Timing-Driven Placement by Grid-Warping

Zhong Xiu, Rob A. Rutenbar

Dept. of ECE, Carnegie Mellon University,
Pittsburgh, Pennsylvania, 15213 USA
{zxiu,rutenbar}@ece.cmu.edu

Abstract

Grid-warping is a recent placement strategy based on a novel physical analogy: rather than move the gates to optimize their location, it elastically deforms a model of the 2-D chip surface on which the gates have been coarsely placed via a standard quadratic solve. In this paper, we introduce a timing-driven grid-warping formulation that incorporates slack-sensitivity-based net weighting. Given inevitable concerns about wirelength and runtime degradation in any timing-driven scheme, we also incorporate a more efficient net model and an integrated local improvement ("rewarping") step. An implementation of these ideas, WARP2, can improve worst-case negative slack by 37% on average, with very modest increases in wirelength and runtime.

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

Placement remains a critical step in the overall IC design process. Much of the final performance of a modern chip implementation -- its size, cost and speed -- is determined by its placement. There are four basic objectives for circuit placement. First, we must minimize the total wirelength to have any hope of routing the design. Second, we must achieve specified clock speed(s), to meet overall chip timing constraints. Third, we must manage congestion so that a complete routing is likely. Finally, we should meet all these objective as quickly as possible, even for extremely large designs. The interplay among these often incompatible constraints and objectives inside different placement strategies has shaped the last two decades of evolution for practical placement implementations.

For example, simulated annealing methods [1], which are flexible at handling complex constraints and produce very good wirelength, have largely disappeared because of their poor scalability for large designs. Quadratic/analytical methods [2]-[6], mincut methods [7] and multilevel/clustering methods [8,9] now dominate. Many efficient algorithms ([5,6,10,11]) have been proposed, emphasizing various improvements in wirelength, timing, or scalability. There remains some controversy about the "gap" between what we achieve in today's placers, and what we *might* achieve in the ideal case [12], but as a practical matter, these methods work well, are thus the subject of ongoing research to improve and extend their capabilities.

Recent work on legalization schemes for analytical placers has been particularly interesting. For example, Viswanathan and Chu's FastPlace [13] introduced an elegant set of engineering improvements to a force-directed scheme, with surprising speedups. Our own work focuses on the *grid-warping* strategy introduced in [14], which formulates placement as a recursive sequence of rough quadratic initial placements and nonlinear "elastic warping" steps in which the curvature of the underlying "fabric" of the placement is directly and nonlinearly optimized to relocate the gates to our liking.

Most new placement schemes target wirelengh as their first optimization target, and grid-warping is no exception. Our goal here is to show how to add a practical timing-driven component to the formulation of [14]. Existing timing-driven algorithms can be placed into two categories: *path-based* and *net-based*. Path-based algorithms generally have untenable complexity, given large designs with millions of cells. Net-based approaches adaptively assign higher weights to the more timing critical nets and use several iterations to improve timing. To minimize the number of these expensive iterations, an effective net weight assignment is critical.

The ability of analytical placers to respond globally to such weight changes is one of their most attractive features; the approach works well in practice [15]. However, the critical question for any analytical placer is how to use net weighting information *outside* of the core quadratic placement step. Grid-warping relies less on repeated large linear solves and min-cut partition improvement than most analytical placers: the nonlinear warping optimization is vital to the quality of its final results. How should we use net weights in this unique optimization step?

In this paper, we show how to formulate a timing-driven grid-warping placer using a recently introduced, accurate slack sensitivity analysis method for net weighting [20]. Given concerns about wirelength and runtime degradation in the timing driven case, we also describe more efficient net model and local improvement strategy that offer useful wirelength and runtime improvements. The rest of the paper is organized as follows: Section 2 describes background information on grid warping. In Section 3, we show how to reduce the wirelength and runtime by using a hybrid net model and a rewarping stage. Section 4 presents a timing-driven grid-warping algorithm using net weighting, along with experimental results from an implementation of these ideals called WARP2. Finally, Section 5 contains some concluding remarks.

DAC 2005, June 13-17, 2005, Anaheim, California, USA.

2. Background: Placement by Grid Warping

The underlying idea of grid-warping is 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. Put simply: warping moves the grid, not the gates. Rather than move each point individually, we "stretch" the underlying sheet until the points arrange themselves in a more optimal way.

Grid warping starts with a conventional quadratic analytical placement, 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. This quadratic placement serves as the initial placement of the "spots on the sheet" for the subsequent warping improvement step. Grid warping is distinguished by how it formulates the legalization problem; it is the space on which the gates have been initially placed which is the focus of optimization.

Conceptually, we put a uniform *nxn* 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. As the grid deforms, the elastic placement sheet deforms, and gates move. A nonlinear optimizer drives this deformation process. (In practice, we also use a somewhat more subtle formulation of the control grid, based on set of slicing style cuts.) The nonlinear optimization is low-dimensional because we need relatively few features to control the deformation. We optimize a cost function that is a weighted linear combination of wirelength and capacity penalty. And for the solver itself, we use a derivative-free local optimizer, since we lack derivatives and guarantees of continuity of any objective function.

Grid-warping still relies on recursive decomposition, since to keep the nonlinear optimization quick, we only use a 2×2 or a 4×4 control grid for warping. To confine the cells inside each decomposed region, we run a global quadratic solve at the beginning of each recursive layer, following the style of [6], but also enforce a center of gravity constraint in each subregion. This placement serves as another starting point for warping in each subregion. We also use ideas from mincut partitioning to disambiguate gates placed very close to the cutlines.

Two final pieces to mention are steps at the beginning and end of warping. A "pre-warping" step, which geometrically "conditions" the problem for an easier solution, spreads the gates more uniformly before each warping commences. And, like all analytical placers, warping requires a separate final legalization step. The implementation in [14] used DOMINO [4]. Figure 1 shows key steps in a grid-warp placement for the ibm06 benchmark from [14].

3. Reducing Wirelength in Grid Warping

Because the addition of a timing-driven component usually degrades both wirelength and runtime, we look first at two basic improvements to the core wirelength-only formulation of grid warping, which offer useful improvements.

3.1 Improved QP

Recall that in the standard quadratic analytical placement formulation, a circuit netlist is represented 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 is most common, a net connecting k gates yields a *clique* in the graph, with $O(k^2)$ connections. 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.

The clique model is the traditional model used in analytical placement algorithms. However, a superior alternative has recently been suggested. Viswanatha and Chu [13] prove the equivalence of a *hybrid* net model, which uses cliques for small nets, and stars--which decompose into just a linear number of 2-terminal connections--just for large nets. This is illustrated in Figure 2. In other words, for a k-pin net of weight W, if we set the weight of the two-pin nets intruded, to rW in the clique model and krW in the star model for any r, the clique model is equivalent to the star model. In their algorithm, they set r to 1/(k-1), use the star model for nets with four or more pins and use the clique model for nets with two or three pins. In the star model, for each net with more than four pins, an additional variable is introduced. Though this leads to more variables in the connectivity matrix A, the total number of non-zero entries in the matrix is greatly decreased. They demonstrated that over the ISPD-02 benchmarks, the Hybrid model leads to 2.95X fewer non-zero entries in matrix A as compared to the clique model, and on average, the total runtime of the placer is 1.5X lesser.

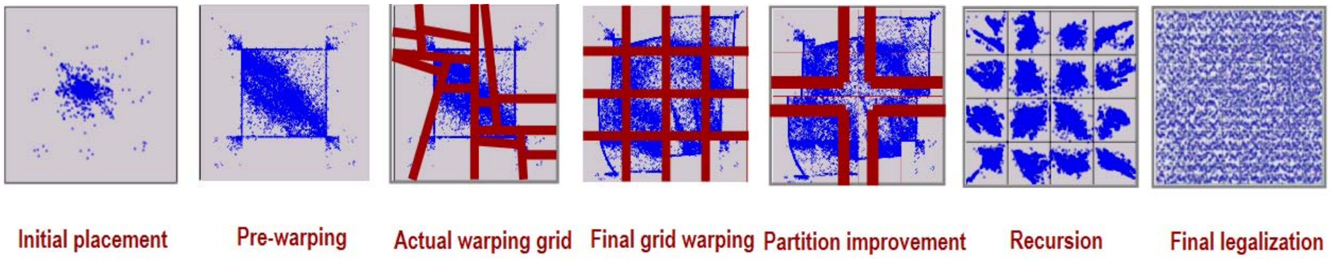
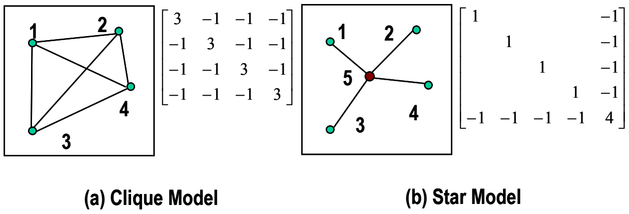


FIGURE 1. Progress through grid-warping flow for the ibm06 benchmark, using an 8x8 pre-warp grid, and a 4x4 unit slicing grid for warping.



$$\begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix} \qquad \begin{bmatrix} 1 & & & & -1 \\ & 1 & & & -1 \\ & & 1 & & -1 \\ & & & 1 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{bmatrix}$$

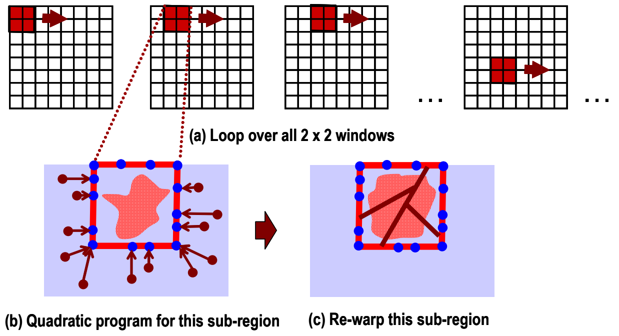
(a) Clique Model (b) Star Model

FIGURE 2. Clique model & star model. (a) Clique model produces 16 non-zero entries; (b) Star model produces 13 non-zero entries.

Following this idea, we replaced the clique net model by the hybrid net model in our grid-warping algorithm. As expected, the quadratic solve step achieved a speedup of approximately 2. But for the global quadratic solve in the second recursive layer and after, since we use center of gravity constraints to confine the movement of cells inside each subregion, the corresponding formulation is changed and the connectivity matrix is not sparse any longer [3, 6]. We found using the hybrid net model did not improve the runtime. However, in the new re-warping stage (described next) which locally improves the wirelength, many local quadratic solves are exploited. In each of these solves, the hybrid model is used to gain the speedup without sacrifice of performance.

3.2 Adding a Re-Warping Stage

Since grid-warping keeps the size of the warping grid small enough for quick nonlinear optimization, it still relies on recursive decomposition. This necessarily involves some loss of optimality for the global solution. To compensate, we introduce a new stage after each recursive layer, inspired by the local improvement step in Vygen's [6]. In our implementation, we call this *re-warping*. The idea is: at the end of each recursion layer (after the quadratic placement, the subsequent warping process and the partitioning improvement in this layer is done), we apply the following procedure to each 2×2 -subgrid of the current grid. (An $n \times n$ -grid contains $(n-1)^2$ 2×2 -subgrids.) All the cells belonging to the four respective subgrids need to be re-placed together again. A quadratic placement of just these cells is performed, with all the other cells outside the four respective subgrids propagated to the boundary of the four subgrids and without the center-of-gravity constraints. The cells inside may move freely within the union of the four subgrids. Then--in contrast to [6]--a warping step is applied to just this 2×2 window follows, yielding a *new* assignment of the cells to the four regions. Finally, mincut partitioning is used to reassign the cells near the cutlines. If the new assignments of cells produces a placement better than the original one in terms of weighted wirelength, it will be accepted. Otherwise the old placement is restored. Figure 3. illustrates the idea; Algorithms 1, 2 give details.



(a) Loop over all 2 x 2 windows

(b) Quadratic program for this sub-region (c) Re-warp this sub-region

FIGURE 3. Re-warping: (a) Loop over all 2x2 windows (b) Quadratic program for this sub-region (c) Re-warp this sub-region

Algorithm 1: ***Grid-Warping with an added re-warping stage***

```
1: Run the quadratic placement algorithm
2: Pre-warping stage
3: Run the nonlinear grid-warping loop
4: Partition improvement
5: If using a $4 \times 4$ grid, run re-warping
6: Repeat
    6.1: The global quadratic placement following the style of [6], with center-of-gravity constraints
    6.2: In each sub-region, repeat step 2, 3 and 4, with cells outside this sub-region propagated to the boundary of this sub-region
    6.3: If it is not the final layer, run re-warping: loop over all $2 \times 2$ windows twice from left to right and from top to bottom
Until each grid cell has no more than 30 gates
```

Algorithm 2: ***Re-Warping stage***

```
1: In the current $2 \times 2$ subregion, cells are considered movable, cells outside the sub-region are propagated to the nearest boundary
2: Run quadratic placement algorithm for this sub-region
3: Run pre-warping and grid-warping on this sub-region
4: Run partitioning improvement
5: Accept the new placement if the wirelength is better, otherwise restore the original placement
```

This stage loops over all possible 2×2 windows, and we try this stage twice with the same order: from left to right, from top to bottom. There are, however, two exceptions: (1) For the first recursive layer in a 2×2 grid-warping formulation, since all the gates are divided and placed into just four subgrid cells, re-warping is not needed until we decompose further. (2) After the final recursive layer of grid warping is completed, we do no re-warping, and rely instead on the fact that this placement will be fed directly into a final legalization engine to yield a legal row-structured placement.

Since the re-warping stage has the ability to fix some of the inferior decisions in grid-warping stage, we can terminate each higher level grid-warping optimization once we get a relatively good solution i.e., we loosen the convergence tolerance on this optimization, shorten the runtime, and enter the re-warping stage. As we shall see in the next section, the complexity of re-warping for each 2×2 sub-region is low compared with the overall placement, and allows to spend less time in each top-level warping optimization.

3.3 Experimental Results

WARP2 is an implementation of these two ideas, and extends the WARP1 placer from [14]. Table 1 compares results from WARP1 and WARP2. To be compatible with previous results, we use the ISPD98 benchmarks with the same modifications, and run on the same 1.6Ghz LINUX machine as in [14, 16]. We still use DOMINO for final legalization. On average, a re-warping stage gives 2% less wirelength than WARP1 and runs only 4% slower. But after the hybrid net model is also incorporated, the wirelength of WARP2 is the same as the wirelength of WARP1 with re-warping and WARP2 is 10% faster. So WARP2 averages 10% less runtime than WARP1 with 2% less wirelength.

We also run WARP2 on the ISPD98 benchmarks with some small differences (e.g., pad locations and channel spacings [8, 9, 16]) and

Bench-mark	Warp1/Domino		Warp1 with re-warping/Domino		Warp2/ Domino
	Wire-length	CPU Time (s)	Wire-length	CPU Time (s)	CPU Time (s)
IBM01	1.35e6	159.69	1.32e6	151.56	139.98
IBM02	3.18e6	298.97	3.13e6	322.52	265.47
IBM03	4.10e6	348.41	3.91e6	305.66	259.31
IBM04	4.80e6	499.37	4.73e6	401.02	361.70
IBM05	8.31e6	363.15	8.02e6	395.88	352.75
IBM06	4.75e6	521.68	4.59e6	613.38	457.55
IBM07	7.61e6	918.01	7.31e6	945.82	845.38
IBM08	8.64e6	1397.28	8.14e6	1301.91	1086.91
IBM09	8.51e6	1211.07	8.03e6	1249.68	1086.88
IBM10	1.36e7	2198.24	1.39e7	1863.28	1624.95
IBM11	1.25e7	1632.94	1.21e7	1669.37	1536.35
IBM12	1.72e7	2184.38	1.70e7	1942.79	1703.78
IBM13	1.53e7	2299.86	1.53e7	2351.02	2068.75
IBM14	2.87e7	5507.48	2.79e7	5985.22	5493.95
IBM15	3.62e7	7500.91	3.62e7	8152.68	7115.20
IBM16	3.72e7	7698.99	3.78e7	9318.45	8331.71
IBM17	4.96e7	7739.09	4.83e7	9975.80	8728.28
IBM18	3.73e7	8570.13	3.61e7	11454.33	10157.54
Ratio	**1.00**	**1.00**	**0.98**	**1.04**	**0.90**

TABLE 1. Placement results comparing Warp1 and Warp2

compare it with several state-of-the-art published placers. We perform these experiments on a 2.0Ghz Linux machine. Table 2 shows the results on this suite of benchmarks, now all normalized to the Warp2 results.

From the results showed in Table 2 we can see, Warp2 outperforms Gordian-L/Domino [3,4] in both wirelength and runtime. And we are 11% better than Capo [11] in wirelength, though we are 2.44 times slower. We do only 3% less well than the Dragon placer [17], but 3.53 times faster. And compared to mPL4 [8,9], we are 2% behind in wirelength with 19% better runtime. Note that we still use Domino as the final legalizer.

4. Timing-Driven Grid Warping

Existing approaches to minimize timing in placement can be generally divided into two classes: *path-based* and *net-based*. A typical path-based algorithm usually considers complete paths directly during the problem solution, so this class of algorithms usually maintain accurate timing information during optimization. But the complexity of such approaches are untenable for today's very large ASIC designs. Compared to path-based algorithms, net-based algorithms assign wire length bounds to critical nets or assign higher net weights to the nets on the timing-critical paths. As placement algorithms are often not suited to enforce bounds, the latter approach -- net weighting -- is the technique most commonly used [5,6,18,19]. The net weights are iteratively updated after each of (potentially) multiple placement runs. Of course, in a large chip with millions of cells, we strongly prefer not to have to run the complete placement engine more than a few times to find the right timing-based solution. Therefore, an effective net weighting method is critical to the success of timing driven placement algorithms.

We implement a timing-driven version of grid-warping by adopting a recently proposed slack sensitivity model for net weight calculation [20]. A popular way to assign net weight is based on the slack of the net; our ultimate goal is to minimize the worst negative slack (*WNS*) for the entire circuit. (Another figure of merit (*FOM*), defined as the total slack difference compared to a certain slack threshold for all timing end points, is considered to have equivalent importance in [20]; however, we only employ the *WNS* metric.)

4.1 Basic Formulation

A timing-driven grid warping placer uses sensitivity-based net weighting to update the weight of each net. The most important questions to answer are exactly where in the warping formulation these net weights appear, and whether they need to be transformed in some way across the various internal steps of our placer. As it turns out, it is very easy to incorporate net weighting all steps of the warping process:

- **Initial quadratic placement steps:** it is trivial to simply adjust the values in the A matrix to reflect the weights.
- **Nonlinear warping (and re-warping) steps:** although the geometric distortion that warping accomplishes is somewhat subtle, the cost function that warping optimizes is rather straightforward. We minimize a weighted combination of wirelength and capacity penalty (which ensures gates spread out uniformly). Since we adjust weight for complete k-terminal nets, we simply incorporate these weights in the overall wirelength term. Note, however, that in the warping step, we minimize a weighted bounding box wirelength, i.e., a more accurate *linear* model of wirelength, not a quadratic model.
- **Partition improvement:** net weights are similarly easy to incorporate in the partition improvement step, which helps disambiguate gates placed close to any of the cut lines. We use hMetis, which easily handles such weighting [7].

Algorithm 3 shows the overall flow. For efficiency, we run our warping algorithm twice and generate new net weights once. Specifically, we run our wirelength driven Warp2 placer with uniform weights for all nets. Then we run a static timing analysis on the near legal place-

Benchmark	Warp2/Domino		Gordian-L/Domino		Capo 8.8		mPL 4		Dragon 3.01	
	Wire-length	CPU Time (s)	Wire-length	CPU Time (s)	Wire-length	CPU Time (s)	Wire-length	CPU Time (s)	Wire-length	CPU Time (s)
IBM04-b	1.00	1.00	1.00	1.69	1.12	0.61	0.98	1.67	0.95	4.63
IBM07-b	1.00	1.00	1.03	1.16	1.12	0.47	1.02	1.16	0.98	2.84
IBM10-b	1.00	1.00	1.01	1.45	1.06	0.41	0.96	1.26	0.95	3.88
IBM17-b	1.00	1.00	1.00	1.21	1.13	0.28	0.98	0.97	1.00	3.48
IBM18-b	1.00	1.00	1.05	1.62	1.11	0.26	0.98	0.87	0.99	2.81
Ratio	**1.00**	**1.00**	**1.02**	**1.43**	**1.11**	**0.41**	**0.98**	**1.19**	**0.97**	**3.53**

TABLE 2. Placement results comparing Warp2 with Gordian, Capo, mPL and Dragon, all values are normalized with respect to Warp2/Domino

ment (before final legalization) to obtain the slack and wirelength for each net. For each multiple-pin net, the bounding box model is used for both the wirelength in placement and the net delay computation in the timer. Finally, after the new weights of all nets are updated, we run our warping placer again, to minimize the total *weighted* wirelength.

Algorithm 3: *Timing-driven Warp*

```
1: Run WARP2 with uniform net weight
2: Run static timing analysis tool to obtain timing information
3: Compute new weight for each net
4: Run timing-driven placement (WARP2) with new weights
```

4.2 Using Slack Sensitivity for Net Weights

For completeness, we review here briefly the formulation from [20] used in our placer. The slack sensitivity to net weight is defined as:

$$S_W^{Slk}(i) = \frac{\Delta Slk(i)}{\Delta W(i)} \tag{2}$$

where $Slk(i)$ and $W(i)$ are the slack and weight of net *i* respectively. Since only net *i* is changed, the slack change of net *i* comes from the delay change of et *i*. So,

$$S_W^{Slk}(i) = -\frac{\Delta T(i)}{\Delta W(i)} \tag{3}$$

where $\Delta T(i)$ is the nominal delay change of net *i*. Naturally, we can decompose Eqn. (3) into the following two terms.

$$S_W^{Slk}(i) = -S_L^{T}(i) S_W^{L}(i) \tag{4}$$

where $S_L^T(i)$ is the net delay sensitivity to wire length, and $S_L^T(i)$ is the wire length sensitivity to net weight:

$$S_L^T(i) = \frac{\Delta T(i)}{\Delta L(i)} \tag{5}$$

$$S_W^L(i) = \frac{\Delta L(i)}{\Delta W(i)} \tag{6}$$

where $L(i)$ is the wire length for net i. For bounding box model, we have:

$$T(i) = rcL(i) \tag{7}$$

where *r* and *c* are the unit length wire resistance and capacitance respectively. So we can obtain for net *i* the delay sensitivity to its wire length change as follows:

$$S_L^T(i) = \frac{\Delta T(i)}{\Delta L(i)} = rc \tag{8}$$

Following [20], we can obtain for net i the wire length sensitivity to its net weight change as below:

$$S_W^L(i) = -L(i)\frac{W_{src}(i) + W_{sink}(i) - 2W(i)}{W_{src}(i)W_{sink}(i)} \tag{9}$$

where *W*(*i*) is the initial weight of net *i*, $W_{src}(i)$ is the total initial weight on the driver cell of net *i* (the summation of net weights of those nets that intersect with the driver), and $W_{sink}(i)$ is the total initial weight on the receiver cell of net *i*.

To use the sensitivity results guide net weight assignment, first of all we need to set a target clock period. Then for those nets with negative slacks, we have:

$$\Delta W(i) = -Slk(i) S_W^{Slk}(i) \tag{10}$$

And we propose that the new weights should be:

$$W(i) = \begin{cases} W_{org}(i) & Slk(i) > 0 \\ W_{org}(i) + \Delta W(i) & Slk(i) \le 0 \end{cases} \tag{11}$$

In the real assignment process, we linearly scale *W*(*i*) to keep it in the range of [10, 60] for every circuit.

4.3 Timing Model and Static Timer

Unfortunately, simple wirelength, and not timing constraints, dominate the literature when placers are compared. The reason is the lack of an accessible, widely shared infrastructure for the many foundational components -- cell libraries, timing views, static timing engine -- necessary for careful comparison. To make our own results easier to compare against, we have adopted the infrastructure introduced by the OpenAccess Gear project [21,22,23]. OpenAccess ("OA") is an open source database system for EDA applications. The OA Gear project provides an open source, industrial-strength static timing engine (OA Gear Timer) and a set of open, hypothetical technology files complete with timing views.

Of central interest to us is the OA Gear Timer [22,23], an open source timer built on top of the OpenAccess database. OA Gear Timer uses standard techniques for static timing analysis. Both batch-mode (i.e., full, flat) timing analysis, and fast incremental timing analysis are available. Arrival and required arrival times and signal slew rates are maintained for all nodes in the circuit. Separate timing figures are kept for rising and falling signals. Internal gate delays utilize the standard interpolated two-dimensional lookup table based on output load and input signal slew rate. Industry standard file formats for timing libraries are supported, as are useful subsets of standard timing constraints, e.g., we can set the clock period, external delays on primary inputs and outputs, set the driving cell for inputs and set load capacitance on outputs.

We use the full OA Gear infrastructure since it provides a rich set of essential analysis tools and models, and should help others to make more comparisons with this work in future.

4.4 Experimental Results

We added timing-driven grid-warping to the improved WARP2 engine already described in Section 3. The high-level flow is as in Algorithm 3. We first run WARP2 with all net weights equal to 10. Then we run the OA Gear Timer to perform static timing analysis. Then we use the method described in section 4.2 to generate new net weights.

For the benchmarks, we use the new netlists provided by OA Gear in native OA format [22,23]. This suite includes a standard cell library along with the ISCAS89 sequential logic benchmarks. The cell library is hypothetical, but the timing and electrical parameters have been chosen to resemble a typical 250nm process. Table 3 shows the characteristics of some selected larger benchmarks in this suite. Table 4 compares the results from the wirelength-only version of WARP2 with the final timing-driven version of WARP2.

.

Design	Cells	Nets	PIs	POs	Registers
S13207	2680	2753	32	121	466
S15850	4565	4600	15	87	540
S35932	11587	11910	36	320	1728
S38417	14762	14838	29	106	1463
S38584	12221	12290	13	278	1292

TABLE 3. Benchmark Sizes and Characteristics

Since we cannot yet compare against other timing-driven placement algorithms on these benchmarks, we only compare the *WNS* and total wirelength of wirelength-only Warp2 with timing-driven Warp2. We can see that our algorithm performs well on this relatively small suite of benchmarks. On average, the timing-driven version of the placer improves the *WNS* by about 36.5% (given the clock period targets specified in the table), with only a very small percentage of wirelength increase--about 1% on average. The cost in increased runtime is also quite acceptable, and averages 47%. Of course, we can improve the timing further if we use additional placement/weighting iterations, at the cost of more runtime.

One final point is worth mention. Currently we still use Domino as the backend legalizer--even for this timing-driven version of Warp2. This is expedient, but clearly suboptimal. One reason the total wirelength does not increase much seems to be the fact that we are still minimizing the total *unweighted* wirelength in this stage. This suggests we may well be improving the overall wirelength, at some as yet unknown cost in achieving timing optimization. Replacing Domino with a more suitable legalizer is clearly a topic for future work of us.

5. Conclusions

Rather than move the gates to optimize their location, a grid-warping placer elastically deforms a model of the 2-D chip surface on which the gates have been quickly and coarsely placed. In this paper, we introduced the first timing-driven grid-warping formulation, based on slack-sensitivity net weighting. Given inevitable concerns about wirelength and runtime degradation in any timing-driven scheme, we also incorporated a more efficient net model and an integrated local improvement ("rewarping") step. An implementation of these ideas, Warp2, can improve worst-case negative slack by 37% on average, with very modest increases in wirelength and runtime. Ongoing work is examining the potential in a backend placer customized to the warping process, and options for incorporating net-based congestion estimators and fixed pre-placement of macroblocks.

Acknowledgments

We thank Andreas Kuehlmann of Cadence for providing us the opportunity to access and integrate with Open Access; Christoph Albrecht and Philip Chong of Cadence for help with the development of OA Gear Timer and discussion about Open Access. This work is supported by 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 1997.

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

[12] C. Chang, J. Cong, M. Xie, "Optimality and scalability study of existing placement algorithms," *Proc. ASP-DAC*, 2003.

[13] N. Viswanathan, C. C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," *Proc. ACM ISPD*, April 2004.

[14] Z. Xiu, J. D. Ma, S. M. Fowler, R. A. Rutenbar, "Large-scale placement by grid-warping," *Proc. ACM/IEEE DAC*, June 2004.

[15] P. Villarrubia, "Important considerations for modern VLSI chips," *Proc. ACM ISPD*, April 2003.

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

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

[18] S. Ou, M. Pedram, "Timing-driven placement based on partitioning with dynamic cut-net control," *Proc. ACM/IEEE DAC*, June 2000.

[19] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, B. Halpin, "Timing driven force directed placement with physical net constraints," *Proc. ACM ISPD*, April 2003.

[20] H. Ren, D. Z. Pan, D. S. Kung, "Sensitivity guided net weighting for placement driven synthesis," *Proc. ACM ISPD*, April 2004.

[21] http://openeda.si2.org/.

[22] Z. Xiu, D. A. Papa, P. Chong, C. Albrecht, A. Kuehlmann, R. A. Rutenbar, I. L. Markov, "Early research experience with OpenAcess Gear: an open source development environment for physical design" *Proc. ACM ISPD*, April 2005.

[23] OA Gear homepage: http://openedatools.si2.org/oagear/.

Design	Wirelength-only Warp2/Domino			Timing-driven Warp2/Domino			WNS Improvement & Wirelength Increase		
	Wirelength	WNS	CPU Time (s)	Wirelength	WNS	CPU Time (s)	Target (ns)	WNS	Wirelength
S1423	22266.6	-0.18088	3.43	22896.9	-0.14390	5.75	4.33	20.4%	2.83%
S1488	23262.1	-0.07757	2.98	22979.2	-0.07080	3.78	1.75	8.7%	-1.22%
S1494	24142.8	-0.15442	1.99	25593.2	-0.14538	3.82	1.75	5.9%	6.01%
S5378	68706.5	-0.09576	7.72	70355.6	-0.04948	10.35	2.42	48.3%	2.40%
S9234	49630.7	-0.12586	9.47	50624.1	-0.10411	10.84	2.85	17.3%	2.00%
S13207	115918	-0.11002	39.95	112419	-0.07510	54.17	3.66	31.7%	-3.02%
S15850	194839	-0.24589	52.40	186849	-0.20061	83.87	4.75	18.4%	-4.10%
S35932	566025	-0.09083	590.84	570007	-0.00270	870.72	2.42	97.0%	0.70%
S38417	697415	-0.19155	422.72	748544	-0.15832	601.29	3.75	17.3%	7.33%
S38584	619260	-0.05733	315.59	606929	0.02596	467.7	3.75	100.0%	-1.99%
Ratio	**1.00**	**1.000**	**1.00**	**1.01**	**0.635**	**1.47**		**36.5%**	**1.09%**

TABLE 4. Placement results comparing Wirelength-only Warp2 against Timing-driven Warp2