PARTITIONING-BASED STANDARD-CELL GLOBAL PLACEMENT WITH AN EXACT OBJECTIVE

Dennis J.-H. Huang[†] *and Andrew B. Kahng*[†‡*]

[†] *UCLA Computer Science Dept., Los Angeles, CA 90095-1596 USA*

[‡] *Cadence Design Systems, Inc., San Jose, CA 95134 USA*

ABSTRACT

We present a new top-down quadrisection-based global placer for standard-cell layout. The key contribution is a new general gain update scheme for partitioning that can exactly capture detailed placement objectives on a per-net basis. We use this gain update scheme, along with an efficient multilevel partitioner, as the basis for a new quadrisection-based placer called QUAD. Even though QUAD is a global placer, it can achieve significant improvements in wirelength and congestion distribution over GORDIAN-L/DOMINO [SDJ91] [DJS94] (a leading quadratic placer with linear wirelength objective and detailed placement improvement). QUAD can be easily extended to capture various practical considerations; our timing-driven placement can obtain wirelength savings (as well as small cycle time improvements) versus the SPEED [RE95].

1. INTRODUCTION

In the physical implementation of high-performance, complex deep-submicron integrated circuits, module placement is a critical step. Given fixed decisions from the upstream stages of the chip design flow – namely, microarchitecture design, chip timing, chip planning, logic synthesis and physical floorplanning – it is placement solution quality that is the major determinant of whether timing correctness and routing completion can be achieved. This paper describes a new placement tool for the standard-cell methodology; we assume a row-based layout with uniform module heights and variable module widths, with instance sizes of up to several tens of thousands of cells being of greatest interest. For overviews of (standard-cell) placement, see, e.g., Lengauer [Len90] or Shahookar and Mazumder [SM91].

A VLSI circuit netlist consists of a set of modules (cells) connected by signal nets. In the corresponding edge- and vertex-weighted netlist hypergraph $N(V,E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$, the n vertices correspond to netlist modules (cells) and the m hyperedges correspond to signal nets. Each hyperedge $e \in E$ is a subset of V containing one *source* vertex, with the remaining vertices of the hyperedge being *sinks*. The input to a placer is assumed to be the netlist and cell library information.

*This work was supported by a grant from Cadence Design Systems, where ABK is currently Visiting Scientist (on sabbatical leave from UCLA) (abk@cs.ucla.edu, ak@cadence.com). Dennis J.-H. Huang is currently at Synopsys, Inc. (dhuang@synopsys.com).

Define the location of a cell (and all its pins) to be the location of its center. A *placement* of the n cells in V is an assignment of cells to locations in two-dimensional plane. The placement is *legal* if cells are not overlapped and are placed within the prescribed row coordinates. The placer typically seeks a legal placement of V, such that layout area is minimized while maintaining auto-routability and satisfying timing and other performance constraints. For cell-based placement, the first-order objective is to place connected cells closer together to reduce both wirelength and lower bounds on signal delay. Thus, most placers have a minimum-wirelength objective: Given a netlist $N(V,E)$, find a legal placement such that $\sum_{e \in E} cost(e)$ is minimized, where $cost(e)$ is the routing cost of the net e.

It is difficult to estimate routed wirelength, and hence only simple estimates are used in practice. Let $MST(e)$ denote the *minimum spanning tree* (MST) cost over the locations of cells belonging to net e. Also, let $HP(e)$ denote the *half perimeter* of the minimum enclosing bounding box of the locations of cells belonging to net e. In practice, total MST cost and total HP cost are the most commonly used wirelength estimates for wirelength driven placement; any other practical estimate needs to have similarly low time complexity of evaluation. (Following several previous works, we will use the MST estimate for illustrative purposes and to evaluate total wirelength of our placements; however, our placer handles arbitrarily complicated per-net placement objectives.)

2. PARTITIONING-BASED PLACEMENT

Our proposed placement approach is based on top-down partitioning. In this section, we first review the traditional (KL-FM) iterative partitioning approach, along with its gain update scheme. We then review several partitioning-based placement techniques in the literature, centering on the issue of terminal propagation. We will omit discussion of local-improvement techniques (e.g, simulated annealing [SS93] [SS95] and DOMINO [DJS94]).

2.1. Gain Update in Iterative Partitioning

Iterative improvement heuristics for netlist partitioning typically start with an initial solution and make a series of *passes*. Each pass iteratively determines the *move* of one or more cells which achieves the best possible *gain* in the partitioning objective. After all cells have been moved in a given pass, the best solution seen during the entire pass is selected; the next pass begins with this selected solution. The process terminates when a local minimum is reached, i.e., the current pass does not improve the objective. Computing and updating gain data is the heart of the iterative improvement approach.

The prototype iterative heuristic is that of Kernighan and Lin (KL) [KL70], which uses a pair-swap move structure. During each pass, every cell is moved exactly once between two partitions. At the beginning of the pass, all cells are "unlocked", i.e., free to be swapped. Iteratively, the pair of unlocked cells with highest gain is swapped. After the selected cells are swapped, they become "locked" and the algorithm updates both the cost of the new partition and the gains of the remaining unlocked cells. After all cells are locked, the lowest-cost partition encountered over the entire pass is restored and returned. Further passes are executed, each using the result from the previous pass as its starting point, until no improvement results. Computing gains in the KL heuristic is expensive; $O(n^2)$ swaps are evaluated before every move, resulting in a complexity per pass of $O(n^2 \log n)$ (assuming a sorted list of costs). The method of Fiduccia and Mattheyses (FM) [FM82] reduces the time per pass to linear in the size of the netlist (i.e., $O(p)$, where p is the total number of pins) by adopting a single-cell move structure, and a *gain bucket* data structure that allows constant-time selection of the highest-gain cell and fast gain updates after each move.

2.2. Min-Cut Placement

Placement by recursive (bi-)partitioning is based on repeated division of a given circuit into subhypergraphs to optimize a given partitioning objective. With each partitioning of the circuit, the given layout area is partitioned in either the horizontal or the vertical direction. Each subhypergraph is assigned to a partition; when each subhypergraph has only one cell, then each cell will have been mapped to a unique (non-overlapping) position on the chip. Early approaches which use a min-cut partitioning objective are due to such authors as Breuer [Bre76] [Bre77] or Lauther [Lau79]. Most modern partitioning-based placers use some form of KL-FM partitioning heuristic, also with the minimum net-cut objective. Because the minimum net-cut is a poor abstraction of the real placement cost function (e.g., only in some limiting sense will total cuts capture total (MST) wirelength), various devices have been used to improve min-cut placement; the most important of these are *quadrisection* and *terminal propagation*.

2.3. Quadrisection

While many placement tools have relied on top-down min-cut bipartitioning, the main disadvantage of such an approach is that it can greedily obtain very good results in the first cut, but then bad results in successive cuts. The placement problem is essentially two-dimensional, in that we assign cells to locations in a planar layout. However, min-cut bisection adopts a one-dimensional approach, partitioning the netlist along a single cut line at each step.

Suaris and Kedem [SK87b] [SK87a] [SK88] [SK89] use quadrisection to divide the chip, yielding a truly two-dimensional placement procedure and results that are superior to those of top-down bipartitioning placement. Their quadrisection algorithm uses an extension of the FM heuristic which also runs in linear time per pass. Since a cell in one quadrant can be moved to any of the other three quadrants, there are 12 *gain buckets*, each corresponding to a pair of quadrants. At each step, a cell with highest gain is selected. Suaris and Kedem also apply a more accurate cost function which considers different horizontal and vertical weights.

2.4. Terminal Propagation

When partitioning a (sub-)circuit into several parts, it is not sufficient to consider only the netlist induced over the modules in the subcircuit, i.e., only the internal nets. Nets connecting to external IO pads or other cells in another (higher-level) partition must also be considered. Dunlop and Kernighan [DK85] proposed the *terminal propagation* technique which adds to the current netlist *dummy cells* that are fixed in the appropriate partitions.

For quadrisection, the terminal propagation technique is shown in Figure 1. The figure shows that block B_2 is about to be partitioned into $\{B_{20}, B_{21}, B_{22}, B_{23}\}$. Cells C in B_{02} and E in B_3 are connected to cells D and F in B_2. It would be beneficial to assign D to B_{20} and F to either B_{21} or B_{23}. The terminal propagation is done by inserting dummy cells fixed in specific blocks: in this example, dummy cell G is fixed in block B_{20} and dummy cell H is allowed to move only between B_{21} and B_{23}.

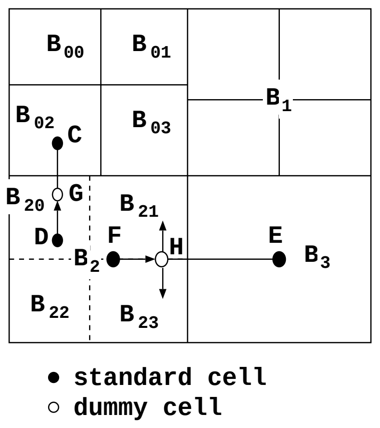

Figure 1. Terminal propagation for quadrisection placement.

3. A GENERAL GAIN UPDATE SCHEME FOR ITERATIVE PARTITIONING

In this section, we introduce an efficient, unified approach to updating gains for arbitrary objective functions during iterative multi-way partitioning. This technique is general, and can capture particular placement objectives exactly for individual nets; it is enabling to the new top-down placer described in the next section.

Given a k-way partitioning $\{P_0, P_1, \ldots, P_{k-1}\}$, define the *configuration* of a given net to be the distribution of its cells into the partitions P_j, $j = 0, 1, \ldots, k-1$. Each P_j contains either zero or a nonzero number of cells in the net. For each net e, let $c_j(e)$ be the number of cells in e that are distributed in partition P_j, i.e., $c_j(e) = |\{v | v \in e \text{ and } v \in P_j\}|$. We can use a binary number $f_0 f_1 \ldots f_{k-1}$ to represent each configuration, where $f_j = 1$ if $c_j(e) \geq 1$ and $f_j = 0$ if $c_j(e) = 0$. There are at most $2^k - 1$ different configurations for each net in a k-way partitioning. Figure 2 shows the 15 possible configurations in a 4-way partitioning.

We use $conf_{id}(f_{k-1} f_{k-2} \ldots f_0) = \sum_{j=0}^{k-1} 2^{f_j}$ to denote the *configuration_id* of a given net. In our new gain update scheme, each net e has an associated net vector V_e with length $2^k - 1$. Each entry of the net vector $V_e[d]$ corresponds to the *net cost* of the configuration with configuration_id d. The net cost can be specific to an underlying objective function, as shown by the following examples.

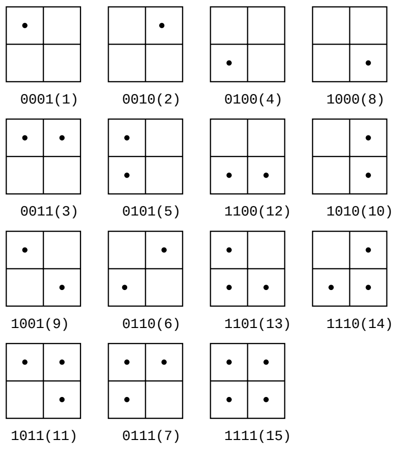

Figure 2. Fifteen configurations for 4-way partitioning. Each configuration is represented by a 4-bit binary number as shown at the bottom. The numbers in parentheses are the configuration_ids.

- **Net-cut Cost**: Minimize $\sum_{e \in E} cost(e)$, where $cost(e) = 1$ if net e distributes cells in more than one partition; $cost(e) = 0$ otherwise.
- **Absorption Cost** [SS93]: Minimize $\sum_{e \in E} cost(e)$, where $cost(e) = k - 1$, if net e distributes cells in exactly k partitions.
- **Quadratic Cost**: Minimize $\sum_{e \in E} cost(e)$, where $cost(e) = \frac{k(k-1)}{2}$, if net e distributes cells in exactly k partitions.
- **Sum-of-degrees Cost**: Minimize $\sum_{e \in E} cost(e)$, where $cost(e) = 0$, if net e distributes cells in one partition; $cost(e) = k$, if net e distributes cells in exactly k partitions.
- **MST Cost**: This is a special objective for 4-way partitioning. The hypergraph is partitioned among the upper-right, upper-left, lower-right and lower-left quadrants of the layout. Minimize $\sum_{e \in E} cost(e)$, where $cost(e)$ is the MST routing cost based on the cell distribution of a net e.

As noted above, Sanchis [San89] developed a multi-way gain computation with lookahead for net-cut cost; she also developed gain computation schemes for absorption cost and quadratic cost in [San93]. Here, we propose to use the net vector concept to unify the gain computation for various objectives. Examples of net vectors with different values corresponding to different objectives are shown in Table 1 for 4-way partitioning. The method can be extended to any k-way partitioning as long as k is not too large.

A more detailed discussion of the gain computation and update is now in order. We will center on the MST cost objective and 4-way partitioning. This is because our placement approach is based on recursive quadrisection. Also, since the MST is more accurate than net-cut as an estimate of routing cost, our placer uses an MST cost objective instead of the traditional cut-based objective.

Objective function	Net vector $V_e[0..15]$
Net-cut cost	0,0,0,1,0,1,1,1,0,1,1,1,1,1,1
Absorption cost	0,0,0,1,0,1,1,2,0,1,1,2,1,2,2,3
Quadratic cost	0,0,0,1,0,1,1,3,0,1,1,3,1,3,3,6
Sum-of-degrees cost	0,0,0,2,0,2,2,3,0,2,2,3,2,3,3,4
MST cost (4-way)	0,0,0,1,0,1,2,2,0,2,1,2,1,2,2,3

Table 1. Net vector entries according to various objective functions for 4-way partitioning.

We first observe that the net vector given in Table 1 for the MST cost objective assumes unit wire cost in both the horizontal and vertical directions. In other words, resources, congestions and routing costs are equal in both directions. In practice, horizontal and vertical wire costs should be weighted according to the available resource, e.g., a three-layer HVH design might be relatively richer in horizontal resources, while a four-layer HVHV design might be relatively richer in vertical resources. Let h and v respectively be the unit costs of horizontal and vertical wiring. We can easily create a net vector to capture this kind of objective function, as shown in Table 2.

i	$V_e[i]$	i	$V_e[i]$	i	$V_e[i]$	i	$V_e[i]$
0	0	1	0	2	0	3	h
4	0	5	v	6	$h+v$	7	$h+v$
8	0	9	$h+v$	10	v	11	$h+v$
12	h	13	$h+v$	14	$h+v$	15	$h+2v$

Table 2. Net vector entries for quadrisection with different horizontal and vertical weights (wire costs).

We next observe that our partitioning algorithm will use the same FM gain bucket data structure as in [FM82]. However, our gain computation is different from that of previous works. There are $k(k-1)$ gain buckets for k-way partitioning. We let $\gamma^j(v)$ denote the gain for moving cell v to partition j. Suppose cell a is moved from partition P_s to partition P_t. For each net e incident to a, we must update the gain of each cell $b \in e$ $(b \neq a)$ as it moves from its current partition P_x to partition P_y, i.e., $b \in P_x$ and $y \neq x$. To see how this gain update can be accomplished in constant time, consider the following four configurations:

1. $conf_0$ is the original configuration for net e (i.e., $a \in P_s$ and $b \in P_x$);
2. $conf_1$ is the configuration after a is moved to P_t (i.e., $a \in P_t$ and $b \in P_x$);
3. $conf_2$ is the configuration after b is moved to P_y but before a is moved (i.e., $a \in P_s$ and $b \in P_y$); and
4. $conf_3$ is the configuration after b is moved to P_y and after a is moved (i.e., $a \in P_t$ and $b \in P_y$).

The gain for moving cell $b \in P_x$ to P_y before a is moved is

$$gain_{a \in P_s}(b) = V_e[conf_{id}(conf_0)] - V_e[conf_{id}(conf_2)]$$

and this same gain after a is moved is

$$gain_{a \in P_t}(b) = V_e[conf_{id}(conf_1)] - V_e[conf_{id}(conf_3)].$$

The gain update for moving cell b to P_y is

$$\Delta\gamma^y(b) = gain_{a \in P_t}(b) - gain_{a \in P_s}(b)$$

and thus computing the gain update for cell b requires looking up only the four configurations $conf_0, \ldots, conf_3$, independent of k.

Figure 3 shows an example of the gain update computation. In the figure, the 4-pin net e contains cells $\{a, b, c, d\}$: a is in P_1, b is in P_0, and c and d are in P_2. Cell a is about to be moved to P_0, and we would like to update the gain for moving cell b to P_3. The current configuration for net e (i.e., $conf_0$) is "0111". After a is moved to P_3, $conf_1$ is "0101". The configuration after moving b to P_3 before a is moved (i.e., $conf_2$) is "1110". The configuration after moving b to P_0 after a is moved (i.e., $conf_3$) is "1101". Therefore, the gain update for moving b to P_3 is

$$\begin{aligned}\Delta\gamma^3(b) &= (V_e[conf_{id}(0101)] - V_e[conf_{id}(1101)]) \\ &\quad -(V_e[conf_{id}(0111)] - V_e[conf_{id}(1110)]) \\ &= (V_e[5] - V_e[13]) - (V_e[7] - V_e[14])\end{aligned}$$

If net cut is our objective function, the gain update is $\Delta\gamma^3(b) = (1-1)-(1-1) = 0$. By contrast, if MST cost is our objective function, the gain update is $\Delta\gamma^3(b) = (1-2)-(1-1) = -1$. In other words, the net cut cost does not change when a and b are moved to their new partitions, while the MST cost is reduced by 1.

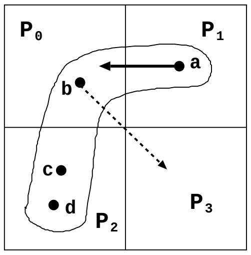

Figure 3. A gain update example: cell a is moved from P_1 to P_3, and we wish to update the gain of cell b moving from P_0 to P_3.

Figure 4 summarizes our gain update scheme when cell a is moved from P_s to P_t. We also illustrate how to efficiently compute the four configurations using bit operations. We emphasize that this gain update scheme can be used within almost any iterative partitioning approach, including k-way FM, 2-phase FM [BCL87], CLIP-FM [DD96a] and multilevel FM [Alp96].

```
Procedure Gain Update: ($a$ is moved from $P_s$ to $P_t$)
Input: $a$ = the cell to be moved from $P_s$ to $P_t$
       $k$ = number of partitions
Variables:
  $e \rightarrow confid$ = current configuration_id for net $e$
                   before $a$ is moved
  $b \rightarrow block$ = current partition that cell $b$ is located
  $b \rightarrow gain[j]$ = the gain for moving cell $b$ to partition $P_j$
Constants:
  $turnOn[0..k-1]$, where $turnOn[j] = 2^j$
  $turnOff[0..k-1]$, where $turnOff[j] = 2^k - 2^j - 1$
1. for each $e$ incident from $a$ do
       // compute two configurations before the move
2.     $confid_0 = e \rightarrow confid$
3.     for $j = 0$ to $k-1$ do
4.         $beforeSize[j] = afterSize[j] = c_j(e)$
5.     $afterSize[t]++;$
6.     $afterSize[s]--;$
7.     if $beforeSize[s] = 1$
8.         $confid_1 = confid_0$ & $turnOff[s]$
9.     if $beforeSize[t] = 0$
10         $confid_1 = confid_1 \mid turnOn[t]$
11.    $e \rightarrow confid = confid_1$
       // compute two configurations after the move
12.    for each $b \in e$ do
13.        $from = b \rightarrow block$
14.        for $to = 0$ to $k-1$ do
15.            if $to = from$
16.                continue
17.            if $beforeSize[from] = 1$
18.                $confid_2 = confid_0$ & $turnOff[from]$
19.            if $beforeSize[to] = 0$
20.                $confid_2 = confid_2 \mid turnOn[to]$
21.            if $afterSize[from] = 1$
22.                $confid_3 = confid_1$ & $turnOff[from]$
23.            if $afterSize[to] = 0$
24.                $confid_3 = confid_3 \mid turnOn[to]$
25.            $gain = (V_e[confid_1] - V_e[confid_3]) -$
                   $(V_e[confid_0] - V_e[confid_2])$
26.            $b \rightarrow gain[to] += gain;$
27.        endfor
28.    endfor
29.    for $j = 0$ to $k-1$ do
30.        $c_j(e) = afterSize[j]$
31.endfor
```

Figure 4. Template for gain update procedure.

4. A NEW TOP-DOWN QUADRISECTION BASED PLACER

We now describe a new top-down quadrisection-based placement algorithm, based on the gain update scheme described in the previous section along with a multilevel partitioning engine We also describe meta-heuristic approaches to improve solution quality, as well as means to accommodate such practical constraints as differing vertical and horizontal cut costs.

Our quadrisection-based placement is based on the gain update scheme of Section 3. The differences between our new placer and previous quadrisection-based placers are as follows. First, our approach handles instances with mixed floating and fixed pads, or even without any pads at all. E.g., if all IO pad locations are fixed, we only partition the internal (core) standard cells and do not need to partition the fixed IO pads. (Quadratic placement approaches such as PROUD [TKH88] [TK91] and GORDIAN [KSJ91] will degrade when there are no fixed pads to anchor the placement and spread out the locations of the core cells.) Second, our placer does not require terminal propagation, but still considers the exact connectivity from external blocks, as well as its exact impact on placement objectives such as MST or HP cost; the exact capture of arbitrary detailed objectives is enabled by the net cost vector concept. Traditional cut-based placement requires terminal propagation and cannot capture the placement objective. Finally, our placer integrates a number of practical extensions, including control of area utilization, timing-driven capability, capability to honor a given hierarchical netlist clustering, and control of relative routing resource demands in the horizontal and vertical wiring directions.

4.1. Multilevel FM-Based Partitioning

Our top-down quadrisection based placement (QUAD) is based on a *multilevel* iterative partitioner (ML) by Alpert et al. and the general gain update scheme of Section 3..

The basic multilevel algorithm consists of two phases, bottom-up matching (or clustering) and top-down partitioning. Given a netlist N_0, the matching phase of [Alp96] [AHK96] uses a matching-based clustering scheme for each level of the netlist. A clustering of N_0 is used to induce the coarser netlist N_1, then a clustering of N_1 induces N_2, etc. until the most coarsened netlist N_m is constructed. During the partitioning phase, a 2-way FM-based partitioning algorithm (e.g., LIFO FM [HHK95] or CLIP-FM [DD96b] [DD96a]) is applied at each level of the netlist. When a bipartitioning solution $P_m = \{X_m, Y_m\}$ is found for N_m, this solution is projected (unclustered) into $P_{m-1} = \{X_{m-1}, Y_{m-1}\}$, where it serves as the initial partitioning solution of N_{m-1} and is refined by the FM-based partitioner. The unclustering and refinement procedure continues until the original netlist N_0 is partitioned. A similar approach can be applied to multi-way partitioning. ML is efficient (an untuned implementation performs 4-way partitioning of a $25,000$-cell design in 32 CPU seconds on a SUN Ultra 1 (140 MHz)), and yields excellent results when compared against the best known methods from the literature [Alp96] [AHK96].

4.2. Net Vector Computation

During each stage of quadrisection, only the cells located in the current partition are movable; cells outside the current partition are fixed. We first compute the center coordinates of the four quadrants in the current partition. For each net e, we compute the number of pins located in the current partition, as well as all possible configurations with respect to the net e. Next, we evaluate the user-specified cost function (e.g., MST cost or half-perimeter) for the net e according to the pin distributions of all possible configurations, and normalize the costs so that the lowest cost is zero (this reduces the index of the highest-gain bucket, i.e., the maximum possible gain, and improves runtime efficiency). Finally, we assign the net costs to their corresponding net vector entries. Figure 5 shows a snapshot of the top-down quadrisection process, with the northeast quadrant as the current partition. In the figure, the northwest quadrant has already been quadrisected and the northeast quadrant will be processed next. Consider a 5-pin net with two pins located in the current partition and three pins fixed outside the partition, with one of the fixed pins an IO pad. There are 10 different configurations. Figure 5 illustrates the configuration_id, MST cost and half-perimeter cost for each configuration. If the MST cost function is selected, the net cost ordered by the configuration_id is $[-, 5, 7, 6, 7, 6, 9, -, 8, 7, 8, -, 8, -, -, -]$, and the resulting net vector is $[0, 0, 2, 1, 2, 1, 4, 0, 3, 2, 3, 0, 3, 0, 0, 0]$. When this quadrant is partitioned, the hypergraph instance contains a 2-pin net which has the above net vector. Again, our approach does not require terminal propagation, and exactly captures the placement cost function during partitioning. Figure 6 shows the algorithm template for QUAD.

4.3. Meta-Heuristic Improvements

We can further improve the placement quality by the following two operations.

- **Cycling**: At each level of quadrisection, we may *cycle* the partitioning process. Figure 7(a)-(d) illustrate the first iteration of the second level of quadrisection, where B_0, B_1, B_2 and B_3 are partitioned. After this first iteration, the cells of each net have been distributed to the centers of 16 blocks. In cycling, we begin the next iteration by repartitioning B_0 based on the new cell distribution as shown in Figure 7(e). When each iteration is finished, we compute the placement cost based on the new cell locations. This cycling process of Figure 7(e)-(h) is repeated until no further cost improvement is possible.

```
Quadrisection Based Placement (QUAD)
Input: Netlist $N$ with fixed IO locations
Output: cell location $cellLoc[]$
Variables:
    $Q$ = queue storing the subnetlists to be partitioned
    $N \rightarrow nRows$ = number of rows to be placed in netlist $N$
    $N \rightarrow nCells$ = number of cells in netlist $N$
    $N \rightarrow center$ = center location of netlist $N$
  $newN$ = removeIOPad($N$);
  $Q$.push($newN$);
  while $Q$ is not empty do
      $currN = Q$.pop();
      if ($currN \rightarrow nRows > 1$)
          compute4WayNetVector($currN$);
          ($P_0, P_1, P_2, P_3$) = Quadrisection($currN$);
          for each $i = 0$ to 3 do
              /* create a subnetlist according to $P_i$. */
              $N_i$ = formSubNetlist($P_i, currN$);
              forall $cell \in N_i$ do
                  $cellLoc[cell] = N_i \rightarrow center$;
              if ($N_i \rightarrow nCells > 1$)
                  $Q$.push($N_i$);
          endfor
      else
          compute2WayNetVector($currN$);
          ($P_0, P_1$) = Bisection($currN$);
          for each $i = 0$ to 1 do
              /* create a subnetlist according to $P_i$. */
              $N_i$ = formSubNetlist($P_i, currN$);
              forall $cell \in N_i$ do
                  $cellLoc[cell] = N_i \rightarrow center$;
              if ($N_i \rightarrow nCells > 1$)
                  $Q$.push($N_i$);
          endfor
      endif
  endwhile
  return $cellLoc$;
```

Figure 6. Quadrisection-based placement algorithm (QUAD) template.

- **Overlapping**: While cycling the partitioning procedure at each level, a second performance improvement is possible by performing the quadrisection on *overlapped* regions. Figure 8 shows nine overlapped regions that are quadrisected at the second level. In general, there are $(2^k - 1)^2$ overlapped regions at the k^{th} quadrisection level.

5. EXPERIMENTAL RESULTS

Our experiments were run on a Sun Ultra 1 (140 Mhz) with 192 MB RAM, and all runtimes reported (mm:ss) are for this machine. Our versions of the test cases were imported in PROUD [TK91] or timingPROUD format generated by colleagues at TU Munich (<http://www.regent.e-technik.tu-muenchen.de/>), and have up to $25,000$ cells.

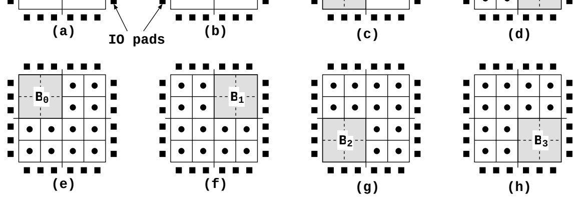

Figure 7. (a)–(d) The first iteration of the second level of quadrisection; (e)–(h) successive iterations of the second level of quadrisection.

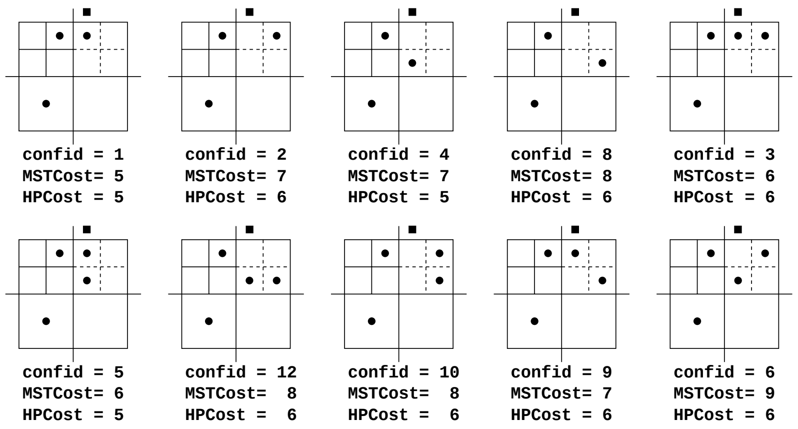

Figure 5. Ten configurations for a net with two pins in the current partition.

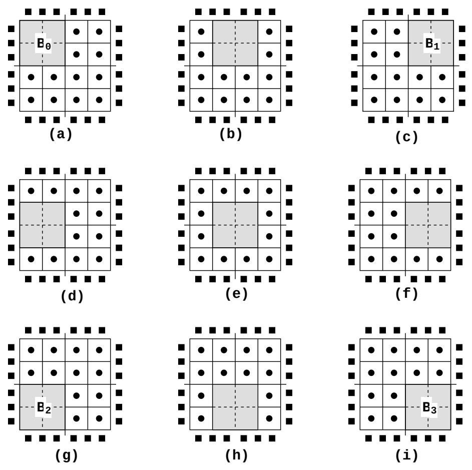

Figure 8. (a)–(i) Nine overlapped regions that are partitioned in the second level of quadrisection.

Our first experiment compares QUAD without cycling/overlapping (QUAD w/o CO), QUAD without overlapping (QUAD w/o O) and QUAD. All test cases were placed with 100% area utilization. The results are shown in Table 3. QUAD w/o CO averages 10% greater wirelength but can require as little as 17% of the runtime of QUAD for large benchmarks.

Our second experiment compares our quadrisection results with GORDIAN-L [SDJ91] and the post-processing detailed placer DOMINO [DJA94] on 18 benchmarks with 100% area utilization (results for GORDIAN-L/DOMINO were provided by Guenter Stenz [Ste97] at TU Munich). Note that GORDIAN-L is a global quadratic placement tool, while DOMINO is a detailed placer; QUAD should be considered as a global placer. The MST wirelength results are shown in Table 4. QUAD outperforms GORDIAN-L on 15 benchmarks, and performs about 1% worse on three benchmarks. The average MST wirelength improvement over GORDIAN-L is 4.8%. QUAD also performs slightly better than DOMINO. Table 5 compares QUAD against GORDIAN-L/DOMINO on the same set of benchmarks using the half-perimeter (HP) objective; this is the measure used by the authors of GORDIAN-L and DOMINO. QUAD has an average of 4.4% improvement over GORDIAN-L, but

	QUAD w/o CO	Q w/o O	QUAD	Impr. vs. Q w/o CO
Case	MSTx100(time)			
prim1	11432(01:02)	11196	10208(02:50)	10.7%
prim2	48674(06:31)	45736	44478(23:50)	8.6%
ind2	420299(33:52)	395434	380194(144:30)	9.5%
ind3	1070927(48:12)	1042366	970068(168:46)	9.4%
fract	410(00:05)	402	380(00:08)	7.3%
C1908	1963(00:21)	1925	1830(01:46)	6.8%
C5315	6887(00:58)	6421	6185(07:02)	10.2%
C6288	10285(03:05)	9004	8312(26:17)	19.2%
s1423	2456(00:27)	2384	2265(02:57)	7.8%
s1488	2684(00:36)	2539	2470(02:27)	8.0%
s5378	8807(01:26)	8640	8208(12:13)	6.2%
s9234	15229(02:41)	14782	13848(19:57)	9.1%
s13207	30733(04:23)	29236	28161(20:07)	8.4%
s15850	36478(05:32)	35002	33625(30:54)	7.8%
struct	5050(03:31)	4644	4296(09:55)	14.9%
biomed	38792(13:31)	36074	33787(64:35)	12.9%
avq_s	108266(45:38)	103862	95867(235:04)	11.5%
avq_l	114408(53:50)	110197	101930(315:50)	10.9%
Impr.				9.9%

Table 3. MST cost comparison of QUAD w/o CO, QUAD w/o O and QUAD.

uses 1.2% more wirelength than DOMINO.

We have also compared 2-D congestions as measured by a simple supply and demand model, where "supply" is the available horizontal and vertical routing tracks and "demand" is the MST routing edge for the net. We did this to verify that our wirelength improvements did not come at the cost of routing hotspots. Figure 9 depicts the overcongested areas of the avq_small placements generated by QUAD and DOMINO; overcongested resources are those for which the sum of vertical and horizontal demands exceeds the sum of supplies (for space reasons, we dispense with the details of these measurements). The QUAD placement has 0.8% overcongested area while the DOMINO placement has 1.2% overcongested area. Thus, although QUAD uses 1% more wirelength for this case, it has better congestion distribution than DOMINO.

6. EXTENSIONS TO TIMING-DRIVEN PLACEMENT

We have extended our basic quadrisection-based global placement engine in a number of directions. One direction of interest is timing-driven placement, where simple extensions allow the top-down quadrisection to be driven by net cost vectors that capture both timing and wirelength aspects of the circuit layout. Our timing-driven im-

Case	GORD-L	DOMINO	QUAD	Impr. GOR-L	Impr. DOMI
	MSTx100				
prim1	10500	10059	10208	2.8%	-1.5%
prim2	45994	43705	44478	3.3%	-1.8%
ind2	436300	417264	380194	12.9%	8.9%
ind3	1121000	1048673	970068	13.5%	7.5%
fract	400	383	380	5.0%	0.8%
C1908	1858	1767	1830	1.5%	-3.6%
C5315	6220	5922	6185	0.6%	-4.4%
C6288	8794	8339	8312	5.5%	0.3%
s1423	2334	2208	2265	3.0%	-2.6%
s1488	2680	2558	2470	7.8%	3.4%
s5378	8609	8182	8208	4.7%	-0.3%
s9234	14848	14023	13848	6.7%	1.3%
s13207	31284	29995	28161	9.9%	6.1%
s15850	37020	35591	33625	9.2%	5.5%
struct	4160	3967	4196	-0.9%	-5.8%
biomed	34677	33712	33787	2.6%	-0.2%
avq_s	95648	92355	95867	-0.2%	-3.8%
avq_l	100650	97825	101930	-1.3%	-4.2%
Impr.				4.8%	0.3%

Table 4. MST cost comparison of GORDIAN-L, DOMINO and QUAD.

Case	GORD-L	DOMINO	QUAD	Impr. GOR-L	Impr. DOMI
	HPx100				
prim1	9171	8900	8972	2.2%	-0.8%
prim2	38702	36542	36824	4.9%	-0.8%
ind2	354850	333019	332318	6.3%	0.2%
ind3	1040444	974327	938682	9.8%	3.7%
fract	360	339	337	6.4%	0.6%
C1908	1568	1497	1520	3.1%	-1.5%
C5315	5612	5344	5466	2.6%	-2.3%
C6288	7084	6690	6663	5.9%	0.4%
s1423	2140	2025	2075	3.0%	-2.5%
s1488	1813	1639	1623	10.5%	0.9%
s5378	7908	7522	7578	4.2%	-0.7%
s9234	12975	12321	12217	5.8%	0.8%
s13207	27547	26559	26234	4.8%	1.2%
s15850	33132	31946	31647	4.5%	0.9%
struct	3816	3499	3780	0.9%	-8.0%
biomed	25170	23697	23765	5.6%	-0.3%
avq_s	62824	59075	62890	-0.2%	-6.4%
avq_l	65894	61966	65906	-0.0%	-6.4%
Impr.				4.4%	-1.2%

Table 5. Half perimeter cost comparison of GORDIAN-L, DOMINO and QUAD.

plementations update net cost vectors according to various schemes, e.g., based on timing analysis that is interleaved with the partitioning. Table 6 shows results comparing our timing-driven placement results with those of SPEED [RE95]. Here, "Delay" (a sort of "cycle time") is the maximum path delay between any pair of sequentially adjacent storage elements (flip-flops). Path delays are computed using pin parasitics and cell intrinsic delays from the timing-PROUD library data, along with a centroid-star net model and Elmore delay for the interconnect. This is the same delay evaluation (with the same interconnect parasitics) used in [RE95], except that we apply factors of 1/2 in the Elmore delay expressions that were not applied in [RE95]. We see that timing-driven QUAD ("Timing-QUAD") outperforms SPEED by an average of 3% in terms of delay while maintaining an average of 4.7% less MST cost.

We have also compared Timing-QUAD with the TimberWolf simulated annealing based timing-driven placement package (results obtained from Swartz [Swa96]) on the three test cases fract, struct and avq_small using the same technology parameters as in the previous experiment. For each test case, TimberWolf uses different IO locations, number of rows and row locations. Thus, comparisons with TimberWolf involve completely different QUAD results from those of Table 6. The TimberWolf comparison with Timing-QUAD is shown in Table 7; the two packages seem very comparable.

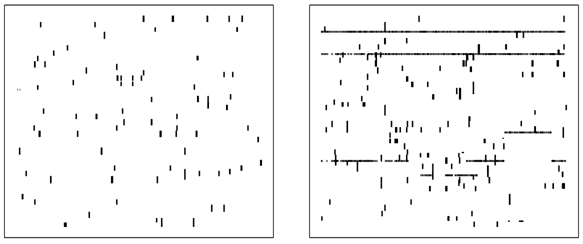

Figure 9. Map of avq_small placement overcongestions with respect to sum of horizontal and vertical demands, for (a) QUAD (0.8% overcongested), and (b) DOMINO (1.2% overcongested).

Test Case	Measure	QUAD	SPEED	Timing-QUAD	Impr. vs. SPEED
fract	Delay	18.8	18.4	18.4	0 %
	MSTx100	380	433	442	-2.1%
C1908	Delay	19.8	19.9	18.4	7.9%
	MSTx100	1830	2291	1792	21.8%
C5315	Delay	23.9	21.6	22.1	-2.3%
	MSTx100	6185	6763	6454	4.6%
C6288	Delay	64.5	61.4	61.6	-0.3%
	MSTx100	8312	13891	9564	31.1%
s1423	Delay	36.4	33.0	33.5	-1.4%
	MSTx100	2265	2907	3114	-7.1%
s1488	Delay	9.5	11.9	9.2	22.3%
	MSTx100	2470	4235	2593	38.8%
struct	Delay	82.0	77.7	79.3	-2.1%
	MSTx100	4296	5521	5244	5.0%
biomed	Delay	30.7	29.7	29.3	1.4%
	MSTx100	33787	40892	39935	2.3%
avq_s	Delay	75.3	75.0	71.1	5.2%
	MSTx100	95867	98094	102435	-4.4%
avq_l	Delay	93.3	76.7	76.9	-0.3%
	MSTx100	101930	110034	115234	-4.7%
Avg. Improv.	Delay				3.0%
	MST				4.7%

Table 6. Comparison of timing-driven QUAD and SPEED.

REFERENCES

[AHK96] C. J. Alpert, L. W. Hagen, and A. B. Kahng. "A Hybrid Multilevel/Genetic Approach for Circuit Partitioning." In *Proc. ACM/SIGDA Physical Design Workshop*, pp. 100–105, 1996.

[Alp96] C. J. Alpert. *Multi-way Graph and Hypergraph Partitioning*. PhD thesis, University of California, Los Angeles, 1996.

[BCL87] T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. "Graph Bisection Algorithms with Good Average Case Behavior." *Combinatorica*, **7**(2):171–191, 1987.

[Bre76] M. A. Breuer. "Min-cut Placement." *Design Automation and Fault-Tolerant Computing*, **1**(4):343–362, 1976.

[Bre77] M. A. Breuer. "A Class of Min-cut Placement Algorithm for the Placement of Standard Cells." In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 284–290, 1977.

[DD96a] S. Dutt and W. Deng. "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 194–200, 1996.

[DD96b] S. Dutt and W. Deng. "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement

Case	Measure	Max Intrinsic Path Delay	TW7.0	Timing-QUAD
fract	Delay	10.6	17.9	18.1
	MSTx100		349	347
struct	Delay	40.0	78.8	79.3
	MSTx100		5130	5103
avq_s	Delay	37.3	61.4	60.9
	MSTx100		46763	47153

Table 7. Comparison of timing-driven QUAD and TimberWolf7.0.

Techniques." In *Proc. ACM/SIGDA Physical Design Workshop*, pp. 92–99, 1996. Also see corresponding Technical Report, Dept. of Electrical Engineering, U. Minnesota.

[DJA94] K. Doll, F. M. Johannes, and K. J. Antreich. "Iterative Placement Improvement by Network Flow Methods." *IEEE Transactions on Computer-Aided Design*, **13**:1189–1200, 1994.

[DJS94] K. Doll, F. M. Johannes, and G. Sigl. "Iterative Placement Improvement by Network Flow Methods." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **13**(10):1189–1199, 1994.

[DK85] A. E. Dunlop and B. W. Kernighan. "A Procedure for Placement of Standard Cell VLSI Circuits." *IEEE Transactions on Computer-Aided Design*, **4**(1):92–98, 1985.

[FM82] C. M. Fiduccia and R. M. Mattheyses. "A Linear Time Heuristic for Improving Network Partitions." In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 175–181, 1982.

[HHK95] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng. "On Implementation Choices for Iterative Improvement Partitioning Algorithms." In *Proceedings European Design Automation Conf.*, pp. 144–149, 1995.

[KL70] B. W. Kernighan and S. Lin. "An Efficient Heuristic Procedure for Partitioning Graphs." *Bell Syst. Tech. J.*, **49**(2):291–307, 1970.

[KSJ91] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization." *IEEE Transactions on Computer-Aided Design*, **10**(3):356–365, 1991.

[Lau79] U. Lauther. "A Min-cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation." In *Proceedings of the 16th Design Automation Conference*, pp. 1–10, 1979.

[Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, 1990.

[RE95] B. M. Riess and G. G. Ettelt. "SPEED: Fast and Efficient Timing Driven Placement." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 377–380, 1995.

[San89] L. A. Sanchis. "Multiple-Way Network Partitioning." *IEEE Transactions on Computers*, **38**(1):62–81, 1989.

[San93] L. A. Sanchis. "Multiple-Way Network Partitioning with Different Cost Functions." *IEEE Transactions on Computers*, **42**(22):1500–1504, 1993.

[SDJ91] G. Sigl, K. Doll, and F. M. Johannes. "Analytical Placement: A Linear or a Quadratic Objective Function?" In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 427–432, 1991.

[SK87a] P. R. Suaris and G. Kedem. "Quadrisection: A New Approach to Standard Cell Layout." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 474–477, 1987.

[SK87b] P. R. Suaris and G. Kedem. "Standard Cell Placement by Quadrisection." In *Proceedings IEEE Intl. Conf. Computer Design*, pp. 612–615, 1987.

[SK88] P. R. Suaris and G. Kedem. "An Algorithm for Quadrisection and Its Application to Standard Cell Placement." *IEEE Transactions on Circuits and Systems*, **35**(3):294–303, 1988.

[SK89] P. R. Suaris and G. Kedem. "A Quadrisection-based Combined Place and Route Scheme for Standard Cells." *IEEE Transactions on Computer-Aided Design*, **8**(3):234–244, 1989.

[SM91] K. Shahookar and P. Mazumder. "VLSI Cell Placement Techniques." *Computing Surveys*, **23**(2):143–220, 1991.

[SS93] W-J. Sun and C. Sechen. "Efficient and Effective Placements for Very Large Circuits." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 170–177, 1993.

[SS95] W. Swartz and C. Sechen. "Timing Driven Placement for Large Standard Cell Circuits." In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 211–215, 1995.

[Ste97] G. Stenz. 1997. Personal communication.

[Swa96] W. Swartz, 1996. Personal communication.

[TK91] R.-S. Tsay and E. S. Kuh. "A Unified Approach to Partitioning and Placement." *IEEE Transactions on Circuits and Systems*, **38**(5):521–533, 1991.

[TKH88] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu. "PROUD: A Sea-of-Gates Placement Algorithm." *IEEE Design & Test of Computers*, **5**(6):44–56, 1988.