164	277636	277604	32	528	284479	1144691	100	120
bigblue2	557866	535741	534782	959	22125	577235	2122282	100	30
bigblue3	1096812	1095583	1093034	2549	1229	1123170	3833218	100	470
bigblue4	2177353	2169382	2169183	199	7970	2229886	8900078	100	550
adaptec5	843128	842558	842482	76	570	867798	3493147	50	440
newblue1	330474	330137	330073	64	337	338901	1244342	80	2000
newblue2	441516	440264	436516	3748	1252	465219	1773855	90	190
newblue3	494011	482884	482833	51	11127	552199	1929892	80	170
newblue4	646139	642798	642717	81	3341	637051	2499178	50	400
newblue5	1233058	1228268	1228177	91	4790	1284251	4957843	50	570
newblue6	1255039	1248224	1248150	74	6815	1288443	5307594	80	650
newblue7	2507954	2481533	2481372	161	26421	2636820	10104920	80	650

Table 1: Statistics of the Modern Mixed-Size placement benchmarks.

I. To test the capability of handling the large-scale mixed-size placement, we compare *FLOP* with five state-of-the-art mixed-size placers *APlace2*, *NTUplace3*, *mPL6*, *Capo10.5* and *Kraftwerk* on MMS benchmarks. *Before the experiments, we have contacted the authors of each placer above, and they provided us their best-available binary for MMS circuits.* In Table 2, for the ISPD06 circuits (*adaptec5* – *newblue7*) the reported HPWL is the scaled HPWL. *FLOP* is the default mode of *FLOP* with all macros rotatable, and *FLOP-NR* restricts the rotation on all macros. *APlace2* crashed on every circuit, so we do not report its results. For the default mode, *FLOP* generates 8%, 2%, 44% and 26% better HPWL compared with *NTUplace3*, *mPL6*, *Capo10.5* and *Kraftwerk*, respectively. About the runtime, *FLOP* is $7\times$ and $3\times$ faster than *Capo10.5* and *mPL6*. Also *FLOP* achieves legal solution on all circuits. Compared with *FLOP-NR*, *FLOP* generates 4% better HPWL by rotating the macros.

II. To show the importance of the initial positions of small objects in the incremental placement step, we generate the results of *FLOP-NI* that *discards* such information and places all small objects from scratch. As shown in Table 2, *FLOP-NI* produces 5% worse HPWL and 17% slower than *FLOP*.

III. We compare *FLOP* with leading macro placers *CG*, *MPT* and *XDP*. Due to the *IP* issues, their binaries are not available. But the authors sent us the benchmarks used in [9]. So in Table 3 the other placers' results are cited from [9]. These benchmarks allow the rotation of macros and do not consider the target density. As shown in Table 3, *FLOP* achieves 1%, 12%, 7% and 14% better HPWL compared with *CG*, *MPT*, *XDP* and *NTUplace3*, respectively. To show which algorithm provides the best macro location, we use *NTUplace3* to substitute the incremental placer inside *FLOP* (*NTUplace3* does not support incremental placement). The results show that *FLOP+NTUplace3* generates 9% worse HPWL than *CG*. But this does not mean *FLOP* is weaker than *CG* in terms of handling the macros. We observe that *FLOP+NTUplace3* produces significantly worse HPWL on *newblue7*. However, using the *same* macro locations generated by *FLOP*, the incremental placer inside *FLOP* achieves the *best* HPWL on *newblue7*. We believe this is because *NTUplace3* is not an incremental placer. As shown earlier, non-incremental placement will significantly degrade *FLOP*'s result.

IV. The runtime breakdown of *FLOP* is shown in Figure 4. We can see that the LP-based shifting takes almost $1/3$ of the total runtime. This is the main bottle neck of the runtime in *FLOP*.

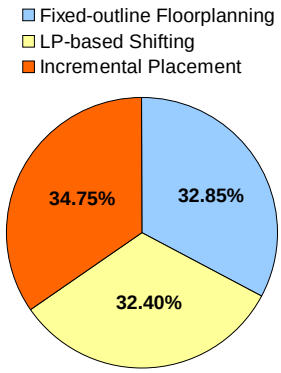

Figure 4: Runtime breakdown of *FLOP*.

8. CONCLUSION

This paper presents a new algorithm flow for large-scale mixed-size placement. To show the effectiveness of such flow, a high-quality mixed-size placer *FLOP* is proposed. Compared with state-of-the-art mixed-size placers and leading macro placers, *FLOP* achieves the best HPWL, and easily produces the legal layout for every circuit.

We believe there is much room to further improve the QoR of *FLOP*. For example, we can use the min-cost flow algorithm to substitute the linear programming formulation in order to speed up the LP-based shifting step. We also observe that the partitioning takes around 80% of the total runtime in the floorplanning step. Thus a stand-alone clustering algorithm is needed in the block formation step to cut down the problem size before partitioning. This will definitely improve both the runtime and HPWL. In the future, different floorplanners and placers can be incorporated into this flow to handle other problems, e.g. placement with geometry constraints.

Acknowledgment

The authors would like to thank Dr. George Karypis, Dr. Gi-Joon Nam, Guojie Luo, Tung-Chieh Chen and Peter Spindler for the help with hMetis2.0, ISPD05/06 Placement Benchmarks, mPL6, NTUplace3 and Kraftwerk, respectively. Also thanks goes to Yi-Lin Chuang and Dr. Yao-Wen Chang for providing us their benchmarks.

9. REFERENCES

[1] T. Taghavi, X. Yang, B.-K. Choi, M. Yang, and M. Sarrafzadeh. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proc. ISPD*, pages 209–211, 2006.

[2] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut floorplacement. *IEEE Trans. on Computer-Aided Design*, 25(7):1313–1326, July 2006.

Circuit	NTUplace3 [7]		mPL6 [6]		Capo10.5 [2]		Kraftwerk [5]		FLOP-NR		FLOP-NI		FLOP	
	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)
adaptec1	80.45	630	77.84	2404	84.77	5567	86.73	285	77.18	722	85.27	931	76.83	824
adaptec2	136.46	1960	88.40	2870	92.61	7373	108.61	408	87.17	1219	86.72	1371	84.14	1192
adaptec3	*illegal*	–	180.64	5983	202.37	16973	272.76	773	182.21	1943	173.07	2352	175.99	2116
adaptec4	177.23	1896	162.02	5971	202.38	17469	260.96	962	166.55	2398	175.67	2689	161.68	2427
bigblue1	95.11	955	99.36	2829	112.58	8458	130.78	348	95.45	1776	98.91	2174	94.92	2007
bigblue2	144.42	1661	144.37	12202	149.54	17647	*aborted*	–	150.66	2373	162.40	3622	153.02	3110
bigblue3	*illegal*	–	319.63	9548	583.37	69921	417.73	2369	372.79	7037	394.75	6531	346.24	5437
bigblue4	764.72	8182	804.00	23868	915.73	109785	969.03	6000	807.53	13816	839.53	21927	777.84	19671
adaptec5*	*aborted*	–	376.30	22636	565.88	23957	402.21	1653	381.83	5048	385.07	3882	357.83	3293
newblue1*	60.73	875	66.93	3171	110.54	3133	76.22	481	73.36	1368	71.69	1280	67.97	1012
newblue2*	*aborted*	–	179.18	6044	303.25	8156	272.67	615	231.94	2646	190.50	2830	187.40	2414
newblue3*	*aborted*	–	415.86	17623	1282.19	73339	374.54	578	344.71	2336	355.07	3237	345.99	2757
newblue4*	*aborted*	–	277.69	9732	300.69	6589	291.45	1266	256.91	2575	268.46	3282	256.54	2455
newblue5*	*aborted*	–	515.49	24806	570.32	16548	503.13	2639	516.71	8801	536.38	10546	510.83	9163
newblue6*	*aborted*	–	482.44	13112	609.16	18076	651.34	2726	502.24	9450	506.99	10740	493.64	9563
newblue7*	*aborted*	–	1038.66	31680	1481.45	43386	*illegal*	–	1113.07	18765	1101.07	27029	1078.18	25104
Norm	**1.08**	**0.78**	**1.02**	**2.95**	**1.44**	**6.56**	**1.26**	**0.35**	**1.04**	**1.00**	**1.05**	**1.17**	**1**	**1**

Table 2: Comparison with mix-size placers on MMS Benchmarks (* comparison of scaled HPWL), HPWL($\times 10e6$).

Circuit	CG [9]+NTUplace3	MPT [8]+NTUplace3	XDP [23]+NTUplace3	NTUplace3	FLOP+NTUplace3	FLOP
	HPWL	HPWL	HPWL	HPWL	HPWL	HPWL
adaptec5	29.46	31.01	31.08	29.03	27.94	27.36
newblue1	6.23	6.50	6.32	6.06	7.41	7.32
newblue2	18.89	22.60	18.90	28.09	25.38	23.79
newblue3	30.18	37.57	37.64	53.48	31.20	33.61
newblue4	21.38	23.77	22.01	22.83	21.03	19.72
newblue5	42.92	43.71	45.41	39.91	37.21	36.66
newblue6	44.93	50.50	46.43	44.24	45.80	41.87
newblue7	99.03	108.06	102.21	100.06	170.00	86.96
Norm	**1.01**	**1.12**	**1.07**	**1.14**	**1.10**	**1**

Table 3: Comparison with macro placers on modified ISPD06 benchmarks [9] with default chip utilization, HPWL($\times 10e7$).

[3] N. Viswanathan, M. Pan, and C. Chu. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. ASP-DAC*, pages 135–140, 2007.

[4] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. ISPD*, pages 218–220, 2006.

[5] P. Spindler and F. M. Johannes. Fast and robust quadratic placement combined with an exact linear net model. In *Proc. ICCAD*, pages 179–186, 2006.

[6] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-sized placement. In *Proc. ISPD*, pages 212–214, 2006.

[7] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *Proc. ICCAD*, pages 187–192, 2006.

[8] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and D. Liu. MP-tree: A packing-based macro placement algorithm for mixed-size designs. In *Proc. DAC*, pages 447–452, 2007.

[9] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang. Constraint graph-based macro placement for modern mixed-size circuit designs. In *Proc. ICCAD*, pages 218–223, 2008.

[10] S. Adya and I. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proc. ISPD*, pages 12–17, 2002.

[11] J. Z. Yan and C. Chu. DeFer: Deferred decision making enabled fixed-outline floorplanner. In *Proc. DAC*, pages 161–166, 2008.

[12] T.-C. Chen, Y.-W. Chang, and S.-C. Lin. IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In *Proc. ICCAD*, pages 159–164, 2005.

[13] X. Tang, R. Tian, and M. D. F. Wong. Minimizing wire length in floorplanning. *IEEE Trans. on Computer-Aided Design*, 25(9):1744–1753, September 2006.

[14] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In *Proc. DAC*, pages 268–273, 1999.

[15] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *IEEE Trans. on Computer-Aided Design*, 25(4):678–691, April 2006.

[16] M. Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proc. ICCAD*, pages 48–55, 2005.

[17] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov. Unification of partitioning, placement and floorplanning. In *Proc. ICCAD*, pages 550–557, 2004.

[18] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz. The ISPD2005 placement contest and benchmarks suite. In *Proc. ISPD*, pages 216–220, 2005.

[19] G.-J. Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proc. ISPD*, pages 167–167, 2006.

[20] MMS Placement Benchmarks. `http://www.public.iastate.edu/~zijunyan/`.

[21] hMetis2.0. `http://glaros.dtc.umn.edu/gkhome/`.

[22] QSopt LP Solver. `http://www.isye.gatech.edu/~wcook/qsopt/`.

[23] J. Cong and M. Xie. A robust detailed placement for mixed-size ic designs. In *Proc. ASP-DAC*, pages 188–194, 2006.