# Gravity: Fast Placement for 3-D VLSI

STEFAN THOMAS OBENAUS
Montreal, Canada
and
TED H. SZYMANSKI
McMaster University

Three dimensional integration is an increasingly feasible method of implementing complex circuitry. For large circuits, which most benefit from 3-D designs, efficient placement algorithms with low time complexity are required.

We present an iterative 3-D placement algorithm that places circuit elements in three dimensions in linear time. Using an order of magnitude less time, our proposed algorithm produces placements with better than 11% less wire lengths than partitioning placement using the best and fastest partitioner. Due to the algorithms iterative nature, wire-length results can be further improved by increasing the number of iterations.

Further, we provide empirical evidence that large circuits benefit most from 3-D technology and that even a small number of layers can provide significant wire-length improvements.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*placement and routing*; J.6 [**Computer-Aided Engineering**]—*computer-aided design (CAD)*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Placement, 3-D VLSI, 3-D integrated circuits

## 1. INTRODUCTION

As integrated circuits become more complex, utilization of the third dimension is becoming a more realistic solution. Recent work has resulted in 3-D field programmable arrays (FPGAs) with a mesh-like distribution of programmable circuit elements [Leeser et al. 1998]. However, cell placement for 3-D integration is still in its infancy [Alexander et al. 1996; Leeser et al. 1998; Ohmura 1998; Tong and Wu 1995; Tanprasert 2000].

If 3-D integration is to help the implementation of very large circuits, efficient placement and routing tools are required. Therein lies the main differences

Authors' addresses: S. T. Obenaus, 605-3460 Peel Street, Montreal, Quebec, H3A 2M1 Canada; email: obenaus@obenaus.org; T. H. Szymanski, Department of Electrical and Computer Engineering, McMaster University, 1280 Main Street West, Hamilton, Ontario, Canada L8S 4K1; email: teds@ece.eng.mcmaster.ca.

between 2-D and 3-D placement algorithms. 3-D placement methods will have to place much larger circuits than 2-D placers, thus they need to have near linear run-time complexities.

In this paper, we will present a fast placement algorithm for three dimensional cell placement. Our algorithm runs in linear time, and produces better placements than 3-D partitioning placement using a leading partitioner. The quality of the placement is measured by the resulting total wire length. For future benchmarking, we generated 3-D placement results for two benchmark circuit suites, ACM/SIGDA and ISPD98, using our Gravity algorithm and partitioning placement method. These benchmarks are the first comprehensive wire-length benchmarks published for 3-D placements.

Previously mentioned three dimensional placement methods are recursive minimum-cut partitioning into octants [Alexander et al. 1996; Leeser et al. 1998], an analytical approach by Tanprasert [2000], and a placement algorithm due to Ohmura [1998]. Ohmura's algorithm has a much higher run-time complexity than our, namely $O(|V| \cdot |E|)$ where $|V|$ is the number of circuit nodes, and $|E|$ is the number of nets. Thus, Ohmura's algorithm is primarily useful for small circuits. The largest circuit Ohmura's algorithm was tested on had 64 nodes and 60 nets. Gravity, the algorithm we propose, was tested on circuits up to 210,000 nodes and 200,000 nets. The recursive minimum-cut partitioning algorithms by Alexander et al. and Leeser et al. rely on the strength of their minimum cut partitioner. Since recursive partitioning methods have proven strong in run time and placement quality in two dimensions [Shahookar and Mazumder 1991] we will compare our algorithm against 3-D recursive partitioning placement using the currently fastest and best minimum cut partitioner by Karypis et al. [1997].

## 2. RELATED METHODS

Force-directed and quadratic placement algorithms are popular for the 2-D placement problem. An in-depth overview of existing two-dimensional placement methods can be found in Shahookar and Mazumder [1991]. In principle, these 2-D algorithms can be extended to the 3-D domain with minor adjustment. However, 3-D placers will have to place considerably larger circuits, thereby increasing the importance of near linear run times.

Force-directed algorithms [Antreich et al. 1982; Chang and Hsiao 1993; Eisenmann and Johannes 1998; Goto 1981; Koford 1998; Tia and Liu 1993; Ueda et al. 1985] typically base their placements on a set of contractive and repulsive forces that draw connected circuit elements closer together but repulse overlapping nodes. An equilibrium of these forces is computed either by solving sets of linear equations, or by iteration. The equilibrium positions may be post-processed to remove cell overlaps to yield the final cell positions. Our proposed algorithm Gravity uses attractive forces to bring together connected nodes iteratively. However, Gravity does not use repulsive forces to remove cell overlap. Gravity relies on a novel bucket rescaling technique that reasserts an even cell distribution periodically with low computational overhead.

Quadratic placement methods pioneered by Wipfler et al. [Kleinhans et al. 1991; Parakh et al. 1998; Tsay and Kuh 1991; Tsay et al. 1988; Vygen 1997; Wipfler et al. 1982], use a force-directed algorithm to compute the initial positions of cells. These initial positions are used to seed the partitions of a minimum-cut partitioner, which then recursively partitions the chip area and the cells while minimizing the number of nets cut. The minimum-cut partitioners eliminate cell overlap, albeit at considerable computational expense in addition to the force-directed step. Gravity similarly ignores cell overlap during its force-directed step. However, in Gravity the final placement is determined by recursively partitioning the cells based upon their computed positions, with low computational overhead.

## 3. THE MODEL

Our 3-D circuit placement model aims to reflect the most general scientific definition of the wire-length placement model, while observing engineering constraints. The result is a 3-D lattice into which circuit elements are placed such that the cumulative length of rectilinear Steiner trees connecting the nodes of each net is approximately minimized.

We define a rectilinear Steiner tree:

*Definition* 1. Rectilinear Steiner Tree

A rectilinear Steiner tree $S(V, f)$ is the shortest tree that connects all nodes $v \in V$ at positions $f(v)$ using only orthogonal horizontal and vertical segments. Its length is $|S(V, f)|$.

### 3.1 2-D Model

The purest and widely accepted placement model for conventional 2-D VLSI placement models the chip surface as a square grid with all circuit elements being of unit square size and having their I/O connections in the center. Circuit elements are then placed in checkerboard-fashion onto the grid [Shahookar and Mazumder 1991].

A circuit to be placed is abstracted into a hypergraph $G(V, E)$. The circuit elements form the set $V$ of nodes of the hypergraph, and the circuit nets form the set of hyperedges $E$ that are subsets of $V$.

The goodness of the placement is measured by its total wire length. The wire length is measured for each net. This is the length of the shortest rectilinear tree that connects all nodes in the net. Such a tree is called a rectilinear Steiner tree [Hanan 1966]. Determining the lengths of a rectilinear Steiner tree is difficult. A popular method for estimating the wire length is the semi-perimeter bounding box approximation [Shahookar and Mazumder 1991]. This estimate simply adds the horizontal and vertical distance spanned by the nodes in the net. This estimate is exact for two- and three-node nets, which typically form the majority of all nets.

Consequently, we define the placement problem formally:

*Definition* 2. 2-D Placement Problem

Given a hypergraph $G(V, E)$, find a reversible function $f : V \to \{1, \ldots, n_1\} \times \{1, \ldots, n_2\}$, such that $\sum_{e \in E} |S(e, f)|$ is minimized.

In the case where $G(V, E)$ is a graph, that is, all nets have two nodes, and when $n_2 = 1$, this problem degenerates into the well-known problem *Optimal Linear Arrangement* [Garey and Johnson 1979]. Optimal Linear Arrangement is NP-complete, and hence the 2-D placement problem is also intractable. This intractability means that finding an optimal solution to the placement problem in an acceptable amount of time is generally not possible. Hereafter, we will consider only algorithms that approximately solve the placement problem, that is, reduce the total wire length as much as possible.

### 3.2 3-D Model

The three dimensional model is a natural extension of the 2-D model:

*Definition* 3. 3-D Placement Problem
Given a hypergraph $G(V, E)$, find a reversible function $f : V \rightarrow \{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \times \{1, \ldots, n_3\}$ such that $\sum_{e \in E} |S(e, f)|$ is minimized.

This definition also describes the model used by Ohmura [1998].

The length of orthogonal Steiner trees representing the minimum net lengths are now approximated by the sum of the height, width, and depth of a Steiner tree.

The reader will have noticed that we treat the vertical dimension like the horizontal dimensions to create a homogeneous three-dimensional mesh. Even though connections in the vertical dimension often correspond to *vias* connecting elements in different layers [Chiricescu and Vai 1998; Depreitere et al. 1994; Leeser et al. 1998; Leighton and Rosenberg 1986; Ohmura 1998; Reber and Tielert 1986; Tong and Wu 1995], Leeser et al. [1998] have shown that vias need not be more expensive in terms of connection delay than intra layer connections. Thus, it is reasonable to focus on the underlying fundamental problem of finding a mapping of nodes into a uniform three-dimensional grid.

We should also point out that problem Definition 3 allows for specification of the number of layers $n_3$. Consequently, this paper is not restricted to placements into three-dimensional cubes but rather arbitrary 3-D grids. In Section 5.1, we examine the effect of adding more layers to a three dimensional placement mesh.

## 4. ALGORITHM

The Gravity placement algorithm has four simple stages. The first stage is a random placement of nodes into the unit cube. This is followed by a force-based iteration that moves neighboring nodes closer together. After a number of force-based iterations, node positions are rescaled in stage three to reachieve an approximate uniform node distribution. After a number of repetitions of stages two and three, stage four determines the final placement through a recursive partitioning phase based on the nodes' computed coordinates.

In this section, we will explain each stage, and provide some performance remarks. Figure 1 gives an overview of the Gravity algorithm.

| Input: | | | Default |
|---|---|---|---|
| | $G(V, E)$ | hypergraph | |
| | $(m_1, m_2, m_3)$ | gridsize | |
| | $I$ | number of iterations | |
| | $K$ | number of final placements | 25 |
| | $r$ | rescaling interval | 10 |

| Step | Action | Section |
|---|---|---|
| 1: | Perform random initial placement. | (4.1) |
| 2: | For $i = 1$ to $I$ | |
| 3: | Compute force based iteration. | (4.2) |
| 4: | if $i \bmod \lfloor I/K \rfloor = 0$ | |
| 5: | Compute a final placement. | (4.4) |
| 6: | if $i \bmod r = 0$ | |
| 7: | Perform bucket rescaling. | (4.3) |

Output best computed final placement.

Fig. 1. Gravity placement algorithm.

### 4.1 Random Initial Placement

Initially all circuit nodes are assigned a random initial position with a uniform distribution over the unit cube. Nets, that is, hyperedges, connecting the nodes are ignored. Circuit elements may overlap. In fact, we will tolerate node overlap until the final placement step (see Section 4.4). This is one of the most distinctive features of our algorithm and allows for very fast iterations.

### 4.2 Force-Based Iteration

In the force-based iteration, each node $n$'s new position $x'_n = (x'_{n1}, x'_{n2}, x'_{n3})$ is the weighted average of its own position, and the positions of its neighbors.

$$x'_n = \frac{x_n + \sum_{e \in E_n} w_e \left[ \left( \sum_{n' \in e} x_{n'} \right) - x_n \right]}{1 + \sum_{e \in E_n} w_e \big( |e| - 1 \big)} \tag{1}$$

where

$$E_n = \text{set of hyperedges incident to node } n,$$
$$w_e = \frac{2}{|e|(|e|-1)} = \text{weight of edge } e.$$

The formula for the edge weight $w_e$ reflects the fact that each edge enters the position calculations $\binom{|e|}{2}$ times when it should only be counted once. It should be observed, that the cardinalities and weights of all edges have to be computed only once as they are constants. Further, the position sums $\sum_{n' \in e} x_{n'}$ have to be computed only once for each edge $e$ at each iteration. Thus this iteration step's execution time is linear in the number of pins $p = \sum_e |e|$.

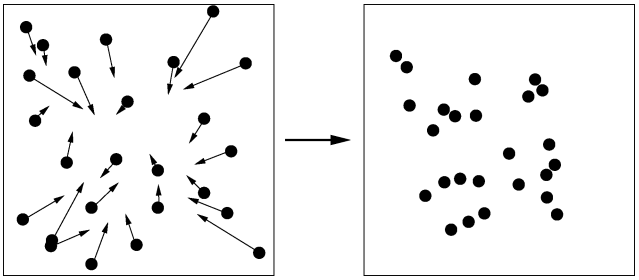

Fig. 2. Step 2: Gravitate nodes. Projection onto 2-D.

Equation (1) utilizes the position sum of all nodes in a net, every time a node coordinate is updated. By using the node positions from the last iteration, this sum is constant and can be precomputed for all nets before every iteration, resulting in a significant computational savings. An alternative would be to use the latest node positions to compute the position sum in Equation (1). This approach would require significant added computation every time a node coordinate is updated, since the position of every node on every adjacent net must be examined, and some nets can be very large. Further, the positions of nodes tend to converge after many iterations, so the benefits of using the latest coordinates over the last iteration's coordinates diminish. Empirical tests confirm that using new values does not lead to a faster convergence and increases run-times.

In contrast to most previous methods, we allow nodes to move freely, even if nodes overlap and occupy the same volume. We call this a force based method because an attractive force between neighbors exerts a pull on each node, thus reducing the total wire length needed to embed the circuit. The overall effect is that all nodes are slowly pulled to the center of the chip volume. See Figure 2 for an illustration. In the next step, we counter the pull toward the center through uniform rescaling.

### 4.3 Bucket Rescaling

After a few iterations, it becomes necessary to counter the nodes' tendency to cluster near the center. Two problems arise if nodes are allowed to cluster unchecked. For one, the resolution of the nodes' positions is limited by the precision of the variables they are stored in. A 24-bit variable, as used in the current implementation, can store positions with a resolution of $2^{-24}$. If nodes are separated by less than $2^{-24}$, their positions become indistinguishable. At this point, force step computations according to Equation (1) become ineffective. The second reason why rescaling of node positions becomes necessary is rooted in the problem definition itself. The problem definition calls for a placement of circuit nodes into a 3-D mesh. In a mesh, nodes are separated by an even spacing. However, heavily clustered nodes are far from being equally spaced. This changes the nature of the optimization process and results in poor quality placements after the final placement step.

Gravity exploits a novel bucket rescaling technique to achieve a uniform cell distribution. After every $r$ iterations, the unit cube is sliced into a grid of

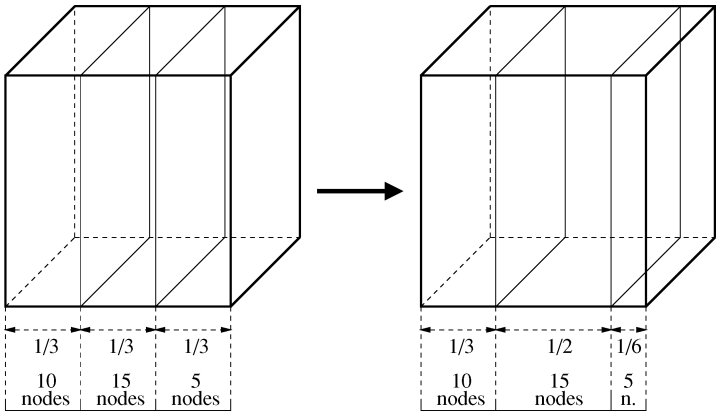


Fig. 3. Step 3: Re-scaling of node distribution.

$m_1 \times m_2 \times m_3$ buckets for rescaling. The nodes in each bucket are counted. Then, the slice widths are resized proportional to their node counts (see Figure 3). The same rescaling process is repeated for each column in each slice, and for each bucket within each column. This entire rescaling step is repeated until all bucket counts are with 20% of the mean $N/(m_1 m_2 m_3)$. The number of iterations between rescalings $r$ and the number of buckets $M = m_1 \times m_2 \times m_3$ are correlated parameters. The longer the algorithm runs before rescaling, the greater the tendency of nodes to cluster near the center. To rebalance these nodes efficiently, the number of buckets M should be chosen to minimize the variance of the number of nodes per bucket after rescaling. We chose to fix the parameter $r = 10$, which we observed empirically to yield good results, and adjust $M$ according to the theoretical analysis of Equations (2) and (3) in the following section.

4.3.1 *Number of Buckets.* The number of buckets used for rescaling affects the homogeneity of the rescaled node distribution and also the variance in individual bucket counts. On one hand, we wish to choose as many buckets as possible in order to achieve high homogeneity of node positions since this would closely resemble a final node placement. On the other hand, a larger number of buckets leads to a higher variance in individual bucket counts. This can result in an excessive number of repetitions of the rescaling step as bucket counts are increasingly likely to fall outside 20% of the mean.

If we model the rescaled node distribution over the cube by a uniform node distribution, we can arrive at a formula to estimate the optimal number of buckets. For a uniform node distribution, the bucket counts are governed by a binomial distribution. We use a normal distribution to approximate the binomial distribution in order to determine the probability $P_\varepsilon$ that all $M$ bucket counts are within a factor of $\varepsilon$ of the mean:

$$P_\varepsilon = \left[ \operatorname{erf}\left( \frac{\varepsilon N/M + 1/2}{\sqrt{2}\sqrt{N/M(1 - 1/M)}} \right) \right]^M . \tag{2}$$

Since we expect the node distribution to be symmetrical, we choose an odd number of buckets $m_i$ in each dimension $i$ for a total number of $M = m_1 \times m_2 \times m_3$ buckets. Starting with a $3 \times 3 \times 3$ grid of buckets, we increase each $m_i$ in turn by 2 as long as $P_\varepsilon \geq 50\%$. This ensures that the expected number of rescaling iterations is no more than 2. The actual node distribution after a rescaling step outlined in section 4.3 is not a true uniform distribution. In fact, the resulting distribution exhibits a smaller variance in bucket counts as this is the aim of our rescaling technique. By empirical observation, we determined that $\varepsilon$ should be scaled by 1.85. Thus, the actual number of buckets used by Gravity, $M'$ is

$$\begin{aligned} M' = \max\{M : & P_{1.85\varepsilon} \geq 0.5 \\ & \text{and } M = m_1 m_2 m_3 \\ & \text{and } m_1, m_2, m_3 \text{ odd} \\ & \text{and } \max\{m_1, m_2, m_3\} \\ & - \min\{m_1, m_2, m_3\} \leq 2\}. \end{aligned} \tag{3}$$

Observation has shown that choosing the initial number of buckets according to (3) leads to very good placement results for all benchmark circuits ranging from 833 nodes for `p1` to 210,613 nodes for `ibm18`. Occasionally, degenerate circuit features can still lead to excessive rescaling iterations. In the rare case where more than 12 rescaling iterations occur, we reduce $m_1$, $m_2$, or $m_3$ in turn by 2 and jiggle the node positions randomly by a small amount between $\pm 1/(2\sqrt{N})$.

After a preset number of iterations of stages two and three, Gravity performs a final placement step in which all node positions are adjusted to remove overlap.

### 4.4 Final Placement

After a number of iterations of the force based iteration (see Section 4.2), and the rescaling bucket mapping step (see Section 4.3), some cells are expected to overlap partially or completely. In a final placement such an overlap is not tolerated. Hence, this step will assign a unique grid position to each cell.

We describe a final placement method based on a recursive grid-splitting technique. This method allows nodes to be placed into arbitrary 3-D grids in compliance with problem Definition 3.

Our grid-splitting method recursively splits an arbitrarily chosen $n_1 \times n_2 \times n_3$ grid along the largest dimension. At the same time, the nodes are recursively split according to their position along the same dimension. This splitting process is continued until each node $u$ is assigned to a unique position $f_V(u) \in \{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \times \{1, \ldots n_3\}$. For illustration purposes, Figure 4 shows a 2-D version of this recursive grid-splitting procedure for a 25-node circuit and a $5 \times 5$ grid.

The size of the grid dimensions $n_1$, $n_2$, and $n_3$ can be arbitrarily set, but normally they are chosen such that the grid closely resembles a cube with approximately $N$ nodes. The exact grid dimensions for an $N$ node circuit are

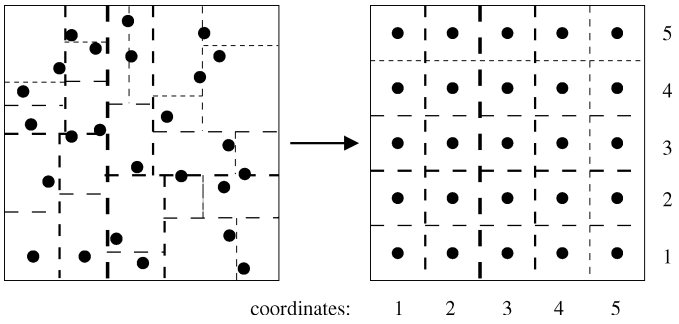


Fig. 4. Step 4: Recursive grid splitting (left) leads to final placement of nodes (right).

given by

$$n_1 = \lceil \sqrt[3]{N} \rceil \tag{4}$$

$$n_2 = \lceil \sqrt{N/n_1} \rceil \tag{5}$$

$$n_3 = \left\lceil \frac{N}{n_1 n_2} \right\rceil . \tag{6}$$

Typically, we compute a final placement at regular intervals 25 times during a run. This lets us sample the solution in an effort to eliminate noise and to avoid potential local maxima.

### 4.5 Performance Remarks

The overall time complexity of Gravity is $\Theta(p)$, provided that, for an $N$ node circuit and $I$ force step iterations, $N \ll 2^I$, and the space requirement is the size of the input, i.e., $\Theta(p)$.

The run time of Gravity is reduced by simplifying the data structures, simplifying the arithmetic, and improving cache hit rates. The data structures accessed in the force step are stripped to the minimum information necessary. Thus more nodes and edges can be kept in cache and fewer pointer dereferences have to be performed. Secondly, we use integer operations to simulate fixed point arithmetic. Finally, we arrange the node and edge data arrays according to the order of a depth first search before we start the iterations. This helps keeping more nodes and edges in L1 cache.

## 5. RESULTS

We will now examine the results of our 3-D placement algorithm. Of particular interest are two aspects. On one hand, we would like to know how three dimensional placement improves the wire length over two dimensional placement. This aspect is examined in Section 5.1. On the other hand, we are interested to see how 3-D Gravity compares to other near-linear time placement algorithms. For this reason we compare, in Section 5.2, Gravity's placements to placement by recursive partitioning using one of the leading partitioners.

In both cases, we use the 1993 ACM/SIGDA [Brglez 1993] and the 1998 ISPD [Alpert 1998] benchmark suites (see Table I).

Table I. The 1993 ACM/SIGDA (top) and 1998 ISPD Benchmark Circuits (bottom)

| Circuit | Nodes | Nets | Pins | $\frac{Pins}{Node}$ | $\frac{Pins}{Net}$ |
|---|---|---|---|---|---|
| 19ks | 2844 | 3282 | 10547 | 3.71 | 3.21 |
| avq.large | 25178 | 25384 | 82751 | 3.29 | 3.26 |
| avq.small | 21918 | 22124 | 76231 | 3.48 | 3.45 |
| baluP | 801 | 735 | 2697 | 3.37 | 3.67 |
| biomedP | 6514 | 5742 | 21040 | 3.23 | 3.66 |
| golem3 | 103048 | 144949 | 338419 | 3.28 | 2.33 |
| industry2 | 12637 | 13419 | 48158 | 3.81 | 3.59 |
| industry3 | 15406 | 21923 | 65791 | 4.27 | 3.00 |
| p1 | 833 | 902 | 2908 | 3.49 | 3.22 |
| p2 | 3014 | 3029 | 11219 | 3.72 | 3.70 |
| s13207P | 8772 | 8651 | 20606 | 2.35 | 2.38 |
| s15850P | 10470 | 10383 | 24712 | 2.36 | 2.38 |
| s35932 | 18148 | 17828 | 48145 | 2.65 | 2.70 |
| s38417 | 23949 | 23843 | 57613 | 2.41 | 2.42 |
| s38584 | 20995 | 20717 | 55203 | 2.63 | 2.66 |
| s9234P | 5866 | 5844 | 14065 | 2.40 | 2.41 |
| structP | 1952 | 1920 | 5471 | 2.80 | 2.85 |
| t2 | 1663 | 1720 | 6134 | 3.69 | 3.57 |
| t3 | 1607 | 1618 | 5807 | 3.61 | 3.59 |
| t4 | 1515 | 1658 | 5975 | 3.94 | 3.60 |
| t5 | 2595 | 2750 | 10076 | 3.88 | 3.66 |
| t6 | 1752 | 1641 | 6638 | 3.79 | 4.05 |
| ibm01 | 12752 | 14111 | 50566 | 3.97 | 3.58 |
| ibm02 | 19601 | 19584 | 81199 | 4.14 | 4.15 |
| ibm03 | 23136 | 27401 | 93573 | 4.04 | 3.41 |
| ibm04 | 27507 | 31970 | 105859 | 3.85 | 3.31 |
| ibm05 | 29347 | 28446 | 126308 | 4.30 | 4.44 |
| ibm06 | 32498 | 34826 | 128182 | 3.94 | 3.68 |
| ibm07 | 45926 | 48117 | 175639 | 3.82 | 3.65 |
| ibm08 | 51309 | 50513 | 204890 | 3.99 | 4.06 |
| ibm09 | 53395 | 60902 | 222088 | 4.16 | 3.65 |
| ibm10 | 69429 | 75196 | 297567 | 4.29 | 3.96 |
| ibm11 | 70558 | 81454 | 280786 | 3.98 | 3.45 |
| ibm12 | 71076 | 77240 | 317760 | 4.47 | 4.11 |
| ibm13 | 84199 | 99666 | 357075 | 4.24 | 3.58 |
| ibm14 | 147605 | 152772 | 546816 | 3.70 | 3.58 |
| ibm15 | 161570 | 186608 | 715823 | 4.43 | 3.84 |
| ibm16 | 183484 | 190048 | 778823 | 4.24 | 4.10 |
| ibm17 | 185495 | 189581 | 860036 | 4.64 | 4.54 |
| ibm18 | 210613 | 201920 | 819697 | 3.89 | 4.06 |

### 5.1 From Two to Three Dimensions

One of the anticipated advantages of 3-D circuitry is that total circuit wire length is expected to be shorter than in two dimensions. We are substantiating this expectation by providing placement results in $d = 2$, 2 1/3, 2 2/3, and 3 dimensions for a sample of benchmark circuits. Table II shows the results for four circuits covering the dynamic range from 800 to 210,000 nodes. The grid sizes were computed to most closely resemble a $\sqrt[d]{N} \times \sqrt[d]{N} \times N/(\sqrt[d]{N})^2$ grid for an $N$-node circuit in $d$ dimensions. Since the semi-perimeter bounding box

Table II. Wire-Length Improvement from Two to Three Dimensions

| | | | | | All Nets Counted | | 2- and 3-node Nets | |
|---|---|---|---|---|---|---|---|---|
| Circuit | Nodes | | Grid | | Length (approx.) | Change (approx.) | Length (exact) | Change (exact) |
| p1 | 833 | 29 | 29 | 1 | 5027.6 | | 2778.9 | |
| | | 17 | 17 | 3 | 3751.3 | −25.38% | 2202.3 | −20.74% |
| | | 12 | 12 | 6 | 3418.3 | −32.00% | 2032.3 | −26.86% |
| | | 10 | 10 | 9 | 3385.9 | −32.65% | 2021.6 | −27.25% |
| s9234P | 5866 | 77 | 77 | 1 | 27055.6 | | 18728.4 | |
| | | 39 | 38 | 4 | 18749.9 | −30.69% | 13745.2 | −26.60% |
| | | 26 | 26 | 9 | 16573.6 | −38.74% | 12570.3 | −32.88% |
| | | 19 | 18 | 18 | 16041.9 | −40.70% | 12241.2 | −34.63% |
| ibm06 | 32498 | 181 | 180 | 1 | 712282.8 | | 233084.3 | |
| | | 81 | 81 | 5 | 365606.2 | −48.67% | 126683.4 | −45.64% |
| | | 49 | 48 | 14 | 277323.5 | −61.06% | 101553.6 | −56.43% |
| | | 32 | 32 | 32 | 254276.8 | −64.30% | 95390.0 | −59.07% |
| ibm18 | 210613 | 459 | 459 | 1 | 7128339.7 | | 2179268.3 | |
| | | 188 | 187 | 6 | 3140755.9 | −55.93% | 1011482.3 | −53.58% |
| | | 98 | 98 | 22 | 2111055.3 | −70.38% | 728096.2 | −66.58% |
| | | 60 | 60 | 59 | 1875854.6 | −73.68% | 655591.2 | −69.91% |

approximation method underestimates the actual wire length for nets of 4 or more nodes, we also included the exact total wire length for the subset of nets containing only 2- and 3-node nets into Table II. For 2- and 3-node nets, the semi-perimeter method is exact. Therefore, the wire length reductions shown in the last column of Table II are exact. The entries are based on 10 runs of Gravity. Even though Gravity is a randomized algorithm, due to its random initial placement, its results are very stable. The average standard deviation of each run from the mean is less than 2%.

This table shows that the wire length advantage of 3-D placements over 2-D increases substantially from 27% to 70% as the circuit size increases from 833 to 210,613 nodes. The results also show that even a small number of additional layers results in significant wire length savings for larger circuits, for example 2 1/3 dimensions for ibm06 and ibm18.

### 5.2 Gravity vs. 3-D Partitioning Placement

Since no 3-D placement results have been published before, we needed to create a 3-D placement comparison basis on a fundamentally proven and strong technique. Partitioning placement is one of the basic and proven placement schemes in two dimensions. Based on the simplicity of the partitioning placement method, and recent advances in partitioning algorithms, it is natural to extend partitioning placement to three dimensions for comparison purposes. In the following section we describe the 3-D partitioning placement algorithm that we used to present the first 3-D placement results for benchmark circuits at VLSI'99 [Obenaus and Szymanski 1999]. In Section 5.4 we compare the results of this 3-D partitioning placer with the 3-D results produced by Gravity.

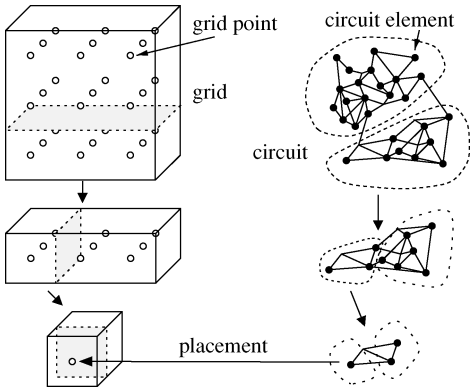


Fig. 5. Partition placement process: simultaneous splitting of the grid and partitioning of the circuit.

### 5.3 Recursive Partitioning Placement using Grid Splitting

As mentioned above, partitioning placement is one of the fundamental placement methodologies. For our 3-D recursive partitioning placer, we implement a grid splitting technique that mirrors 3-D Gravity's final placement step with the important difference that here the node partitioning has to be computed using a partitioner, whereas in Gravity the splitting of nodes is based on its computed coordinates. Figure 5 illustrates how this algorithm proceeds, and Figure 6 provides the pseudo code of this partition placement algorithm.

Partitioning placement in three dimensions has been suggested before. Leeser et al. [1998] used a partitioning placement method for placement in the Rothko architecture, and Alexander et al. [1996] suggested it for three dimensional FPGA placement. Their partition placement methods were based on a 2-D variation of partitioning placement, called *quadrisection* [Shahookar and Mazumder 1991]. In quadrisection, the chip area is recursively split into four quadrants and circuits are recursively partitioned four ways. They extended this method into three dimensions by splitting the chip's volume into eight octants while concurrently partitioning the circuit eight ways. However, no large circuit placement results were published.

As our partitioner, we chose the hMetis hypergraph partitioner developed by Karypis et al. [1997]. To our knowledge, this partitioner is currently the best of the published near-linear-run-time partitioners. Although we use recursive two-way partitioning along each axis in 3-D, we could easily implement 3-D quadrisection with a few modifications to the hMetis library interface. Restrictions in the current hMetis library interface made it necessary to compute a recursive, balanced $(k + l)$-way partitioning to achieve a $k{:}l$ split as is sometimes necessary in step 12 of the algorithm in Figure 6 when an odd number of rows, columns, or layers needs to be split. While recursive multi-way partitioning increases run-time and memory requirement, it does not affect the quality of the cut (Karypis, personal communication). According to Karypis, the hMetis interface could easily be adapted to allow explicit $k{:}l$ cuts.

Variables and predicates:

| | |
|---|---|
| $V = \{v_1, \ldots, v_{\|V\|}\}$ | set of circuit nodes |
| $x[v]$ | coordinates of gate for circuit node $v$ |
| $(a_1, a_2, a_3)$ | coordinates of lower left front corner |
| $(b_1, b_2, b_3)$ | lengths of sides of gate array box |
| $(n_1, n_2, n_3)$ | initial size of gate array box |

Initial Call:

`place`($V$,(0, 0, 0),($n_1, n_2, n_3$))

```
place(V,(a1, a2, a3),(b1, b2, b3))
1:      if |V|=1 then
2:          x[v1] := (a1, a2, a3)
3:      else
            find largest side of box
4:          k := i such that bi = max(b1, b2, b3)
            split box b into two boxes b1 and b2
5:          (b11, b12, b13) := (b1, b2, b3)
6:          b1k := ⌊bk/2⌋
7:          (b21, b22, b23) := (b1, b2, b3)
8:          b2k := ⌈bk/2⌉
            determine coordinates of lower left
            front corner of b1 and b2
9:          (a11, a12, a13) := (a1, a2, a3)
10:         (a21, a22, a23) := (a1, a2, a3)
11:         a2k := ak + b1k
            partition V into subcircuits V1 and V2
            of sizes no more than b11·b12·b13
            and b21·b22·b23, respectively
12:         (V1, V2) := partition(V,b11·b12·b13,b21·b22·b23)
            invoke placement routine on subcircuits
13:         place(V1,(a11, a12, a13),(b11, b12, b13))
14:         place(V2,(a21, a22, a23),(b21, b22, b23))
```

Fig. 6. Generic partitioning placement algorithm.

In order to compensate for the excessive memory requirement for large $(k + l)$-way partitionings, we restricted the largest dimensions of the grid for the largest circuits to be of even length. Consequently, the largest cuts are balanced two-way cuts which require substantially less memory resources. Further, to obtain accurate estimates of the run time, assuming the hMetis interface was adapted to allow explicit $k$:$l$ splits, we ran the algorithm while forcing balanced splits at all levels of recursion down to 8 or less nodes when no more partitioning intelligence is required.

### 5.4 Result Comparison

We compare the placement results of our 3-D Gravity algorithm against the performance of the 3-D partitioning placer described above (see Section 5.3).

Table III. 3-D Placement Grids for the ACM/SIGDA Suite (top) and the ISPD98 Suite (bottom)

| Circuit | Nodes | 3-D Gravity Grid | 3-D hMetis Grid |
|---|---|---|---|
| 19ks | 2844 | $15 \times 14 \times 14$ | $15 \times 14 \times 14$ |
| avq.large | 25178 | $30 \times 29 \times 29$ | $30 \times 30 \times 28$ |
| avq.small | 21918 | $28 \times 28 \times 28$ | $28 \times 28 \times 28$ |
| baluP | 801 | $10 \times 9 \times 9$ | $10 \times 9 \times 9$ |
| biomedP | 6514 | $19 \times 19 \times 19$ | $19 \times 19 \times 19$ |
| golem3 | 103048 | $47 \times 47 \times 47$ | $48 \times 48 \times 45$ |
| industry2 | 12637 | $24 \times 23 \times 23$ | $24 \times 23 \times 23$ |
| industry3 | 15406 | $25 \times 25 \times 25$ | $26 \times 25 \times 24$ |
| p1 | 833 | $10 \times 10 \times 9$ | $10 \times 10 \times 9$ |
| p2 | 3014 | $15 \times 15 \times 14$ | $15 \times 15 \times 14$ |
| s13207P | 8772 | $21 \times 21 \times 20$ | $21 \times 21 \times 20$ |
| s15850P | 10470 | $22 \times 22 \times 22$ | $22 \times 22 \times 22$ |
| s35932 | 18148 | $27 \times 26 \times 26$ | $27 \times 26 \times 26$ |
| s38417 | 23949 | $29 \times 29 \times 29$ | $29 \times 29 \times 29$ |
| s38584 | 20995 | $28 \times 28 \times 27$ | $28 \times 28 \times 27$ |
| s9234P | 5866 | $19 \times 18 \times 18$ | $19 \times 18 \times 18$ |
| structP | 1952 | $13 \times 13 \times 12$ | $13 \times 13 \times 12$ |
| t2 | 1663 | $12 \times 12 \times 12$ | $12 \times 12 \times 12$ |
| t3 | 1607 | $12 \times 12 \times 12$ | $12 \times 12 \times 12$ |
| t4 | 1515 | $12 \times 12 \times 11$ | $12 \times 12 \times 11$ |
| t5 | 2595 | $14 \times 14 \times 14$ | $14 \times 14 \times 14$ |
| t6 | 1752 | $13 \times 12 \times 12$ | $13 \times 12 \times 12$ |
| ibm01 | 12752 | $24 \times 24 \times 23$ | $24 \times 24 \times 23$ |
| ibm02 | 19601 | $27 \times 27 \times 27$ | $28 \times 28 \times 26$ |
| ibm03 | 23136 | $29 \times 29 \times 28$ | $30 \times 30 \times 27$ |
| ibm04 | 27507 | $31 \times 30 \times 30$ | $32 \times 30 \times 29$ |
| ibm05 | 29347 | $31 \times 31 \times 31$ | $32 \times 32 \times 30$ |
| ibm06 | 32498 | $32 \times 32 \times 32$ | $32 \times 32 \times 32$ |
| ibm07 | 45926 | $36 \times 36 \times 36$ | $36 \times 36 \times 36$ |
| ibm08 | 51309 | $38 \times 37 \times 37$ | $38 \times 38 \times 36$ |
| ibm09 | 53395 | $38 \times 38 \times 37$ | $38 \times 38 \times 37$ |
| ibm10 | 69429 | $42 \times 41 \times 41$ | $42 \times 41 \times 41$ |
| ibm11 | 70558 | $42 \times 41 \times 41$ | $42 \times 41 \times 41$ |
| ibm12 | 71076 | $42 \times 42 \times 41$ | $42 \times 42 \times 41$ |
| ibm13 | 84199 | $44 \times 44 \times 44$ | $44 \times 44 \times 44$ |
| ibm14 | 147605 | $53 \times 53 \times 53$ | $54 \times 54 \times 52$ |
| ibm15 | 161570 | $55 \times 55 \times 54$ | $56 \times 56 \times 54$ |
| ibm16 | 183484 | $57 \times 57 \times 57$ | $58 \times 58 \times 57$ |
| ibm17 | 185495 | $58 \times 57 \times 57$ | $58 \times 58 \times 57$ |
| ibm18 | 210613 | $60 \times 60 \times 59$ | $60 \times 60 \times 59$ |

As a comparison basis we used the ACM/SIGDA and ISPD98 benchmark circuit suites. For each benchmark circuit with $N$ nodes, both algorithms computed a placement into a homogeneous cube-like three-dimensional grid with a cube-edge length of approximately $\sqrt[3]{N}$ nodes. The exact 3-D grid dimensions are governed by Equations (4)–(6), subject to the constraints described in Section 5.3 above. Table III shows the exact grid dimensions. The cumulative wire lengths were estimated using an extension to three dimensions of the semi-perimeter bounding box method. This 3-D extension adds the height, width, and length of the volume spanned by the nodes in a net. This estimate

Table IV. Overview of 3-D Gravity versus 3-D Partitioning Placement for the ACM/SIGDA Suite

| | hMetis | | 250 iterations | | 500 iterations | | 1000 iterations | | 2000 iterations | |
|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | length | time (s) | change (%) | speed-up | change (%) | speed-up | change (%) | speed-up | change (%) | speed-up |
| 19ks | 14,493.3 | 37.41 | −19.87 | 13.87 | −20.95 | 8.61 | −21.58 | 4.69 | −22.50 | 2.49 |
| avq.large | 104,104.1 | 302.81 | −14.54 | 10.00 | −20.54 | 5.83 | −23.91 | 2.96 | −25.68 | 1.51 |
| avq.small | 94,688.0 | 277.32 | −15.68 | 10.52 | −20.23 | 6.18 | −23.70 | 3.17 | −24.95 | 1.65 |
| baluP | 3,263.5 | 13.34 | −16.05 | 14.79 | −16.02 | 8.69 | −16.70 | 4.49 | −16.41 | 2.34 |
| biomedP | 25,239.2 | 106.47 | −13.08 | 15.74 | −14.64 | 9.65 | −15.57 | 5.02 | −16.01 | 2.66 |
| golem3 | 687,104.9 | 1,519.41 | −3.58 | 10.97 | −12.25 | 7.24 | −16.58 | 3.96 | −18.89 | 2.07 |
| industry2 | 78,997.7 | 201.72 | −7.04 | 11.22 | −7.04 | 6.92 | −6.83 | 3.79 | −6.15 | 2.00 |
| industry3 | 152,962.3 | 300.69 | −18.36 | 13.20 | −19.13 | 7.75 | −19.17 | 4.26 | −19.21 | 2.27 |
| p1 | 4,156.1 | 14.11 | −17.70 | 15.93 | −17.54 | 9.43 | −18.09 | 5.07 | −19.22 | 2.63 |
| p2 | 18,562.5 | 43.16 | −15.50 | 13.00 | −16.42 | 7.70 | −15.98 | 4.12 | −15.69 | 2.13 |
| s13207P | 26,501.3 | 103.16 | −13.32 | 12.17 | −15.94 | 7.34 | −17.31 | 3.76 | −17.20 | 1.93 |
| s15850P | 30,950.7 | 121.61 | −8.92 | 11.30 | −12.16 | 6.59 | −13.49 | 3.36 | −12.93 | 1.84 |
| s35932 | 57,926.1 | 218.19 | −4.92 | 11.11 | −10.42 | 6.46 | −14.51 | 3.26 | −15.67 | 1.70 |
| s38417 | 73,282.6 | 252.72 | 2.44 | 8.55 | −2.77 | 4.78 | −4.10 | 2.45 | −3.43 | 1.29 |
| s38584 | 72,643.9 | 258.05 | −2.10 | 9.78 | −4.76 | 5.46 | −5.43 | 2.92 | −5.26 | 1.56 |
| s9234P | 17,670.1 | 85.74 | −6.88 | 16.57 | −8.46 | 9.99 | −9.41 | 5.49 | −9.19 | 2.82 |
| structP | 7,064.1 | 22.93 | −13.33 | 13.31 | −15.09 | 8.24 | −15.99 | 4.46 | −16.48 | 2.31 |
| t2 | 8,501.9 | 22.21 | −19.47 | 13.71 | −20.78 | 8.24 | −21.49 | 4.44 | −21.08 | 2.23 |
| t3 | 7,828.2 | 22.00 | −14.65 | 13.09 | −14.80 | 7.88 | −14.54 | 4.21 | −14.80 | 2.15 |
| t4 | 7,375.9 | 21.88 | −11.89 | 12.68 | −12.20 | 7.61 | −13.06 | 4.04 | −13.49 | 2.06 |
| t5 | 12,568.2 | 36.24 | −12.35 | 11.99 | −11.69 | 7.30 | −11.20 | 3.99 | −10.79 | 2.05 |
| t6 | 7,968.1 | 22.52 | −14.27 | 11.66 | −14.20 | 6.71 | −14.72 | 3.45 | −14.52 | 1.79 |
| Average | | | −11.87 | 12.51 | −14.00 | 7.48 | −15.15 | 3.97 | −15.43 | 2.07 |

is exact for nets with two or three nodes, which form the majority of all nets.

Tables IV and V show wire-length and run-time comparisons on a Pentium II/300 for the ACM/SIGDA and ISPD98 circuit suites, for 250, 500, 1000, and 2000 force-step iterations. Gravity outperforms generic 3-D partitioning placement using the most powerful efficient partitioning algorithm currently available. On the more established ACM/SIGDA suite, Gravity with 250 force-step iterations runs on the average a factor of 12.5 faster while producing placements with approximately 12% less wire length. By increasing the number of iterations to 2000, Gravity can improve the wire-length advantage to over 15% while still requiring only half the time of the hMetis partitioning placer. For the newer ISPD98 circuit suite with the larger and more modern circuits, Gravity performs even better. With a 1/13 of the run time, 250-iteration Gravity produces results that are on the average almost 20% better than the partitioning placer. This advantage can be increased to 22.6% with a speed-up of 2.3 by using 2000 iterations.

The target circuits for which Gravity is expected to compute placements in the future are expected to be large. For this reason it is encouraging to observe that Gravity performs even better and faster on the benchmark circuit suite with the larger circuits.

As a final indication of the potential of 3-D Gravity, we found in Obenaus [2000] that the wire-length improvement of 3-D Gravity over 3-D hMetis partitioning placement is roughly twice that of 2-D Gravity over 2-D hMetis partitioning placement.

Table V. Overview of 3-D Gravity versus 3-D Partitioning Placement for the ISPD98 Suite

| Circuit | hMetis | | 250 iterations | | 500 iterations | | 1000 iterations | | 2000 iterations | |
|---|---|---|---|---|---|---|---|---|---|---|
| | length | time (s) | change (%) | speed-up | change (%) | speed-up | change (%) | speed-up | change (%) | speed-up |
| ibm01 | 92,601.5 | 239.33 | −15.16 | 13.87 | −16.39 | 8.18 | −16.63 | 4.38 | −17.20 | 2.38 |
| ibm02 | 202,121.7 | 391.13 | −15.44 | 13.69 | −15.78 | 8.23 | −15.98 | 4.46 | −16.45 | 2.31 |
| ibm03 | 234,600.9 | 432.92 | −16.24 | 12.17 | −18.42 | 7.05 | −19.00 | 3.81 | −18.97 | 2.05 |
| ibm04 | 301,324.1 | 497.46 | −22.79 | 11.96 | −23.47 | 7.36 | −23.99 | 3.90 | −24.45 | 2.05 |
| ibm05 | 367,004.5 | 553.55 | −23.79 | 13.79 | −25.11 | 8.44 | −25.11 | 4.58 | −26.00 | 2.40 |
| ibm06 | 333,985.0 | 655.11 | −21.58 | 13.21 | −22.93 | 8.01 | −24.08 | 4.28 | −24.37 | 2.17 |
| ibm07 | 473,844.3 | 1,084.43 | −20.14 | 15.04 | −22.07 | 9.15 | −23.39 | 4.92 | −22.98 | 2.56 |
| ibm08 | 531,860.7 | 1,191.15 | −22.54 | 14.79 | −23.64 | 9.39 | −24.42 | 5.17 | −24.73 | 2.73 |
| ibm09 | 617,201.7 | 1,263.16 | −23.44 | 13.13 | −24.49 | 7.95 | −25.24 | 4.43 | −25.27 | 2.36 |
| ibm10 | 832,125.1 | 1,761.46 | −22.16 | 15.04 | −24.57 | 9.12 | −25.22 | 4.82 | −26.18 | 2.50 |
| ibm11 | 872,118.2 | 1,660.90 | −26.25 | 13.72 | −27.83 | 8.28 | −29.17 | 4.40 | −29.83 | 2.29 |
| ibm12 | 991,783.9 | 1,863.66 | −20.84 | 14.57 | −21.12 | 9.56 | −21.59 | 5.40 | −22.10 | 2.89 |
| ibm13 | 1,000,941.8 | 1,864.37 | −19.58 | 12.49 | −21.08 | 7.59 | −21.80 | 4.03 | −22.27 | 2.12 |
| ibm14 | 1,657,408.1 | 3,064.37 | −17.56 | 12.07 | −20.12 | 6.84 | −21.58 | 3.70 | −21.89 | 1.98 |
| ibm15 | 1,994,685.8 | 3,768.55 | −10.54 | 12.19 | −13.31 | 7.27 | −14.07 | 3.89 | −14.27 | 2.13 |
| ibm16 | 2,222,138.0 | 4,029.56 | −14.28 | 11.52 | −17.02 | 7.13 | −18.07 | 3.84 | −18.65 | 2.04 |
| ibm17 | 2,745,042.7 | 4,462.59 | −18.23 | 12.66 | −20.31 | 7.83 | −21.32 | 4.21 | −21.72 | 2.18 |
| ibm18 | 2,639,356.6 | 4,284.66 | −24.62 | 11.47 | −27.01 | 7.07 | −28.92 | 3.66 | −29.61 | 1.94 |
| Average | | | −19.73 | 13.19 | −21.37 | 8.03 | −22.20 | 4.33 | −22.61 | 2.28 |

## 6. CONCLUSION

With Gravity, we have developed one of the first fast 3-D placement algorithms. To our best knowledge this is the first effectively linear time 3-D placement algorithm. With its linear run time, Gravity is suited for very large circuits. The previously published 3-D placement algorithm by Ohmura [1998] has a significantly higher run-time complexity, $O(n \cdot m)$ where $n =$ number of nodes, and $m =$ number of nets. Thus it is not equally well suited for very large circuits. 3-D placement algorithms such as 3-D quadrisection [Alexander et al. 1996; Leeser et al. 1998], or recursive hMetis partitioning placement also have at least a complexity of $O(n \log n)$.

In Section 5.1, we provided evidence that large circuits benefit the most from 3-D placements. We also showed that even a small number of layers in the third dimension provides significant wire-length improvements for large circuits. For example, the two largest circuits in Table II exhibited a 64% and 74% reduction in wire-length in a full 3-D placement. However the majority of the wire-length savings materialized in five- or six-layer placements.

In Section 5.2, we compared Gravity to another promising near-linear time methodology. We used the currently fastest and best published partitioner to implement a 3-D minimum cut partitioning placer. Gravity outperformed these placements in wire length by more than 11% while being at least an order of magnitude faster (at 250 iterations).

### ACKNOWLEDGMENTS

We would like to thank Prof. Karypis for making the hMetis partitioner library available.

## REFERENCES


ALEXANDER, M. J., COHOON, J. P., COLFLESH, J. L., KARRO, J., PETERS, E. L., AND ROBINS, G. 1996. Placement and routing for three-dimensional FPGAS. In *Proceedings of the 4th Canadian Workshop on Field-Programmable Devices*. 11–18.

ALPERT, C. J. 1998. The ISPD98 circuit benchmark suite. In *Proceedings of the International Symposium on Physical Design*. 85–90.

ANTREICH, K. J., JOHANNES, F. M., AND KIRSCH, F. H. 1982. A new approach for solving the placement problem using force models. In *1982 IEEE International Symposium on Circuits and Systems*. 481–486.

BRGLEZ, F. 1993. ACM/SIGDA design automation benchmarks: Catalyst or anathema? *IEEE Des. Test Comput. 10*, 3 (Sept.), 87–91.

CHANG, R.-I. AND HSIAO, P.-Y. 1993. Force directed self-organizing map and its application to VLSI cell placement. In *1993 IEEE International Conference on Neural Networks*. 103–109.

CHIRICESCU, S. M. S. A. AND VAI, M. M. 1998. A three-dimensional FPGA with an integrated memory for in-application reconfiguration data. In *1998 IEEE International Symposium on Circuits and Systems*. Vol. 2. 232–235.

DEPREITERE, J., NEEFS, H., VAN MARCK, H., VAN CAMPENHOUT, J., BAETS, R., DHOEDT, B., THIENPONT, H., AND VERETENNICOFF, I. 1994. An optoelectronic 3-D field programmable gate array. In *Field-programmable logic: architectures, synthesis, and applications : 4th International Workshop on Field-Programmable Logic and Applications*. Lecture notes in computer science, vol. 849. 352–360.

EISENMANN, H. AND JOHANNES, F. M. 1998. Generic global placement and floorplanning. In *Proceedings of the 35th ACM/IEEE Design Automation Conference*. 269–274.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability*. W. H. Freeman and Company, New York.

GOTO, S. 1981. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Trans. Circuits Syst. CAS-28*, 1 (Jan.), 12–18.

HANAN, M. 1966. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Mathem. 14*, 2 (Mar.), 255–265.

KARYPIS, G., AGGARWAL, R., KUMAR, V., AND SHEKHAR, S. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*. 526–529.

KLEINHANS, J. M., SIGL, G., JOHANNES, F. M., AND ANTREICH, K. J. 1991. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. Computer-Aided Des. 10*, 3 (Mar.), 356–365.

KOFORD, J. S. 1998. Method and system for improving a placement of cells using energetic placement units alternating contraction and expansion operations. *United States Patent* 5754444.

LEESER, M., MELEIS, W. M., VAI, M. M., CHIRICESCU, S., XU, W., AND ZAVRACKY, P. M. 1998. Rothko: A three-dimensional FPGA. *IEEE Designs and Test of Computers 15*, 1 (Jan.), 16–23.

LEIGHTON, F. T. AND ROSENBERG, A. L. 1986. Three-dimensional circuit layouts. *SIAM J. Comput. 15*, 3 (Aug.), 793–813.

OBENAUS, S. T. AND SZYMANSKI, T. H. 1999. Placement benchmarks for 3-D VLSI. In *VLSI: Systems on a Chip*, L. M. Silveira, S. Devadas, and R. Reis, Eds. Kluwer Academic Publishing, 447–455.

OBENAUS, S. T. H. 2000. Fast placement algorithms for grids in two and three dimensions. Ph.D. thesis, McGill University.

OHMURA, M. 1998. An initial placement algorithm for 3-D VLSI. In *1998 IEEE International Symposium on Circuits and Systems*. Vol. 6. 195–198.

PARAKH, P. N., BROWN, R. B., AND SAKALLAH, K. A. 1998. Congestion driven quadratic placement. In *Proceedings of the 35th ACM/IEEE Design Automation Conference*. 275–278.

REBER, M. AND TIELERT, R. 1986. Benefits of vertically stacked integrated circuits for sequential logic. In *1996 IEEE International Symposium on Circuits and Systems*. 121–124.

SHAHOOKAR, K. AND MAZUMDER, P. 1991. VLSI cell placement techniques. *ACM Comput. Surveys 23*, 2 (June), 143–220.

TANPRASERT, T. 2000. Analytical 3-D placement that reserves routing space. In *2000 IEEE International Symposium on Circuits and Systems*. Vol. 3. 69–72.

TIA, T.-S. AND LIU, C. 1993. A new performance driven macro-cell placement algorithm. In *European Design Automation Conference*. 66–71.

TONG, C. C. AND WU, C. 1995. Routing in a three-dimensional chip. *IEEE Trans. Comput. 44*, 1 (Jan.), 106–117.

TSAY, R.-S. AND KUH, E. 1991. A unified approach to partitioning and placement. *IEEE Trans. Circuits Syst. 38*, 5 (May), 521–533.

TSAY, R.-S., KUH, E. S., AND HSU, C.-P. 1988. PROUD: A fast sea-of-gates placement algorithm. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*. 318–323.

UEDA, K., KITAZAWA, H., AND HARADA, I. 1985. CHAMP: Chip floor plan for hierachical VLSI layout design. *IEEE Trans. Computer-Aided Design CAD-4*, 1 (Jan.), 12–22.

VYGEN, J. 1997. Algorithms for large-scale flat placement. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*. 746–751.

WIPFLER, G. J., WIESEL, M., AND MLYNSKI, D. A. 1982. A combined force and cut algorithm for hierachical VLSI layout. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*. 671–677.

Received July 2001; revised February 2003; accepted February 2003