Large-Scale Placement by Grid-Warping

Zhong Xiu, James D. Ma, Suzanne M. Fowler**, Rob A. Rutenbar
Dept. of ECE, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213 USA
**Intel Corp., Chandler, Arizona, 85224, USA
{zxiu,jdma,rutenbar}@ece.cmu.edu suzanne.m.fowler@intel.com

Abstract

Grid-warping is a new placement algorithm based on a strikingly simple idea: rather than move the gates to optimize their location, we elastically deform a model of the 2-D chip surface on which the gates have been roughly placed, "stretching" it until the gates arrange themselves to our liking. Put simply: *we move the grid, not the gates*. Deforming the elastic grid is a surprisingly simple, low-dimensional nonlinear optimization, and augments a traditional quadratic formulation. A preliminary implementation, WARP1, is already competitive with most recently published placers, *e.g.*, placements that average 4% better wirelength, 40% faster than GORDIAN-L-DOMINO.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids-placement and routing. G.4 [Mathematical Software]: Algorithm Design and Analysis

General Terms

Algorithms, Design

Keywords

Algorithms, Placement

1. Introduction

Circuit placement remains a critical step in the physical realization of any large design. Iterative improvement methods such as annealing [1] dominated in the 1980s, yielding to either quadratic/analytical methods [2]-[6] or mincut methods [7] in the 1990s. The last few years have seen an especially vigorous competition to evolve efficient analytical methods (e.g., [5,6,8,9]) to handle larger netlists, produce better wirelengths or better timing, or run faster. Debates between and linear wirelength estimation, between flat and hierarchical placement strategies, and among alternatives for embedding timing optimization, continue with equal vigor. Despite roughly two decades of impressive progress, the problem remains an important one to focus on. Much of the final performance—size, yield, cost, speed—of a modern IC implementation is determined by its placement.

In this paper we describe a novel placement algorithm. We start with the well-known quadratic point-placement formulation, and improve the layout via recursive subdivision, but most similarities to prior methods end here. Our idea is strikingly simple: rather than move the gates to optimize their location, we elastically deform a model of the 2-D chip surface on which the gates have been quickly and coarsely placed [11,14]. Put simply: *we move the grid, not the gates*. Rather than move each point individually, we "stretch" the underlying sheet until the points arrange themselves to our liking. This strategy has three advantages: (1) deforming the elastic sheet is a surprisingly simple, low-dimensional optimization problem; (2) freed of the need to rely on matrix solves as the sole engine of placement evolution, we can add optimization using powerful nonlinear methods, and choose any well-behaved objective function we like, for example, a combination of local congestion and exact half-perimeter wirelength; (3) this very big design problem is transformed from a very high-dimensional optimization task into a very large numerical cost function with a small number of degrees of freedom that determine the deformation of the placement grid. We call this *placement by grid-warping*.

As we shall see in the remainder of the paper, augmenting the traditional high-dimensional linearized solution step with a low-dimensional nonlinear improvement step—albeit one with an expensive-to-calculate objective function—turns out to be an attractive addition to make. However, the warped placement model creates some novel placement behaviors we must confront. For example, in most placers, the key problem is how not to incorrectly separate gates that wish to be close. In the warping model, this is less of a problem than determining *how* to make gates separate, since adjacent gates intrinsically stay close as the local surface deforms. In the sequel, we show how to solve these problems with a mix of new geometric optimization steps, and reuse of some existing heuristics from analytical placers. The overall structure of the placer is a quadratic analytical initial step serving to create a quick coarse placement in each (sub)region, followed by an improvement loop comprising the nonlinear numerical solution of a warping problem, followed by partitioning and recursion.

The rest of this paper is organized as follows: In Section 2, we give a brief qualitative motivation and description of how grid-warping works. In Section 3, we formulate in detail all the steps of the grid-warping placement algorithm. In Section 4, we offer detailed comparisons with several published placement algorithms to demonstrate the potential of our approach. Finally, Section 5 contains some concluding remarks and the directions of future work.

2. Grid-Warping: Motivation and Approach

Let us assume that we start with a conventional quadratic analytical placement [2,3,6], in which each gate to be placed is represented as a dimensionless point connected to a set of appropriately weighted 2-point wires. Overall squared Euclidean wirelength is the objective we minimize. (We shall describe more precisely our formulation in the following section.) This placement is, in some

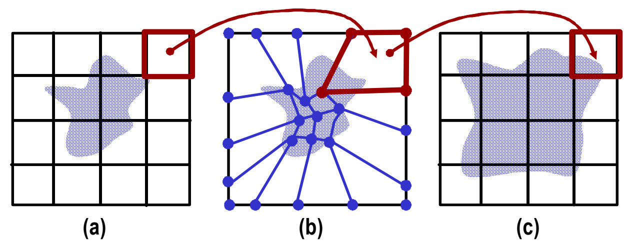


FIGURE 1. Basic warping concept. (a) An initial quadratic placement. (b) The placement grid itself is deformed, and each cell takes "ownership" of a new set of initially placed gates. (c) Deformation back to the original grid "warps" the gates into new locations.

mathematical sense, "optimal" with respect to wirelength. Unfortunately, however, cell sizes are not considered explicitly, overlaps are rampant, and 80% of the total gate area may be placed densely in a few hot spots comprising only 20% of the chip image.

This is the departure point for all subsequent efforts to make practical analytical placement techniques. How we formulate this *legalization problem* distinguishes prior efforts, and determines the overall success of each algorithm. Historically, several options have been suggested. One can use spatial recursion, and locate a balancing bisecting cut [2, 3] or quadrisecting cut [6], then recursively place each subregion. This requires *confinement* of the gates in each partitioned region; this can be accomplished by computing new pseudo-pin locations on region boundaries [2,6] for strict confinement, or adding center-of-gravity constraints for a looser confinement [3]. Another approach is to modify the objective or constraint formulation to address overlaps directly. One option is to add repulsion forces dependent on the local placement density [5] to a standard quadratic formulation. A different approach exploits ideas from multilevel algorithms, recursively aggregating/disaggregating the gates and handling gate overlaps directly, in a more general formulation similar to quadratic programming [8,9].

All these approaches use quadratic wirelength, or a linearized approximation thereof [10], and all except [8,9] use a large matrix solve as the essential engine for placement progress in each recursive or iterative solution step. Moreover, in all these approaches, the *gates* are the principal actors in the optimization: their (x,y) locations are the degrees of freedom we seek to optimize.

In contrast, in our approach it is the *space* on which the gates have been quadratically initially placed that is the focus of optimization. Figure 1 illustrates the basic idea. It is easiest to conceptualize "warping" as a uniform grid *above* the placement surface, with each grid intersection defining a *control point*. Warping elastically moves these control points to approximate some continuum deformation of the grid. In our model, each of these *unit cells* deforms so as to *acquire* a new set of initially placed gates by overlapping them, and then *pull* those gates back to its original location. Roughly speaking, the grid deforms, grabs the elastic placement sheet, and stretches it as it returns to its undeformed state. Thus, there are two essential operations: *warping* determines how the original grid deforms; *inverse warping* determines how each (x,y) gate location in the original placement is transformed back into a new location.

We note that while the term "gravity" is often abused in the placement literature, the analogy is reasonably good. The placement mass *warps* the space around it, attracting the underlying control points so as to spread out the placed gates in some optimal way. The gates, however, *never* move independently: they are each "spots" on the underlying elastic grid we use to model space. We deform this space directly, the placement mass moves as an indirect consequence.

Given just this simple overview, we can immediately see several important properties of grid-warping:

- **Low-dimensional**: The problem we optimize is how to deform the control points on the grid. Thus, the number of degrees of freedom of this optimization task is both small, and rather loosely coupled to the size of the netlist. Indeed, we can use the exact same formulation for 1,000 gates or 100,000 gates.
- **Flexibly nonlinear:** Given that the size of the nonlinear problem is modest, we have significant engineering choice in the form of the geometric warping transformations, and the overall objective function. In particular, since we are not restricted to a quadratic form (either classical [3, 6] or generalized [8,9]) we can directly optimize metrics regarded as mathematically difficult, for example, *exact* half-perimeter wirelength.
- **Expensive objective function:** The grid warping itself is a problem with a modest number of variables. However, each step of the nonlinear warping optimization must recalculate the objective function, which requires a full, flat evaluation of, for example, the global wirelength and local congestion. The essential tradeoff of grid-warping is to rely on the solution of a "small" nonlinear problem which has a "large" cost function that may be evaluated many times. As we shall see, this turns out to be an attractive tradeoff.
- **Locality preserving**: A critical problem in most placers is how *not* to separate gates that want to be nearby, while enforcing legalization constraints. Our "spots on an elastic sheet" model is intrinsically quite good on this metric, since it is the space itself that deforms, and gates *cannot* move independently. Of course, this is both a blessing and a curse. We often *need* the gates to move independently, to decongest a local hot spot, and this turns out to be a particular challenge in the design of the geometric warping transformations.

To expand briefly on this last point, the illustration of Figure 1 is a good conceptual model of grid-warping, but proves to be a poor model for the actual warping transformations. The need for nearby gates to be able to separate more independently is a significant problem in this model, one we solve in the following section. Nevertheless, the idea of a sheet of "unit" cells deforming to "acquire" new sets of gates, then "dragging" them back to their original home location, is a good mental model for the main idea of grid-warping.

3. Grid-Warping: Detailed Formulation

In this section, we work step by step through all the details of a grid-warping placer.

3.1 Quadratic Initial Placement

To put the initial "spots on the elastic sheet", we use a standard quadratic analytical placement formulation. A circuit netlist is represented for as a weighted hyper-graph, with $m = |M|$ vertices corresponding to gates and $n = |N|$ hyper-edges corresponding to signal nets. Initial placement seeks to assign all m movable gates of the design onto legal locations in a fixed-size two-dimensional layout region. Pad constraints fix the locations of certain vertices, while all others remain movable. Each net n is a set of pins and has a weight w_n. For each gate i, two variables (x_i, y_i) represent the x- and y-coordinates, respectively, of the center of the cell. As usual, a net connecting k gates yields a clique in the graph. A weight factor $1/(k-1)$ is used to prevent large nets from dominating the objective function.

We place to minimize squared Euclidean wirelength, so the distance between two connected gates i and j is $(x_i - x_j)^2 + (y_i - y_j)^2$. The two-dimensional problem is decomposed into independent horizontal and vertical placements, each minimizes the classical quadratic form:

$$\frac{1}{2}x^T Ax + b^T x + constant \tag{1}$$

where A is a symmetric and positive definite $m \times m$ matrix representing weighted connectivity, b is an m-dimensional vector representing fixed pad locations, and x (or y) is an m-dimensional vector representing the coordinates to be solved for. This has the familiar optimal solution $x = A^{-1}b$, obtainable via pre-conditioned Conjugate Gradients.

A common optimization here is *linear reweighting* [10] to better approximate a linear, rather than quadratic wirelength. This requires a sequence of additional linear solves (typically < 5). These extra solves are a consequence of the fact that the quadratic wirelength form, and its linear solution, are among the few optimization formulations that can scale to large placement problems. Grid-warping has no such limitation: we move space itself with a nonlinear model, and optimize half-perimeter wirelength explicitly. Hence, we do no linear reweighting. Our quadratic placement serves as the initial placement of the "spots on the sheet" for the subsequent warping improvement step.

3.2 Grid Warping with a Slicing-Style Unit Grid

The illustration of Figure 1 is a good starting point for how to formulate effective warping, but as we discovered, it has some significant limitations [11]. Let us first describe the advantages of this approach. The idea is to impose a regular *unit grid* on the surface of the placement, and regard the *(x,y)* intersections of the gridlines inside the placement, and at its periphery, as movable control points. Our goal is to arrange these control points under some suitable objective function so that an inverse warping transformation will "pull" an appropriate set of gates back to each original unit cell's location, and arrange these gates suitably inside each unit cell.

We can immediately use ideas from quadratic placement to formulate this problem: regard each control point as a movable object, and each edge between control points as a quadratic spring. Optimization re-weights each spring, thus changing the placement of the control points after a standard quadratic placement solve. Thus, an outer nonlinear optimization loop adjusts the weights on the edges, while an inner quadratic loop solves for the locations of the control points after each weight perturbation, and computes the appropriate gate location changes under some as-yet-to-be-described warping transformation. This problem is easy to formulate, and has attractive complexity: a $k \times k$ unit grid has $2(k+1)^2$ control points to be solved for, driven by changes in the weights on $2k(k+1)$ edges. A 4×4 grid, for example, creates a 40-variable nonlinear optimization.

Another extremely attractive feature of this formulation is that the placement surface is guaranteed to be partitioned into a set of equivalence classes—deformed unit grid cells—that are each a convex quadrilateral (or, at worst, a degenerate triangle [11]; see Figure 2). Transformation from one convex quadrilateral to another is a well-studied problem in computer graphics [13] and we can exploit any of several existing options for the required inverse warping transformation.

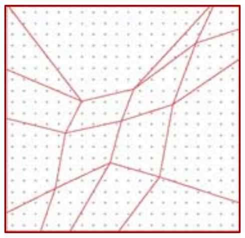


FIGURE 2. Example warping from uniform 4x4 unit grid.

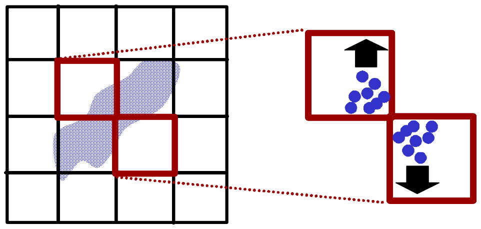


FIGURE 3. Uniform warping grid poorly handles the eccentric, off-axis placement mass; adjacent gates cannot easily *shear* in opposite directions.

What, then, is the problem? The problem, surprisingly, is that this formulation of the elastic grid is "too" continuous. It is extremely difficult for two points placed close together to move in opposite directions. This is essential for the unfortunately common case in Figure 3, where the initial placement mass is a highly eccentric ellipse with its major axis at a large angle to the coordinate axes. Nearby gates may warp into adjacent unit cells, but be required to move in *opposite* directions. This uniform 4-connected mesh model is poor at supporting such "shearing" motions during placement. Implementations based solely on such a grid model perform poorly on wirelength [11].

There is a simple and elegant modification to the basic unit grid that rectifies the problem. We impose now a $2^k \times 2^k$ grid, but regard the grid lines as a set of conventional *slicing* cuts, as from a slicing tree [12]. Figure 4 shows the idea, with slight dislocations of the grid edges added to explicitly highlight the slicing structure. More importantly, given a fixed horizontal/vertical ordering for the cuts (*i.e.*, first cut top-to-bottom), it is also simple to allow the slices to be arbitrary *oblique* cuts, as in Figure 4(b). We need exactly 2 variables to describe each cut-line, and these can be specified as *relative* fractional-valued distances in [0,1] along the edges of the parent region being sliced. Orthogonal cuts yield rectangular regions, oblique cuts yield quadrilateral regions, and we again divide the space into an equivalence partition of convex quadrilaterals. The 2×2 case, with exactly 6 optimization variables, appears in Figure 4(c). The $2^k \times 2^k$ slicing-style unit grid requires $2(4^k - 1)$ variables. Thus, the 4×4 grid requires only 30 variables whose values are to be optimized. We shall solve for these with a novel nonlinear formulation, described in the next two sections.

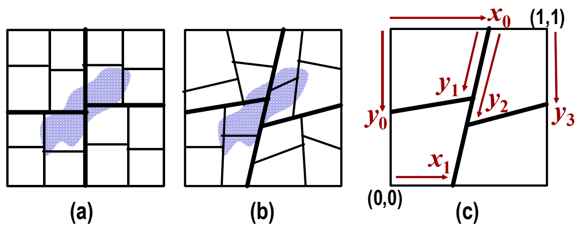


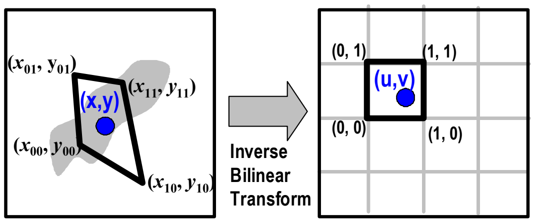
FIGURE 4. Slicing-style warping grid formulation. (a) 4x4 unit grid. (b) 4x4 grid after warping. (c) Optimization variables labeled for 2x2 slicing grid.

3.3 Grid Warping Unit-Cell Transformation

Our next task is actually to *warp* the space, thereby allowing each unit cell in the grid to move to overlap and "acquire" a new set of gates. Warping is physically a three-step process: first, we change the location of each cutline in the slicing-style unit grid, allowing each unit cell to deform and overlap different gates; second, we map all the gates newly overlapped back to a new location inside the undeformed original unit cell; third, we recalculate an objective that measures how well the gates have rearranged themselves. Thus, the next problem is the geometry of how one unit cell is warped.

Our solution is shown in Figure 5. The computer graphics literature is rich with examples of ways to transform between a convex quadrilateral and a unit square, *e.g.*, [13]. We obtained the best results with an *inverse bilinear transform* [14]. Bilinear mapping [13] is a simple, proportional geometric transform, commonly defined as a mapping of a square into a quadrilateral. The forward transform preserves lines which are horizontal or vertical in the source square, and preserves equispaced points along such lines. We actually need the *inverse* bilinear mapping to map back from our warped unit cell to the uniform grid. The inverse mapping can be derived by solving two simple quadratic equations, as in Figure 5.

One implementation detail worth noting is how we efficiently determine which gates are "acquired" by each warped cell, as optimization deforms each unit grid. Given that we expect a large number of gates, and a large number of evaluations of our overall objective function, this must be done very efficiently. We use a modified scanline algorithm [15] to associate each placed gate with the unique warped unit cell that overlaps it. The edges of the warped cells determine the boundaries of each unique warping transformation; we treat them as the edges of a polygon, labeled so that we can always tell "inside" and "outside". We could use a conventional scanline and add each individual gate location, as well as the warped unit cell edges, to the algorithm, and advance the scanline gate by gate. This is, however, much too inefficient, especially since we have many gates, but a relatively small number of grid edges. Hence, we partition the placement into yet another grid we refer to as the *source grid*. We now use a block-oriented scanline which advances row-by-row up the grid, and visits the gates grid by grid, left to right across the columns [14]. The basic idea is that many of these source grid cells will be completely contained in one warped unit cell, and so we know we can apply the same inverse bilinear transform to each gate. Only a relatively small number of source grid cells will actually cross the edge of a warped cell, and so only those cells require the detailed process of discerning exactly which side of the cutline edge they belong to, and thus which inverse bilinear transform to apply to map each gate back to some original unit grid.



Solve for (u,v): $Au^2 + Bu + C = 0$ $Dv^2 + Ev + F = 0$

where $A = af\text{-}be$ $B = ex\text{-}ay+ah\text{-}de+cf\text{-}bg$ $C = gx\text{-}cy+ch\text{-}dg$

$D = ag\text{-}ce$ $E = ex\text{-}ay+ah\text{-}de\text{-}cf+bg$ $F = fx\text{-}by+bh\text{-}df$

and $a = x_{00} - x_{10} - x_{01} + x_{11}$ $e = y_{00} - y_{10} - y_{01} + y_{11}$

$b = -x_{00} + x_{10}$ $f = -y_{00} + y_{10}$

$c = -x_{00} + x_{01}$ $g = -y_{00} + y_{01}$

$d = x_{00}$ $h = y_{00}$

FIGURE 5. Transforming an "acquired" gate at (x,y) in a warped unit cell back to location (u,v) inside the original unit cell via inverse bilinear transform.

3.4 Warping Objective Function and Optimizer Engine

We now know how to represent the placement space as a slicing-style unit grid, and that this grid can be deformed by specifying the values of a modest number of variables (e.g., 6 for a 2x2 grid, 30 for a 4x4 grid, etc.). We now need to choose an objective function to optimize, and a nonlinear solution method.

For the solver itself, we use a classical Brent-Powell engine, in the style of [16]. The choice is motivated by the fact that our problems are small, and we lack derivatives or, indeed, guarantees of continuity of any objective function, given the discrete nature of the warping process. A small change in the variables specifying the location/orientation of each slicing cutline can change the shape and location of the deformed quadrilateral of each warped unit cell, which in turn can add or remove any number of discrete gates from this cell. A derivative-free optimizer is a good choice here, and we find the basic Brent-Powell formulation performs well, even though it is only a local optimizer. We start the optimization with each cutline variable set to value 0.5, i.e., with a perfectly uniform grid of unit cells. The engine converges to a good nearby local optimum, usually making several thousand calls to the objective function.

For the objective function, we use a weighted linear combination of wirelength and congestion. Here, we can see again one of the advantages of using a nonlinear optimization to evolve the placement: we can use any well-behaved functional form here:

$$Cost = Wirelength + W \times CongestionPenalty \qquad (2)$$

We use half-perimeter for the wirelengh, and a penalty function formulation for the congestion that reuses the source grid mentioned earlier. Each source cell ij contributes a penalty C_{ij} based on whether the number of gates mapped to its region exceeds a specified capacity (the total number of gates m divided by the number of source cells $|C|$; call this κ). Let m_{ij} be the number of gates in cell ij, then:

$$C_{ij} = \begin{cases} 0 & if\ m_{ij} \in [0.95\kappa, 1.05\kappa] \\ (m_{ij} - \kappa)^2 & if\ m_{ij} \in [0.85\kappa, 0.95\kappa] \cup [1.05\kappa, 1.15\kappa] \\ M + (m_{ij} - \kappa)^2 & otherwise \end{cases} \qquad (3)$$

Regions with far too many, or too few gates, always incur a large baseline penalty (M) which grows as demand differs from capacity. However, as we near the capacity, the penalty is moderated, and within 5% of the correct capacity, it vanishes. Warping deforms space so that, after each gate is mapped to its new location, each unit grid has roughly the same number of gates in it, while striving to ensure the wirelength is not too compromised.

3.5 Decomposition and Recursion

Grid-warping still relies on recursive decomposition, since we need to keep the size of the warping grid small enough for quick nonlinear optimization. Thus, each cell in the slicing-style unit grid becomes a new problem for placement by grid-warping. We typically use either a 2x2 or a 4x4 slicing-style unit grid for warping.

This means that we need to formulate a way to confine the cells inside each decomposed region, so that we can again run an initial quadratic placement to begin warping each subregion. To do this,

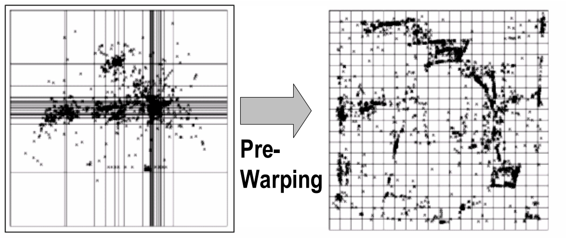


FIGURE 6. Pre-warping the initial quadratic placement with a 20x20 nonuniform gridding.

we propagate pins from other gates in external regions to the boundary of the region being optimized, using the method from [6]. Roughly speaking, we propagate each external gate to the closest point on the boundary of the rectangular region we are optimizing, and proceed forward with optimizing the gates in each region, connected now to new pins on its boundary.

We also borrow one other technique from prior methods: the use of mincut partitioning to disambiguate gates placed very close to our cutlines [3]. We use the hMetis engine [7] in regions ranging from 10-25% of the dimension of the unit cell. Note that 2x2 grid-warping is essentially a quadrisecting cut, albeit one with the twin novelties of cutlines at arbitrary angles, and no requirement that all the cuts meet at a common central point. An advantage of warping is that we free the quadrisection (or even higher-dimensional cut) step from the artificial constraint that each cut is axis parallel. Quadratic placement certainly does not arrange gate clusters so that they form perfect axis-parallel rectangular blocks, and we see no reason to assume that the recursive balancing cuts need to be similarly restricted.

3.6 Geometric Pre-Conditioning: The Pre-Warping Step

The algorithm as defined so far is complete, but not optimal. Experiments showed that the success of warping is extremely dependent on the *density* of the initial quadratic placement: a placement with very dense hot-spots and large empty regions is quite difficult to warp to achieve a more uniform distribution of gates across the chip surface. This is, in fact, *another* reason why we avoid linear reweighting, which tends to cluster gates during initial placement even more densely than a pure quadratic metric.

Our solution is a special geometric pre-conditioning step we call *pre-warping* [11]. The idea is simple: we compute a non-uniform gridding such that each grid row and column has the *same* number of gates, and use this to spread the gates more uniformly, and later rely on warping to repair any artifacts we introduce.

To build a non-uniform PxP grid, the placement surface is swept twice. First, it is swept from left to right, calculating the width of each grid column as the distance swept until the next $1/P$ of the total gate area has been seen. For example, if this grid is 20×20, each step sweeps a sorted list of the gates until the next 1/20th of the gate area has been seen. This process is repeated, except now sweeping

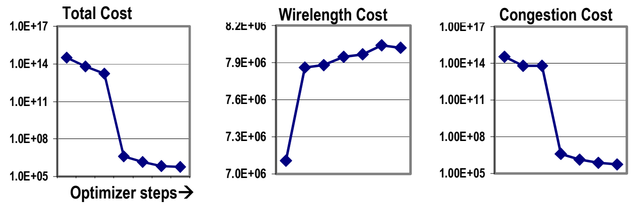


FIGURE 8. Progress of warping as measured by cost function components in Powell outer optimization loop for ibm06 benchmark using a 2x2 warping grid. Warping makes 5468 total calls to the cost function.

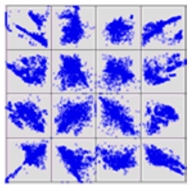


FIGURE 9. Example placement snapshot after recursive decompositions.

from top to bottom. The result is the nonuniform grid shown in Figure 6(left). We then simply linearly *stretch* each row and column of this nonuniform grid, and the gates therein, to *make* it uniform, as in Figure 6(right). This is fast, and surprisingly effective.

4. Experimental Results

We have implemented these ideas into a prototype placer called **WARP1**. With all the steps of our algorithm defined, we first show a few isolated WARP1 examples to give a better sense of just how grid-warping works. Figure 7 shows several snapshots of the progress of top-level warping for the ibm06 benchmark from [17], using a 4x4 warping grid. Figure 8 shows the cost function as the nonlinear optimization runs at top-level for the same benchmark using a 2x2 grid. As we can see, warping arranges the gates in a more uniform way (better congestion) while minimally degrading the wirelength. This proves to be a good tradeoff, and sets up the recursive decomposition to repeat the process in each warped unit cell. Figure 9 shows the placement after a few recursion steps.

Table 1 shows detailed quantitative comparisons between WARP1 and several state-of-the-art published placers. We use the ISPD benchmarks from [17] (ranging from roughly 10,000 to 200,000 gates) with 10% total white-space, uniform cell sizes, no routing channels, and random pad locations. We run on a 1.6Ghz LINUX machine. Following [8], we also use DOMINO [4] for final legalization after warping placement. Although we regard WARP1 as a still preliminary implementation of an immature algorithm, our results are already entirely competitive with several more mature placement engines. In particular, WARP1 averages 4% less wirelength than GORDIAN-L-DOMINO [3,4] running in its maximum quality mode (with several reweighting steps [10]), and runs roughly 40% faster.

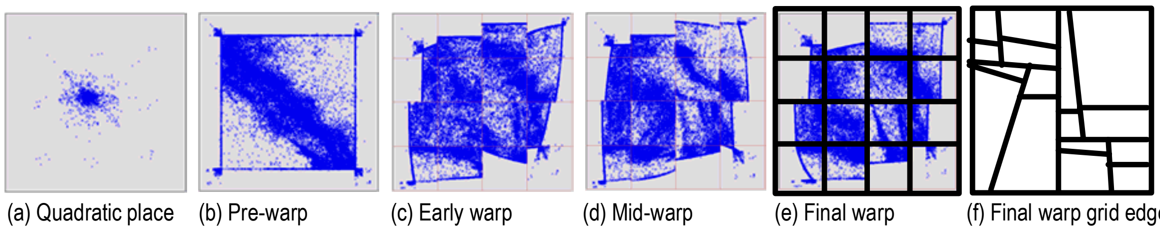


FIGURE 7. Progress through grid-warping flow for top-level for the ibm06 benchmark, using an 8x8 pre-warp grid, and a 4x4 unit slicing grid for warping.

As expected, we do a bit better against CAPO [18], though we are slower than this very fast mincut engine; similarly, we do slightly less well than DRAGON's annealing placer [19], though roughly 4X faster. Comparisons with mPL2 are still in progress, but we note that their most recent version [9] produces wirelengths about 2% better than GORDIAN-L-DOMINO on a set of benchmarks that differ only slightly (e.g., pad locations and channel spacings [20]). Another promising observation is that, unlike other placers, WARP1 results are consistently superior to GORDIAN on *every* benchmark in the ISPD98 suite, without use of *any* linear reweighting [10]. We hope this bodes well for our ongoing efforts to improve the algorithm. We regard this as an extremely satisfactory outcome for a new placement algorithm which is still the subject of ongoing research.

5. Conclusions

Grid-warping is a new placement algorithm based on a simple idea: rather than move the gates to optimize their location, we elastically deform a model of the 2-D chip surface on which the gates have been roughly placed, "stretching" it until the gates arrange themselves to our liking. Deforming the elastic grid is a simple, low-dimensional nonlinear optimization, and augments a traditional quadratic formulation. A preliminary implementation, WARP1, is already competitive with most recently published placers, e.g., 4% better wirelength, 40% faster than GORDIAN-L-DOMINO. We believe there is significant potential in the fundamental "warping" concept for further improvement. We are currently examining options for incorporating timing, net-based congestion estimators and fixed pre-placement of macroblocks.

Acknowledgments

We thank Frank Johannes of the Technical University of Munich for access to GORDIAN and DOMINO; Jason Cong of UCLA for discussions about versions of the ISPD benchmarks; and Juergen Kohl of IBM for illuminating discussions about quadratic placement. This work was supported in part by an Intel graduate fellowship, the Semiconductor Research Corporation, and the Pittsburgh Digital Greenhouse.

References

[1] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, 13 May 1983.

[2] R. S. Tsay, E. Kuh, C. P Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design & Test of Computers,* vol.5, Dec. 1988.

[3] Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "Gordian: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. CAD*, vol. 10, no.3, March 1991.

[4] K. Doll, F. M. Johannes, K. J. Antreich, "Iterative placement improvement by network flow methods," *Proc. IEEE Trans. CAD*, vol. 13, no. 10, Oct 1994.

[5] H. Eisenmann, F. M. Johannes, "Generic global placement and floorplanning," *Proc ACM/IEEE DAC*, June 1998.

[6] J. Vygen, "Algorithms for large-scale flat placement," *Proc ACM/IEEE DAC*, June 1998.

[7] G. Karypis, R. Agarwal, V. Kumar, S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI design," *Proc ACM/IEEE DAC*, June 1997.

[8] T. F. Chan, J. Cong, T. Kong, J. R. Shinner, "Multilevel optimization for large-scale circuit placement.," *Proc. ACM/IEEE ICCAD*, Nov. 2000.

[9] T. F. Chan, J. Cong, T. Kong, J. R. Shinner, K. Sze, "An enhanced multilevel algorithm for circuit placement," *Proc. ACM/IEEE ICCAD*, Nov. 2003

[10] G. Sigl, K. Doll, F. M. Johannes, "Analytical placement: A linear or a quadratic objective function?" *Proc ACM/IEEE DAC*, June 1991.

[11] S. M. Folwer, *Placement by Grid Warping.* Master's Thesis, ECE, Carnegie Mellon University, 2001.

[12] R. H. J. M. Otten, "Efficient floorplan optimization," *Proc. IEEE ICCD*, 1983.

[13] P. S. Heckbert, *Fundamentals of Texture Mapping and Image Warping*, Master's Thesis, EECS, U.C. Berkeley, UCB/CSD-89/516, 1989.

[14] Z. Xiu, *VLSI Component Placement by Grid Warping,* Master's Thesis, ECE, Carnegie Mellon University, 2003.

[15] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications,* Springer-Verlag, 1997.

[16] W. H. Press, *et al., Numerical Recipes in C: The Art of Scientific Computing,* Cambridge University Press, 1992.

[17] C. J. Alpert, "The ISPD98 circuit benchmark suite," *Proc. ACM ISPD*, April 1998.

[18] A. Caldwell, A. Kahng, I. Markov, "Can recursive bisection alone produce routable placements?" *Proc. ACM/IEEE DAC*, June 2000.

[19] M. Wang, X. Yang, M. Sarrafzadeh, "Dragon 2000: Fast standard-cell placement for large circuits," *Proc. ACM/IEEE ICCAD*, Nov. 2000.

[20] J. Cong, UCLA, private communication, Nov. 2003.

Benchmark	Warp1/Domino		Gordian-L/Domino		Capo 8.8		Dragon 3.01	
	Wirelength	CPU Time (s)	Wirelength	CPU Time (s)	Wirelength	CPU Time (s)	Wirelength	CPU Time (s)
IBM01	1.35e6	159.69	1.35e6	175.8	1.40e6	67.99	1.33e6	929.51
IBM02	3.18e6	298.97	3.43e6	533.9	3.36e6	130.85	3.02e6	1442.62
IBM03	4.10e6	348.41	4.26e6	508.0	4.32e6	174.32	3.93e6	1511.12
IBM04	4.80e6	499.37	4.88e6	521.9	5.09e6	193.54	4.64e6	2616.18
IBM05	8.31e6	363.15	8.36e6	1045.0	8.41e6	242.31	7.89e6	4103.36
IBM06	4.75e6	521.68	4.92e6	1001.5	5.47e6	242.07	4.93e6	3394.14
IBM07	7.61e6	918.01	8.04e6	1082.2	7.85e6	369.62	7.26e6	2527.12
IBM08	8.64e6	1397.28	8.90e6	2420.4	8.93e6	382.04	7.92e6	4960.72
IBM09	8.51e6	1211.07	8.70e6	1389.1	8.55e6	434.09	8.24e6	4930.11
IBM10	1.36e7	2198.24	1.40e7	2227.8	1.42e7	647.00	1.30e7	6864.32
IBM11	1.25e7	1632.94	1.27e7	1881.0	1.29e7	644.57	1.16e7	5455.47
IBM12	1.72e7	2184.38	1.78e7	2359.3	1.82e7	725.27	1.64e7	7525.49
IBM13	1.53e7	2299.86	1.57e7	2809.5	1.58e7	862.45	1.48e7	7278.71
IBM14	2.87e7	5507.48	2.99e7	6237.7	2.95e7	1616.38	2.80e7	11120.74
IBM15	3.62e7	7500.91	3.77e7	8529.2	3.75e7	2101.87	3.40e7	15054.41
IBM16	3.72e7	7698.99	3.90e7	9611.4	3.93e7	2234.98	3.66e7	16493.09
IBM17	4.96e7	7739.09	5.15e7	10798.3	5.14e7	2349.72	4.87e7	32464.90
IBM18	3.73e7	8570.13	3.92e7	14274.9	3.89e7	2536.15	3.60e7	28618.87
Ratio	**1.00**	**1.00**	**1.04**	**1.41**	**1.05**	**0.38**	**0.96**	**4.17**

TABLE 1. Placement results comparing Warp1 with Gordian, Capo, and Dragon.