SimPL: An Effective Placement Algorithm

Myung-Chul Kim, Dong-Jin Lee and Igor L. Markov
University of Michigan, Department of EECS, 2260 Hayward St., Ann Arbor, MI 48109-2121
{mckima, ejdjsy, imarkov}@eecs.umich.edu

Abstract—**We propose a self-contained, flat, force-directed algorithm for global placement that is simpler than existing placers and easier to integrate into timing-closure flows. It maintains lower-bound and upper-bound placements that converge to a final solution. The upper-bound placement is produced by a novel *rough legalization* algorithm. Our placer SimPL outperforms mPL6, NTUPlace3, FastPlace3, APlace2 and Capo simultaneously in runtime and solution quality, running 6.4 times faster than mPL6 and reducing wirelength by 2% on the ISPD 2005 benchmark suite.**

I. Introduction

Global placement currently remains at the core of physical design and is a gating factor for downstream optimizations during timing closure [2]. Despite impressive improvements reported by researchers [15] and industry software in the last five years, state-of-the-art algorithms and tools for placement suffer several key shortcomings which are becoming more pronounced at recent technology nodes. These shortcomings fall into four categories: (i) speed, (ii) solution quality, (iii) simplicity and integration with other optimizations, (iv) support for multithreaded execution. We propose the SimPL algorithm that simultaneously improves results in the first three categories and lends itself naturally to parallelism on multicore CPUs.

State-of-the-art algorithms for global placement form two families: (i) *force-directed* placers, such as Kraftwerk2 [20], FastPlace3 [22] and RQL [23], and (ii) *non-linear optimization* techniques, such as APlace2 [12], NTUPlace3 [7] and mPL6 [6]. Force-directed algorithms model total net length by a quadratic function of cell locations and minimize it by solving a large sparse system of linear equations. To discourage cell overlap, *forces* are added pulling cells away from high-density areas. These forces are modeled by *pseudopins* and *pseudonets*, which extend the original quadratic function [11]. They are updated after each linear-system solve until iterations converge. Non-linear optimization models net length by more sophisticated differentiable functions with linear asymptotic behavior which are then minimized by advanced numerical analysis techniques [12]. Cell density is modeled by functional terms, which are more accurate than forces, but also require updates after each change to placement [7], [12]. Algorithms in both categories are directly used in the industry or closely resemble those in industry placers.

Tools based on non-linear optimization achieve the best results reported for academic implementations [7] and EDA vendor tools, but are significantly slower, which is problematic for modern flat SoC placement instances with tens of millions of movable objects. To scale the basic non-linear optimization framework, all tools in this family employ *netlist clustering* and *multilevel extensions*, sometimes at the cost of solution quality. Such multilevel placers perform many sequential steps, obstructing efficient parallelization. Moreover, clustering and refinement do not fully benefit from modern multicore CPUs. Due to their complexity, multilevel placers are also harder to maintain, improve, and combine with other physical-design techniques. In particular, clustered netlists complicate accurate static timing analysis, congestion maps and physical synthesis, such as performance-driven buffering, gate sizing, fanin/fanout optimization, cloning, etc [2]. Hence, timing-closure flows often repeat global placement 3-4 times, alternating it with timing analysis, physical synthesis and congestion improvement.

State-of-the-art force-directed placers tend to run many times faster than non-linear optimization, but also use multilevel extensions in their most competitive configurations. Their solution quality is mixed. FastPlace3 underperforms mPL6 and NTUPlace3 [7], but the industry tool RQL closely related to FastPlace slightly outperforms these two non-linear placers. Kraftwerk2 is the only competitive *flat* placer (i.e., it does not use clustering) and rivals other force-directed placers in speed. However, it lags behind in solution quality. Its implementation poses several challenges, such as quickly solving Poisson's equation, ensuring the convergence of iterations and avoiding halos over macros. Our experience indicates that the performance of Kraftwerk2 can be uneven, and stability can only be achieved with some loss of solution quality [13]. State-of-the-art placers are described in the book [15] and recent journal papers [3], [7], [20].

In this work, we develop a new, self-contained technique for global placement that ranks as a flat force-directed placement algorithm. It maintains lower-bound and upper-bound placements that converge to a final solution. The upper-bound placement is produced by a novel *rough legalization* algorithm based on geometric top-down partitioning and non-linear scaling. Our implementation outperforms published placers simultaneously in solution quality and speed on standard benchmarks. Our algorithm is simpler, and our attempts to improve overall results using additional modules and extensions from existing placers (such as *netlist clustering* [6], [12], [22], *iterative local refinement* (ILR) [22], and median-improvement (*BoxPlace*) [13]) were unsuccessful.

In the remainder of this paper, Section II describes the building blocks from which our algorithm was assembled. Section III introduces our key ideas and articulates our solution of the *force modulation* problem. The SimPL algorithm is presented in Section IV. Extensions and the use of parallelism are discussed in Section V. Empirical validation is described in Section VI, and Section VII summarizes our results.

II. Essential Concepts and Building Blocks

Given a netlist $\mathcal{N} = (E, V)$ with nets E and nodes (cells) V, *global placement* seeks node locations (x_i, y_i) such that the area of nodes within any rectangular region does not exceed the area of (cell sites in) that region.[1] Some locations of cells may be given initially and fixed. The interconnect objective optimized by global placement is the Half-Perimeter WireLength (HPWL). For node locations $\vec{\mathrm{x}} = \{x_i\}$ and $\vec{\mathrm{y}} = \{y_i\}$, HPWL$_{\mathcal{N}}(\vec{\mathrm{x}}, \vec{\mathrm{y}})$= HPWL$_{\mathcal{N}}(\vec{\mathrm{x}})$+HPWL$_{\mathcal{N}}(\vec{\mathrm{y}})$, where

$$HPWL_{\mathcal{N}}(\vec{\mathrm{x}}) = \Sigma_{e \in E}[\max_{i \in e} x_i - \min_{i \in e} x_i] \quad (1)$$

Efficient optimization algorithms often approximate HPWL$_{\mathcal{N}}$ by differentiable functions, as illustrated next.

Quadratic optimization. Consider a graph $\mathcal{G} = (E_{\mathcal{G}}, V)$ with edges $E_{\mathcal{G}}$, vertices V and edge weights $w_{ij} > 0$ for all edges $e_{ij} \in E_{\mathcal{G}}$. The *quadratic objective* $\Phi_{\mathcal{G}}$ is defined as

$$\Phi_{\mathcal{G}}(\vec{\mathrm{x}}, \vec{\mathrm{y}}) = \Sigma_{i,j} w_{i,j}[(x_i - x_j)^2 + (y_i - y_j)^2] \quad (2)$$

Its x & y components are cast in matrix form [3], [20]

$$\Phi_{\mathcal{G}}(\vec{\mathrm{x}}) = \frac{1}{2}\vec{\mathrm{x}}^T Q_x \vec{\mathrm{x}} + \vec{\mathrm{c}}_x^T \vec{\mathrm{x}} + \mathrm{const} \quad (3)$$

The Hessian matrix Q_x captures connections between pairs of movable vertices, while vector $\vec{\mathrm{c}}_x$ captures connections between movable and fixed vertices. When Q_x is non-degenerate, $\Phi_{\mathcal{G}}(\vec{\mathrm{x}})$ is a strictly convex function with a unique minimum, which can be found by solving the system of linear equations $Q_x \vec{\mathrm{x}} = -\vec{\mathrm{c}}_x$. Solutions can be quickly approximated by iterative Krylov-subspace techniques, such as the Conjugate Gradient (CG) method and its variants [19]. Since Q_x is symmetric positive definite, CG iterations provably minimize the residual norm. The convergence is monotonic [21], but its rate depends on the spectral properties of Q_x, which can be enhanced by *preconditioning*. In other words, we solve the equivalent system $P^{-1}Q_x = -P^{-1}\vec{\mathrm{c}}_x$ for a nondegenerate matrix P, such that P^{-1} is an easy-to-compute approximation of Q_x^{-1}. Given that Q_x is diagonally dominant, we chose P to be its diagonal, also known as the *Jacobi preconditioner*. Our placement algorithm (Section IV-C) deliberately enhances diagonal dominance in Q_x.

The Bound2Bound net model [20]. To represent the HPWL objective by the quadratic objective, the netlist $\mathcal{N}$ is transformed in *two* graphs, $\mathcal{G}_x$ and $\mathcal{G}_y$, that preserve the node set V and represent each two-pin net by a single edge with weight $1/length$. Larger nets are decomposed depending on the relative placement of vertices — for each p-pin net, the *extreme* nodes (min and max) are connected to each other and to each *internal* node by edges, with the following weight

$$w_{x,ij}^{B2B} = \frac{1}{(p-1)|x_i - x_j|} \quad (4)$$

For example, 3-pin nets are decomposed into cliques with edge weight $1/2l$, where l is the length of a given edge. In general, this quadratic objective and the Bound2Bound (B2B) net decomposition capture the HPWL objective exactly, but only for the given placement. As locations change, the error may grow, necessitating multiple updates throughout the placement algorithm.

Most quadratic placers use the placement-independent star or clique decompositions, so as not to rebuild Q_x and Q_y many times [3], [22], [23]. Yet, the B2B model uses fewer edges than cliques ($p > 3$), avoids new variables used in stars, and is more accurate than both stars and cliques [20].

III. Key Ideas in Our Work

Analytic placement techniques first minimize a function of interconnect length, neglecting overlaps between standard cells, macros, etc. This initial step places many cells in densely populated regions, typically around the center of the layout. Cell locations are then gradually spread through a series of placement iterations, during which interconnect length slowly *increases*, converging to a final overlap-free placement (a small amount of overlap is often allowed and later resolved during detailed placement).

Our algorithm also starts with pure interconnect minimization, but its next step is unusual — most overlaps are removed using a fast *rough legalizer* based on top-down geometric partitioning and nonlinear scaling. Locations of movable objects in the legalized placement serve as *anchors* to coerce the initial locations into a configuration with less overlap, by adding pseudonets to baseline force-directed placement [11]. Each subsequent iteration of our algorithm produces (i) an illegal placement that *underestimates* the final result — through linear system solver, and (ii) an almost-legal placement that *overestimates* the final

[1]In practice, this constraint is enforced for bins of a regular grid.

result — through rough legalization. The gap between the lower and upper bounds helps monitor convergence (Section IV-C).

Solving the force-modulation problem. A key innovation in SimPL is the interaction between the lower-bound and the upper-bound placements — it ensures convergence to a no-overlap solution while optimizing interconnect length. It solves two well-known challenges in analytic placement: (1) finding directions in which to spread the locations (*force orientation*), and (2) determining the appropriate amount of spreading (*force modulation*) [13], [23]. This is unlike previous work, where spreading directions are typically based on *local information*, e.g., placers based on non-linear optimization use *gradient* information and require a large number of expensive iterations. Kraftwerk2 [20] orients spreading forces according to solutions of Poisson's equation, providing a global perspective and speeding up convergence. However, this approach does not solve the force-modulation problem, as articulated in [13].[2] The authors of RQL [23], which can be viewed as an improvement on FastPlace, revisit the force-modulation problem and address it by a somewhat *ad hoc* limit on the magnitude of spreading forces. In our work, the *rough legalization algorithm* (Section IV-B), invoked at each iteration, determines both the direction and the magnitude of spreading forces. It is global in nature, accounts for fixed obstacles, and preserves relative placement to ensure interconnect optimization and convergence. Our placement algorithm does not require exotic components, such as a Poisson-equation solver used by Kraftwerk; our C++ implementation is self-contained.

Global placement with look-ahead. The legalized upper-bound placements constructed at every iteration of our placer can be viewed as *look-ahead*. They pull cell locations in lower-bound placements not just away from dense regions, but also toward the regions where space is available. Such *area look-ahead* is particularly useful around fixed obstacles, where local information does not offer sufficient guidance. While not explored in this paper, similar *congestion look-ahead* and *timing look-ahead* based on legalized placements can be used to integrate our placement algorithm into modern timing-closure flows.

IV. Our Global Placement Algorithm

Our placement technique consists of three phases: initial placement, global placement iterations and post-global placement (Figure 1). Initial placement, described next, is mostly an exercise in judicious application of known components. Our main innovation is in the global placement phase. Post-global placement is straightforward, given current state of the art.

[2]The work in [13] performs force modulation with *line search* but is not currently competitive with state of the art.

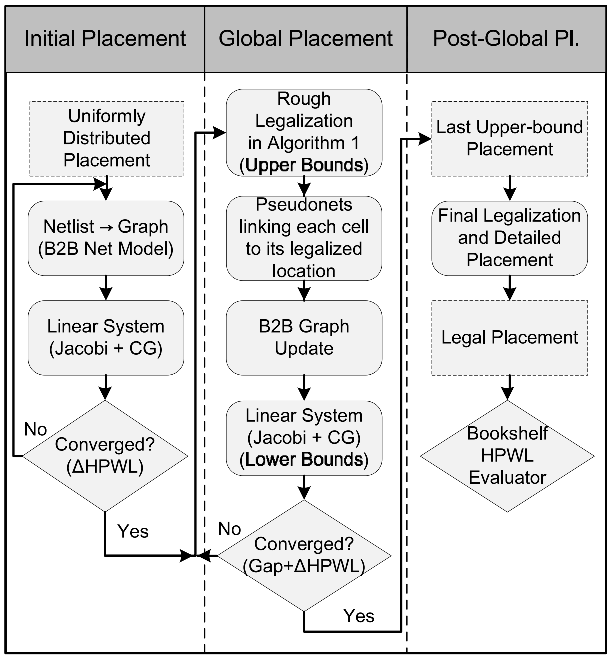

Initial Placement
Global Placement
Post-Global Pl.
Uniformly Distributed Placement
Netlist → Graph (B2B Net Model)
Linear System (Jacobi + CG)
Converged? (ΔHPWL)
No
Yes
Rough Legalization in Algorithm 1 (Upper Bounds)
Pseudonets linking each cell to its legalized location
B2B Graph Update
Linear System (Jacobi + CG) (Lower Bounds)
Converged? (Gap+ΔHPWL)
No
Yes
Last Upper-bound Placement
Final Legalization and Detailed Placement
Legal Placement
Bookshelf HPWL Evaluator

Fig. 1. The SimPL algorithm uses placement-dependent B2B net model, which is updated on every iteration. *Gap* refers to the difference between upper and lower bounds.

A. Initial Placement

Our initial-placement step is conceptually similar to those of other force-directed placers [20], [22], [23] — it entirely ignores cell areas and overlaps, so as to minimize a quadratic approximation of total interconnect length. We found that this step notably impacts the final result. Therefore, unlike FastPlace3 [22] and RQL [23], we use the more accurate BoundingBox net model from [20] reviewed in Section II. After the first quadratic solve, we rebuild the circuit graph because the B2B net model is placement-dependent. We then alternate quadratic solves and graph rebuilding until HPWL stops improving. In practice, this requires a small number of iterations (5-7), regardless of benchmark size, because the relative ordering of locations stabilizes quickly.

B. Rough Legalization

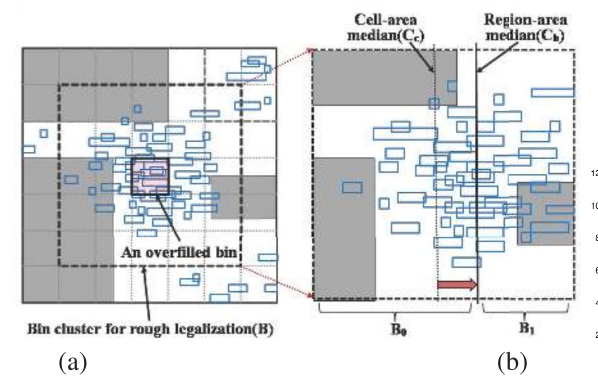
Consider a set of cell locations with a significant amount of overlap as measured using bins of a regular grid. Rough legalization changes the global positioning of those locations, seeking to remove most of the overlap (with respect to the grid) while preserving the relative ordering. This task can be formulated at different geometric scales by varying the grid. The quality of rough legalization is measured by its impact on the entire placement flow. Our rough legalization is based on top-down recursive geometric partitioning and non-linear scaling, as outlined in Algorithm 1.

Algorithm 1 Rough Legalization by Top-down Geometric Partitioning and Non-linear Scaling

```
Maximum allowed density $\gamma$, where $0 < \gamma < 1$
Floorplan with obstacles
Placement of cells
Queue of rectangles $Q = \emptyset$

1: Identify $\gamma$-overfilled bins and cluster them // Fig. 2(a)
2: foreach cluster $c$ do
3:   Find a minimal rectangular region $R \supset c$ with density($R$) $\leq \gamma$
4:   $Q$.enqueue($R$)
5:   while !$Q$.empty() do
6:     $B$=$Q$.dequeue()
7:     $M$={movable cells in $B$}
8:     Find axis-aligned cutline $C_c$ to evenly split cell area in $M$
9:     Find axis-aligned cutline $C_b$ to evenly partition $B$
10:    $(S_0, S_1)$={two sub-regions of $B$ created by cutline $C_c$}
11:    $M_0$={movable cells in $S_0$}
12:    $M_1$={movable cells in $S_1$}
13:    Perform NON-LINEAR SCALING on $M_0$ $\perp$ to $C_b$ // Fig. 3
14:    Perform NON-LINEAR SCALING on $M_1$ $\perp$ to $C_b$ // Fig. 3
15:    if Area($B$) $> 0.001\cdot$LayoutArea then
16:      $(B_0, B_1)$={two sub-regions of $B$ created by cutline $C_b$}
17:      $Q$.enqueue($B_0$)
18:      $Q$.enqueue($B_1$)
19:    end if
20:  end while
21: end foreach
```

Handling density constraints. For each grid bin of a given regular grid, we calculate the total area of contained cells A_- and the total available area of cell sites A_+. A bin is *γ-overfilled* if its *cell density* A_-/A_+ exceeds given density limit $0 < \gamma < 1$. Adjacent γ-overfilled bins are clustered by Breadth-First Search (BFS), and rough legalization is performed on such clusters. For each cluster, we find a minimal containing rectangular region with density $\leq \gamma$ (these regions can also be referred to as "clusters"). A key insight is that overlap removal in a region, that is filled to capacity, is more straightforward because the absence of whitespace leaves less flexibility for interconnect optimization. If relative placement must be preserved, overlap can be reduced by means of x- and y-sorting with subsequent greedy packing. The next step, *non-linear scaling*, implements this intuition, but relies on cell-area cutline C_c chosen in Algorithm 1 and shifts it toward the median of available area C_b, so as to equalize densities in the two sub-regions (Figure 2).



An overfilled bin
Bin cluster for rough legalization(B)
Cell-area median(C_c)
Region-area median(C_b)
B_0
B_1
(a)
(b)

Fig. 2. Clustering of overfilled bins in Algorithm 1 and adjustment of cell-area to region-area median by non-linear scaling (also see Figure 3). Movable cells are shown in blue, obstacles in solid gray.

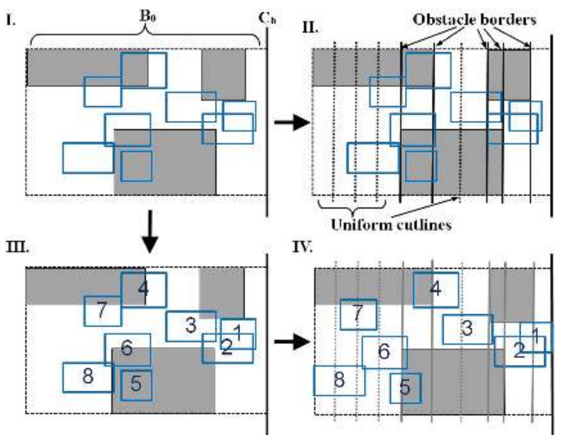


Fig. 3. Non-linear scaling in a region with obstacles (I): the formation of C_b-aligned stripes (II), cell sorting by distance from C_b (III), greedy cell positioning (IV).

Non-linear scaling in one direction is illustrated in Figure 3, where a new region was created by a vertical cutline C_b during rough legalization. This region is further subdivided into vertical stripes parallel to C_b. First, cutlines are drawn along the boundaries of obstacles present in this region. Each vertical stripe created in this process is further subdivided (by up to 10 evenly distributed cutlines) if its width exceeds $1/10$ of the region's width. Movable cells in the region are then sorted by their distance from C_b and greedily packed into the stripes in that order. For each stripe, we calculate the available site area A_+ and consider the stripe filled when the area of assigned cells reaches γA_+. Cell locations within each stripe are linearly scaled from current locations (non-linearity arises from different scaling in different stripes).

Rough legalization applies non-linear scaling in alternating directions, as illustrated in Figure 4 on one of ISPD 2005 benchmarks. Here, a region R is selected that contains overfilled bins, but is wide enough to facilitate overlap removal. R is first partitioned by a vertical cutline, after which non-linear scaling is applied in the two new sub-regions. Subsequently, rough legalization (Algorithm 1) considers each sub-region individually and selects different horizontal cutlines. Four rounds of non-linear scaling follow, spreading

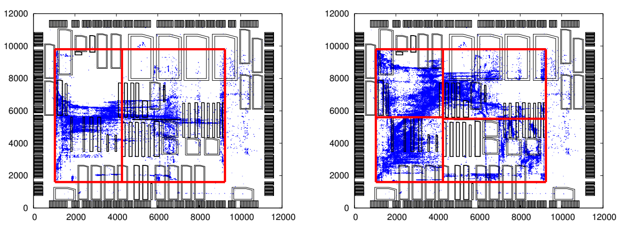


Fig. 4. Non-linear scaling after the first vertical cut and two subsequent horizontal cuts (ADAPTEC1) from intermediate steps between iterations 0 and 1 in Figure 7.

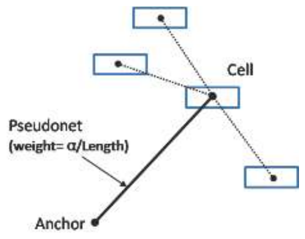


Fig. 5. An anchor with a pseudonet.

cells over the region's expanse (Figure 4).

Despite a superficial similarity to cell-shifting in FastPlace [22], our non-linear scaling does not use cell locations to define bins/ranges, or map ranges onto a uniform grid.

Cutline shifting. Median-based cutlines are neither necessary nor sufficient for good solution quality. We therefore adopt a fast cutline positioning technique from [17]. When obstacles cover <20% of a region's area, we find a cutline position C_c to minimize net cut, with <55% of cell area per partition. We record the ratio ρ of cell areas in the two partitions and adjust the region's C_b cutline to the position that partitions the region's area with the same ratio ρ.

C. Global Placement Iterations

Using legalized locations as anchors. Solving an unconstrained linear system results in a placement with significant amount of overlap. To pull cells away from their initial positions, we gradually perturb the linear system. As explained in Section IV-B, at each iteration of our global placement, top-down geometric partitioning and scaling generates a roughly legalized solution. We use these legalized locations as fixed, zero-area anchors connected to their original cells with artificial two-pin pseudonets. Furthermore, following the discussion in Section II, we note that connections to fixed locations do not increase the size of the Hessian matrix Q, and only contribute to its diagonal elements. This enhances diagonal dominance, condition number of $P^{-1}Q$, and the convergence rate of Jacobi-preconditioned CG.

In addition to weights given by the B2B net model on pseudonets, we control cell movement and iteration convergence by multiplying each pseudonet weight by an additional factor $\alpha > 0$ computed as $\alpha = 0.01 \cdot (1 + iterationNumber)$. At early iterations, small α values weaken spreading forces, giving greater significance to interconnect and more freedom to the linear system solver. As the relative ordering of cells stabilizes, increasing α values boost the pull toward the anchors and accelerate the convergence of lower bounds and upper bounds.

Grid resizing. To identify γ-overfilled bins, we overlay a uniform grid over the entire layout. The grid size is initially set to $S_{init} = 200 \times 200$ to accelerate

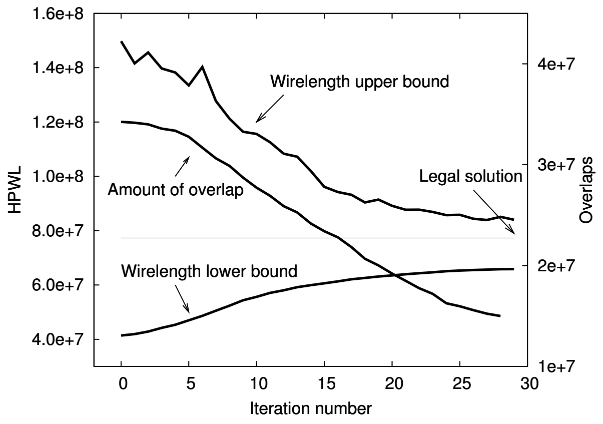


Fig. 6. Lower and upper bounds for HPWL, the amount of overlap at each iteration & HPWL of the legal placement (ADAPTEC1).

the rough legalization. However, in order to accurately capture the amount of overlap, the grid size decreases by $\beta = 1.06$ at each iteration of global placement until it reaches $2\times$ the average movable cell size. Grid resizing also affects the clustering of γ-overfilled bins during rough legalization (Section IV-B) effectively limiting the amount of cell movement and encouraging convergence at later iterations. A progression of global placement is annotated with HPWL values in Figure 7. The upper-bound placements on the right appear blocky in the first iteration, but gradually refine with grid resizing.

Convergence criteria. A convergence criterion similar to that in Section IV-A can be adopted in global placement. We alternate (1) rough legalization, (2) updates to anchors and the B2B net model, and (3) solution of the linear system, until HPWL of solutions generated by rough legalization stops improving. Unlike in the initial placement step, however, HPWL values of upper-bound solutions oscillate during the first 5-10 iterations, as illustrated in Figure 6. To prevent premature convergence, we monitor the gap between the lower and upper bounds. Global placement continues until the gap is reduced and stops improving. On the ISPD 2005 benchmark suite, this convergence criterion entails 26-35 iterations of global placement. The final set of locations (global placement) is produced by the last rough legalization as indicated in Figure 1.

V. Extensions and Improvements

The algorithm in Section IV can be improved in terms of runtime and solution quality.

A. Selecting Windows for Rough Legalization

During early global iterations, most movable cells of the lower-bound placement reside near the center of the layout region (Figure 7). In such cases, there is usually one expanded minimal rectangular region (cluster) that will encompass most of γ-overfilled bins. However, as global iterations progress, γ-overfilled bins will

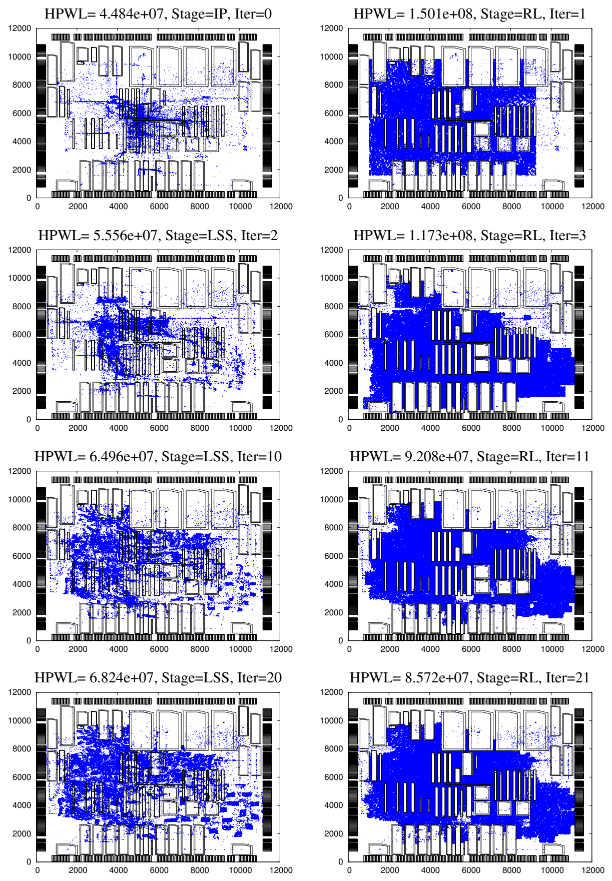


Fig. 7. A progression of global placement snapshots from different iterations and algorithm steps (adpatec1). IP=Initial Placement, RL=Rough Legalization, LSS=Linear System Solver. Left-side placements show lower bounds and right-side placements show upper bounds.

be scattered around the layout region, and multiple clusters of bins may exist. In our implementation, we process γ-overfilled bins in the decreasing order of density. Each expansion stops when the cluster's density drops to γ or the cluster abuts the boundaries of previously processed clusters. This strategy may generate incompletely expanded clusters, especially in mid-stages of global placement iterations. However, as the densest bins are processed first, the number of regions with peak density is guaranteed to decrease at every iteration except when the peak density itself decreases. At each iteration of global placement, rough legalization is repeated up to ten times until maximal density is decreased below γ.

B. *Speeding up Placement Using Parallelism*

Further speed-up is possible for SimPL on workstations with multicore CPUs. Runtime bottlenecks in the sequential variant of the SimPL algorithm (Section VI-A) — updates to the B2B net model and the CG solver — can be parallelized. Given that the B2B net model is separable, we process the x and y cases in parallel. When more than two cores are available, we split the nets of the netlist into equal groups that can be processed by multiple threads. To parallelize the CG solver, we applied a coarse-grain *row partitioning* [10] scheme to the Hessian Matrix Q, where different blocks of rows are assigned to different threads. A critical kernel operation in CG is the Sparse Matrix-Vector multiply (SpMxV). Memory bandwidth is a known performance bottleneck in a uniprocessor environment [9], and its impact is likely to aggravate when multiple cores access the main memory through a common bus. We reduce memory bandwidth demand of SpMxV by using the *CSR* (Compressed Sparse Row) [19] memory layout for the Hessian matrix Q.

As part of our empirical validation, we ran SimPL on an 8-core AMD-based system with four dual-core CPUs. Single-thread execution was compared to eight-thread execution. Solution quality did not appreciably change, but memory usage increased by 50% whereas runtime of global placement iterations was reduced by 2-3 times. The initial placement stage was accelerated by about 4 times. While CG remained the runtime bottleneck of SimPL on 8 cores, *rough legalization*, which we have not yet parallelized, became a close second ($> 20\%$). In addition to thread-level parallelism, our implementation makes use of SSE instructions (through g++ intrinsics) that perform several floating-point operations at once. However, the speed-up they provided to global placement was only several percent and not worth the development effort. The overall speed-up due to parallelism varies between different hardware systems, as it depends on the relation between CPU speed and memory bandwidth.

VI. Empirical Validation

Our implementation was written in C++ and compiled with g++ 4.4.0. Unless indicated otherwise, benchmark runs were performed on an Intel Core i7 Quad CPU Q660 Linux workstation running at 3.2GHz, using only one CPU core. We compared SimPL to other academic placers on the ISPD 2005 placement contest benchmark suite. Focusing on global placement, we delegate final legalization (into rows and sites) and detailed placement to FastPlace-DP [16], but post-process it by a greedy cell-flipping algorithm from Capo [5]. HPWL of solutions produced by each placer is computed by the GSRC Bookshelf Evaluator [1].

A. *Analysis of Our Implementation*

The SimPL global placer is a stand-alone tool that includes I/O, initial placement and global placement iterations. Living up to its name, it consists of fewer than 5,000 lines of C++ code and relies only on standard C++ libraries. There are four command-line parameters that affect performance — two for grid resizing (initial and step), and two for pseudonet weighting (initial

Benchmark size (#cells)		APLACE2.0		CAPO10.5		FASTPLACE3.0		MPL6		SIMPL	
		HPWL	Runtime	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ADAPTEC1	211K	78.35	35.02	88.14	25.95	78.16	2.50	77.93	18.36	**77.73**	**2.27**
ADAPTEC2	255K	95.70	50.57	100.25	36.06	93.56	3.66	92.04	19.91	**90.36**	**3.48**
ADAPTEC3	452K	218.52	119.53	276.80	78.19	213.85	8.48	214.16	58.92	**208.95**	**7.04**
ADAPTEC4	496K	209.28	131.57	231.30	79.32	198.17	7.10	193.89	55.95	**187.40**	**5.30**
BIGBLUE1	278K	100.02	44.91	110.92	41.78	**96.32**	**3.77**	96.80	22.82	97.42	4.01
BIGBLUE2	558K	153.75	100.96	162.81	80.55	154.91	9.62	152.34	61.55	**145.78**	**8.28**
BIGBLUE3	1.10M	411.59	209.24	405.40	182.94	365.59	21.59	344.10	85.23	**339.78**	**13.79**
BIGBLUE4	2.18M	871.29	489.05	1016.19	567.15	834.19	40.93	829.44	189.83	**808.22**	**35.80**
Average		**1.09×**	**14.77×**	**1.22×**	**13.65×**	**1.04×**	**1.22×**	**1.02×**	**6.41×**	**1.00×**	**1.00×**

TABLE I
LEGAL HPWL (×10E6) AND TOTAL RUNTIME (MINUTES) COMPARISON ON THE ISPD 2005 BENCHMARK SUITE. EACH PLACER RAN AS A SINGLE THREAD ON A 3.2GHZ LINUX WORKSTATION. HPWL WAS COMPUTED BY THE GSRC BOOKSHELF EVALUATOR [1].

and step). In all experiments we used default values described in Section IV.

Running in a single thread, SimPL completes the entire ISPD 2005 benchmark suite in 1 hour 20 minutes, placing the largest benchmark, BIGBLUE4 (2.18M cells), in 36 minutes using 2.1GB of memory. We report the runtime breakdown on BIGBLUE4 according to Figure 1, excluding 1.4% runtime for I/O.
Initial placement takes 5.2% of total runtime, of which 3.9% is spent in CG, and 1.3% in building B2B net models and sparse matrices for CG. **Global placement iterations** take 36.2%, of which 18% is in the CG solver, and 7.9% is in sparse matrix construction and B2B net modeling. Inserting pseudonets takes 1.3%, and rough legalization 9%. **Post-global placement** takes 57.2%, predominantly in detailed placement. Greedy orientation improvement and HPWL evaluation were almost instantaneous.

B. *Comparisons to State-of-the-art Placers*

We compared SimPL to other placers whose binaries are available to us. Our requests for NTUPlace3 binaries went unanswered, but NTUPlace3 results [7] are very similar to mPL6, which we compare to SimPL.

We run each available placer,[3] including SimPL, in *default mode* and show results in Table I. The HPWL results reported by APlace2 [12], Capo10.5 [18] and mPL6 [6] were confirmed by the GSRC Bookshelf Evaluator. However, FastPlace3 [22] reported lower HPWL by 0.25% to 0.95%. For consistency, we report the readings of the GSRC Bookshelf evaluator.

SimPL found placements with the lowest HPWL for seven out of eight circuits in the ISPD 2005 benchmark suite (no parameter tuning to specific benchmarks was employed). On average, SimPL obtains wirelength improvement of 8.26%, 18.70%, 3.85%, and 1.96% versus APlace2, Capo10.5, FastPlace3 and mPL6, respectively. SimPL was also the fastest among the placers on seven out of eight circuits, as well as on average. It is 6.4 times faster than mPL6, which appears to be the strongest pre-existing placer. SimPL is 1.22 times faster than FastPlace3.0, which has been the fastest academic placer so far.

While we managed to obtain almost all best-performing academic placers in binaries, RQL reportedly outperforms mPL6 in HPWL by a small amount [23]. Comparing our HWPL results to numbers in [23], we observe four wins for SimPL and four losses. RQL is 3.1 times faster than mPL6, making it more than twice as slow as SimPL.

VII. CONCLUSIONS AND FUTURE WORK

In this work, we developed a new, flat, force-directed algorithm for global placement. Unlike other state-of-the-art placers, it is rather simple, and our self-contained implementation includes fewer than 5,000 lines of C++ code. The algorithm is iterative and maintains two placements — one computes a lower bound and one computes an upper bound on final wirelength. These two placements interact, ensuring stability and fast convergence of the algorithm. The upper-bound placement is produced by a new *rough legalization* algorithm, based on top-down geometric partitioning and non-linear scaling, and converges to final cell locations. In contrast, all analytic algorithms we reviewed (both force-directed and non-linear) derive their final solution from a lower-bound placement.

The use of upper-bound placements offers a solution to the force-modulation problem [13], [23] and removes the need for the so-called *hold forces* used by several force-directed placers. As discussed in Section III, upper-bound placements perform an *area look-ahead*[4] that is instrumental in the handling of layout obstacles. APlace2, NTUPlace3, mPL6, as well as some force-directed placers, model obstacles by additional smoothened *penalty* terms in the objective function. Not only such terms introduce extra work, but they also add imprecisions to modeling. For similar reasons, SimPL avoids netlist clustering used by other placers. We have implemented several other techniques

[3]The KraftWerk2 binary we obtained did not run on our system.

[4]The concept of *area look-ahead* was proposed in [8] for block-packing by nested bisection, where it checks if a given bisection admits a legal block packing in each partition. Area look-ahead was not used in [8] to spread standard cells from dense regions.

essential to well-known placers, such as BoxPlace [13], ILR [22], and *ad hoc* force modulation [23], but they did not improve SimPL results.

SimPL is highly competitive on ISPD 2005 benchmarks where it outperforms every placer available to us in binary both by solution quality and runtime. Additional empirical results (not included due to page limitations) show that SimPL's runtime and solution quality advantages over FastPlace3 and mPL6 grow on larger netlists. Asymptotic complexity analysis of SimPL (not shown due to page limitations) suggests that each iteration runs in $O(n \log^2 n)$ time for a netlist with n movable standard cells. The number of iterations grows very slowly with n and does not exceed 40 in our experiments. Additionally, the algorithm can be improved to run in $O(n \log n)$ time.

SimPL's most compelling advantages over prior state of the art deal with practical uses of placement in modern timing-closure design flows: (1) the reduced complexity of SimPL allows for fast implementation, parallel processing, and effective software maintenance; (2) the upper-bound placements facilitate tighter integration of timing and congestion optimizations into the global placement process, improving the speed and efficacy of physical synthesis.

Attempting to explain theoretically the strong performance of our placement algorithm, we have noticed similarities to *primal-dual* algorithms for convex [24] and combinatorial [4] optimization. Primal-dual methods maintain lower and upper bounds, expressed by primal and dual solutions that eventually converge to an optimal feasible solution. The interpretation of duality as swapping the problem's constraints for the objective function [24] is also consistent with our algorithm — *legalization corresponds to imposing a no-overlap constraint while relaxing the linear constraints that capture the global minimum of the quadratic wirelength objective*. The key to the success of primal-dual algorithms [4], [24] is the observation that alternating progress in *primal* and *dual* solutions, i.e., improving the cost of feasible solutions and tightening the constraints for low-cost solutions, typically leads to faster convergence compared to one-sided optimizations. This effect is empirically observed in Section VI where SimPL is compared to pre-existing placement algorithms, all of which are one-sided.

Strong empirical results for the SimPL algorithm suggest further research on mixed-size, congestion-driven and performance-driven placement.

Acknowledgements. This work was partially supported by the US National Science Foundation (NSF) through a CAREER award, as well as by Sun Microsystems and Texas Instruments.

References

[1] S. N. Adya, I. L. Markov, "Executable Placement Utilities," http://vlsicad.eecs.umich.edu/BK/PlaceUtils/

[2] C. J. Alpert et al., "Techniques for Fast Physical Synthesis," *Proc. IEEE* 95(3), 2007, pp. 573-599.

[3] U. Brenner, M. Struzyna, J. Vygen, "BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms,"*IEEE TCAD* 27(9) 2008,pp.1607-20.

[4] N. Buchbinder, J. Naor, *The Design of Competitive Online Algorithms via a Primal-Dual Approach*, NOW Publishers, 2009.

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

[6] T. F. Chan et al, "mPL6: Enhanced Multilevel Mixed-Size Placement," *ISPD* 2006, pp. 212-214.

[7] T.-C. Chen et al.,"NTUPlace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints," *IEEE TCAD* 27(7) 2008, pp.1228-1240.

[8] J. Cong, M. Romesis, J.R. Shinnerl, "Fast floorplanning by look-ahead enabled recursive bipartitioning," *TCAD* 25(9), 2006, pp. 1719-1732.

[9] G. Goumas et al. "Understanding the Performance of Sparse Matrix-Vector Multiplication," *Euromicro Int'l Conf. on Parallel, Distributed and Network-based Processing* (PDP) 2008, pp. 283-292.

[10] L. Hsu et al. "Exploring the Cache Design Space for Large Scale CMPs," *ACM SIGARCH Computer Architecture News* 2005, pp. 24-33.

[11] B. Hu, M. Marek-Sadowska, "FAR: Fixed-points Addition & Relaxation Based Placement," *ISPD* 2005, pp. 161-166.

[12] A. B. Kahng, Q. Wang, "A Faster Implementation of APlace," *ISPD* 2006, pp. 218-220.

[13] A. A. Kennings, K. Vorwerk, "Force-Directed Methods for Generic Placement," *IEEE TCAD* 25(10), 2006, pp. 2076-2087.

[14] G.-J. Nam, S. Reda, C. J. Alpert, P. Villarrubia, A. B. Kahng, "A Fast Hierarchical Quadratic Placement Algorithm," *IEEE TCAD* 25(4), 2006, pp. 678-691.

[15] G.-J. Nam and J. Cong, *Modern Circuit Placement: Best Practices and Results*, Springer, 2007.

[16] M.Pan, N.Viswanathan, C.Chu, "An Efficient&Effective Detailed Placement Algorithm,"*ICCAD* 2005,pp.48-55.

[17] J. A. Roy, I. L. Markov, "ECO-System: Embracing the Change in Placement," *IEEE TCAD* 26(12) 2007, pp. 2173-2185.

[18] J. A. Roy et al. "Capo: Robust and Scalable Open-source Min-cut Floorplacer," *ISPD* 2005, pp. 224-226.

[19] Y. Saad, "Iterative Methods for Sparse Linear Systems," *SIAM* 2003.

[20] P. Spindler, U. Schlichtmann, F. M. Johannes, "Kraftwerk2 - A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model," *IEEE TCAD* 27(8) 2008, pp. 1398-1411.

[21] L. N. Trefethen and D. Bau "Numerical Linear Algebra," *SIAM* 1997, pp. 296-298.

[22] N.Viswanathan, M.Pan, C.Chu, "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control," *ASPDAC* 2007, pp. 135-140.

[23] N. Viswanathan et al. "RQL: Global Placement via Relaxed Quadratic Spreading and Linearization," *DAC* 2007, pp. 453-458.

[24] S. J. Wright, "Primal-Dual Interior-Point Methods," *SIAM* 1987.