Effective Partition-Driven Placement with Simultaneous Level Processing and Global Net Views*

Ke Zhong and Shantanu Dutt
Dept. of EECS, University of Illinois-Chicago

Abstract : **In this paper we take a fresh look at the partition-driven placement (PDP) paradigm for standard-cell placement for wire-length minimization. The goal is to develop several new algorithms for incorporation into a PDP framework that can rectify the well-known drawbacks of traditional PDP (increasingly localized view of nets with increasing levels of the partitioning tree, min-cut objective, inaccuracy and cost of terminal propagation (TP), irreversibility of move decisions), while preserving its considerable advantages (time efficiency, flexibility in accurately incorporating many optimization metrics, and flexibility in satisfying most constraints). We have developed several novel techniques within a PDP-based framework that yield the best wire-length results so far on all but two of the MCNC benchmark suite. Our major innovations are: (1) simultaneous level partitioning (SLP) in which we partition the entire circuit globally in every level of the partitioning tree, across the current cutline(s); (2) cell gain computation based on a global or distributed view of entire nets (thus obviating TP) and on the bounding-box (BB) minimization of nets (as opposed to mincut in prior PDP); (3) move irreversibility tackled in a post-processing phase via vertical and horizontal swaps. Empirical results indicate that our PDP algorithm** SPADE **(for Simultaneous level PArtitioning with Distributed [i.e., global] nEt views) provides almost 20% better wirelength results than an internal version of "regular" PDP with min-cut based gains, 10.8% better than the previous best PDP method** QUAD**, 10.6% better than** TimberWolf (TW) 7.0**, 15.8% better than the state-of-the-art force-directed technique from U. Munich (termed** FD-98 **here), and 15.3% better than the multilevel placement technique** `Snap-On`**. Besides** TW7.0**, we are also the only ones to report results on the approximately 100K-cell circuit** golem3 **(12.2% better than** TW7.0**). Our run times are quite reasonable.**

1 Introduction

Cell placement is a decisive phase in physical design of VLSI circuits. With the advent of deep sub-micron (DSM) technology, placement tools must be able to handle large input circuits (up to tens of millions of transistors), and optimize different objectives (delay, power, area, etc.) under possibly multiple constraints (like crosstalk bounding and thermal distribution). Bearing these in mind, we find hierarchical partitioning to be a desirable framework for placement, because of the following three properties:

⋆ It is a divide-and-conquer type approach, which is in general time-efficient for large inputs.

⋆ It implies top-down processing. Thus, many circuit parameters can be considered at the same time, enabling us to take more balanced and gradually refined decisions with a global view.

⋆ It offers great flexibility in tackling multiple constraints. Particularly, the partitioning engine can be implemented in an iterative improvement (local search) fashion to proceed with multi-constraint satisfaction on a per-move basis.

We term any placement algorithm based on hierarchical partitioning as a *partition-driven placement (PDP)* method. Basically, PDP is based on repeated subdivision of the given netlist and layout surface (core region) into subcircuits and rectangular regions, respectively, and assignment of subcircuits to those regions [1, 4]. When each subcircuit consists of only one cell, it is also uniquely placed in a region, marking the termination of PDP. Such a process can be represented by construction of a partition-tree (see Fig. 1b), where each node in the tree represents a region and circuit cells assigned to it. The root node is the starting core region, containing the whole circuit. All nodes of same depth in the tree constitute a *level*. In this paper, we focus on application of the PDP methodology to standard cell circuit placement. Given user specified row number and row length upper bound, our objective is to get a placement with minimized total half-perimeter wire length, which is free of cell overlap.

Other mainstream placement algorithms are largely based on either simulated annealing (SA) [12, 13] or mathematical programming [14, 15, 16] (LP, max-flow, or force-directed quadratic programming) methodologies. SA (e.g., TW7.0) is well-known for its good solution quality as applied to placement problems, but can be fairly slow. The Gordian-L/Domino package, based on linear programming and 'soft' partitioning [14, 15], offers good trade-off between run-time and solution quality. Another force-directed method FD-98 [16] gives slightly better performance than TW7.0, and is considered to be the current state-of-the-art. Mathematical programming based methods, though efficient and effective *per se*, suffer from rigid formulation constraints (e.g., linear or quadratic programming). Hence, such algorithms usually adopt some imprecise models (e.g., cell-overlap and lumped-capacitance delay), and render modifications accommodating certain additional constraints difficult. An orthogonal paradigm of two-phase placement is proposed in the `Snap-On` placer [18] which offers good flexibility and enables early prediction of final placement results. In the first phase, the input netlist is hierarchically placed in 'global bins' using a combination of min-cut partitioning for the higher levels (the hMetis partioner is used in [18]) and simulated annealing with min-wirelength objective at the lower levels. This global placement is then refined in the detailed placement phase using simulated annealing.

The PDP methodology is not without drawbacks. First of all, placement is a problem that, in general, cannot be solved optimally through a divide-and-conquer approach like PDP, i.e., optimal solutions to subproblems from PDP (optimal placement within separate regions) may not constitute a globally best solution. Specifically, any cell moved across a cutline during previous level of processing can never go back. Such irreversibility tends to limit the search ability of PDP for a good local optimum when reaching lower levels down the partition hierarchy. Further, the traditional PDP method pursues minimization of *cutsize* across cutlines, while the actual objective is to minimize (estimated) wire length or layout area. Such mismatch in objectives of optimization can have an adverse impact on overall solution quality. We have developed a set of techniques to effectively deal with all the above issues, that are discussed in Sec. 3.

The rest of the paper is organized as follows. Section 2 describes previous work on two fronts: traditional PDP framework (with terminal propagation), and circuit partitioning algorithms. Section 3 consists of our new techniques for making the PDP paradigm significantly more effective: *simultaneous level partitioning*, *global net views*, *min-wirelength partitioning*, and *swap-based post-processing*. We call our PDP algorithm that incorporates these features SPADE (Simultaneous level PArtitioning with Distributed [i.e., global] nEt views). Other important features of SPADE are also discussed in this section. Experimental results under various settings are given in Sec. 4. These include internal comparisons showing utility of certain techniques, and com-

*This work was partly funded by a grant from Intel Corp.

parisons with other placement tools. Conclusions are in Sec. 5.

2 Previous PDP Work

The PDP method was first studied in the late 70's [1]. Sequential cutsize minimization is proposed, so that all regions in the corresponding partition tree are bi-partitioned in a breadth-first manner, and cutsize is minimized locally for each region under processing, regardless of possible resultant suboptimality for other regions or levels not yet processed. We call this sequential level processing (SeqLP), and it can be formulated as

$$obj(i,j) = min\{c(i,j)\}|min\{c(i,j-1)\}|...|\min\{c(i,0)\},$$
$$(1 \le i \le L,\ 1 \le j \le R(i)).$$

where L is total number of levels, R(i) is the number of regions on the i-th level, obj(i, j) is the (local) objective of processing j-th region in i-th level, and c(i, j) is cutsize across region j in i-th level. It is also illustrated in Fig. 1.

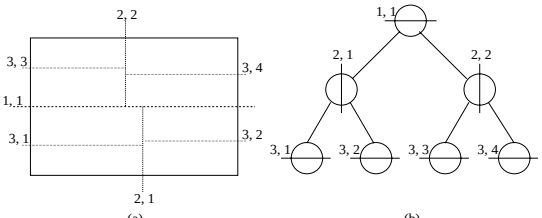

Figure 1: *Illustration of Sequential Level Processing (SeqLP). (a) Regions under SeqLP. (b) Corresponding partition tree.*

An important contribution that makes this framework complete is *terminal propagation* (TP), which results in as much as 30% of improvement in placement quality [2]. The significance of TP lies in the fact that bi-partitioning of each region is not independent of others, and such external influence are largely captured by producing dummy cells serving as virtual I/O pads at the region boundary under processing. However, the TP algorithm of [2], based on some Steiner tree heuristic, is time-consuming. Combined with TP, iterative SeqLP in each level becomes possible. Examples of iterative SeqLP based on quadrisection are 'iterative improvement' [3] and 'cycling' [4].

Circuit Partitioning Engine: Circuit partitioning engines are crucial to the performance of any PDP algorithm. A comprehensive survey of circuit partitioning is given in [9]. Most state-of-the-art bi-partitioners are of the iterative-improvement family, and can be regarded as variants of FM [6], which is linear time per pass. Dutt and Deng proposed two novel iterative-improvement partitioners (IIPs); PROP [7] considers futuristic as well as immediate gain, while CLIP [8] is a general cluster-oriented scheme that can be overlaid on any IIP engine. More recently, two multi-level partitioners hMetis [10] and MLc [11] with improved results and lower runtimes were proposed.

3 Our New PDP Techniques

In this section we present the general framework of our PDP algorithm SPADE, and elaborate on the new PDP techniques used in it, and how they address the drawbacks of classical PDP discussed earlier. In the sequel, we will use *p*, *n* and *e* to denote the total number of pins, cells and nets, respectively, in a circuit, and we will use *d* to denote the degree of an arbitrary net.

3.1 Simultaneous Level Partitioning (SLP) with Global Net Views

To consider the interdependencies among multiple regions under PDP in each level of the partition tree, we partition all regions simultaneously, instead of using the traditional SeqLP paradigm of all previous PDP algorithms [1, 2, 3, 4]. At the same time, information on the local region to which each cell belongs to is maintained, and moves are still local (between the two child regions generated by local cutline). The move sequence is, however, global, i.e., the current best-gain feasible cell across the entire circuit is moved. Cell gains are computed and updated (at each move) based on the global pin distribution of their nets. Pins of each net are stored in a binary search tree (pin-tree) sorted by their coordinates along the direction perpendicular to the current cutlines. In this way, during any level of processing, we can always have a global view of pin distribution of each net, and can find their bounding box (BB) geometries quickly.

For example, in Fig. 2(a), cutlines 3 and 3' (they are actually 'slice lines', to be explained later) are processed together in the third level (assuming cutline 1, and 2 have been processed in the first two levels). As a result, four regions are cut simultaneously. Every cell move will potentially have a global (out-of-region) effect, and the best prefix point after each pass of partitioning is also determined by the overall solution cost at that level, as opposed to local gain update and prefix point determination in previous SeqLP methods. After each cell is moved, we need also to update gains of neighboring cells outside the current region. Again, in Fig. 2, if cell *u*, *v* and *w* are connected by a three-pin net n_1, suppose the horizontal dimension of each level-4 partition is one unit, then *w*'s gain is +1, *u*'s gain -1, and *v*'s gain 0. However, if *w* is moved, *v*'s gain will get updated and become -1, although they are in different regions.

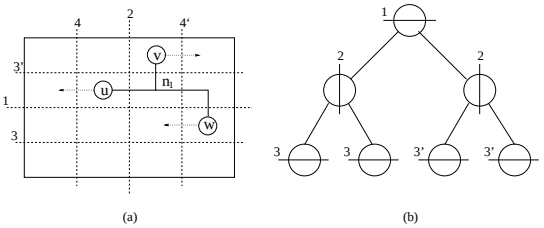

Figure 2: *Illustration of Simultaneous Level Partitioning (SLP) used in SPADE. (a) Regions under SLP. (b) Corresponding partition tree.*

As a comparison, terminal propagation, required for previous SeqLP methods, also takes into account external pin distributions regarding nets inside current region, but it needs to generate additional dummy cells on-the-fly. Moreover, in some cases, TP cannot help SeqLP escape local optima, or it might not be accurate enough. Such an example is shown in Figs. 3b-c. These show that SeqLP cannot escape a local optimum when the subregions are partitioned in clockwise order starting from the upper left quadrant; the cluster move sequence is B, D, E, H, which results in a wire-length reduction of 22*x*, where *x* is the horizontal dimension of each of the 8 regions. However, with SLP, the move order of the clusters is chosen for global optimality, and is H, A, D, E, which results in a greater wirelength reduction of 25*x*. Further, Fig. 3d shows that cell gain calculations based on local subnets using TP are not always accurate. In the figure, cell u_1 can be in either A_1 or A_2 subregions initially, while u_2 can be in either B_1 or B_2 initially; thus they are shown on the cutline to depict this generality. Using TP and the local subnet of n_i in the $\{B_1, B_2\}$ region, the gain of u_2 (w.r.t. n_i) will be 0 in both directions as the nearest dummy position only is considered. The dummy on the upper-left portion of the net does not affect u_2's gain. However, a global view of net n_i, as considered in SPADE, will correctly compute u_2's gain to move from right to left as positive as that decreases the BB of n_i. Note that in SPADE, TP is obviated, since there is no concept of 'external' nets (every net is now globally visible).

Decoupled Regions–A Caveat in SLP: Since standard cell circuits are placed in uniform-height rows, the entire chip layout area is repeatedly divided into implicit row-sets at each horizontal partitioning step, and horizontal (local) cutlines across the regions spanning the same underlying row-set are perfectly aligned. We term such an aligned set of cutlines as a *slice line*. Cell gains in half-perimeter wire length measure

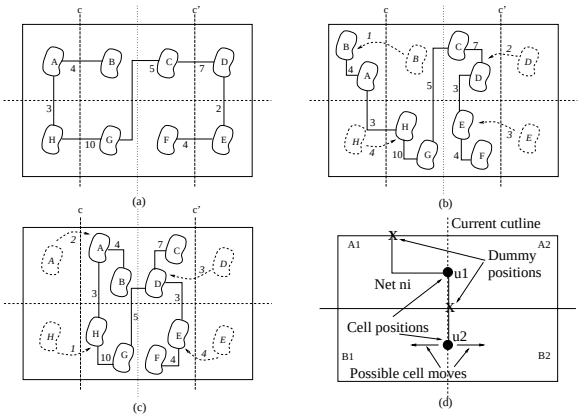

Figure 3: *(a) Initial region state before partitioning; costs of nets between clusters are shown. (b) Cluster moves (arrows are labeled with the move #) and final result in traditional SeqLP processing of subregions in clockwise order from upper left quadrant. (c) Cluster moves and final result showing greater wirelength reduction with SLP used in SPADE. (d) TP in SeqLP can result in misleading cell gains, while a global net view used in SPADE is accurate.*

is solely dependent on pin positions of incident nets. Meanwhile, each cell, once assigned to one side of a slice line, can never go back across it in later levels of processing. That means gain of any cell on one side of a (previous-level) slice line is independent of perturbations on the other side. In other words, any region in a row-set is *decoupled* from the regions in other row-sets, as summarized in Theorem 1.

Theorem 1 *For every net n_i spanning multiple row-sets, its net length change within any row-set is independent of cell moves (due to horizontal partitioning) in other row-sets. Hence, horizontal partitioning of multiple row-sets can be performed sequentially, without loss of solution quality.*

Proof Sketch: Cell moves in different row-sets are only in the vertical direction, thus the horizontal dimension (and coordinates) of the net BB's cannot change. Nets spanning row-sets cannot be removed from the previous levels' slice line(s). Thus changes in pin distribution of such nets in one row-set cannot affect the wire length reduction possible for such nets due to cell moves in other row-sets. □

To exploit such separability of row-based placement, we implemented horizontal partitioning (at any level) as processing each set of regions spanning the same row-set individually, starting from the bottom row-set. Within any row-set, regions are partitioned simultaneously. For instance, in Fig. 2(a), the two lower quadrants spanning slice line 3 are simultaneously processed before the two upper quadrants spanning slice line 3'.

Similarly, in higher levels of processing, we performed vertical partitioning by cutting one 'column' of regions at a time, starting from the leftmost column. Although vertical cutlines are not perfectly aligned, their deviation in coordinates are not significant at higher levels due to the balance constraint in partitioning. When vertically non-cuttable regions (e.g., single-cell regions) start to appear, our PDP algorithm switches to SLP automatically. In general decoupled SLP leads to greatly reduced run time and improved wire length results. The pseudo code for (horizontally and vertically decoupled) SLP is given in Fig. 4.

Choice of cut directions for each level remains important in SPADE. Our experiments revealed that the quadrature style (alternating cut direction in adjacent levels) gives the best results. Hence, all later discussions and experimental results are based on quadrature-style cutline generation.

3.2 Wire-Length Based Gain Computation

Traditional PDP algorithms are oriented toward cutsize minimization. This is a crude approximation of the actual wire length objective, especially when there are many cells of greatly varied dimensions.

Algorithm SLP(partition_tree, head_region) /* Simultaneous partitioning of regions (accessible through head_region) in partition tree */
  best_cost = ∞;
  for (i=0; i<trial_num; i++) /* perform *trial_num* random runs */
    `init_processing`(partition_tree); /* necessary initialization */
    `init_partition`(head_region); /* random initial partitioning */
    done = 0;
    while (!done)
      `init_gain_cmp`(head_region); /* at the beginning of each pass, compute gains of all free cells within current set of regions, using *intrinsic-gain-pairs* of connected net segments */
      while (*cell* = `global_select_cell`() != *NULL*)
      /* select the globally best-gain cell that is non-violating */
        `move_cell`(*cell*); /* lock the cell, record its wire-length gain */
        `local_update_gain`(*cell*); /* update wire length gains of local neighboring cells */
        `global_update_gain`(*cell*); /* update *intrinsic-gain-pairs* of newly perturbed net segments (in other regions), recompute gains of cells connected to those net segments */
      pass_gain = `get_best_prefix`(); /* get best prefix point, revert moves after it, return overall gain of the pass */
      if (*pass_gain* ≤ 0)
        done = 1;
    if (*curr_cost* = `get_cost`() ≤ *best_cost*)
      best_cost = *curr_cost*;
      keep a copy of best result so far;
  update partition_tree with best recorded partitioning result;
  return;

Figure 4: *SLP algorithm used in SPADE: Simultaneous partitioning of multiple regions with global wire length reduction in a level of the partitioning tree.*

Hence, we introduce cost and gain measures based directly on half-perimeter wire length, which has been shown to be a good estimate of wire length [5]. Also, at the end of each level of processing, we recompute the width of each newly generated child region as the total size of cells contained in it, and perform compaction of regions along horizontal slice lines (row-based placement). This is also carried out to estimate the wire length achieved after each run of SPADE, so that the best wire length (assuming every cell at the center of its region) so far is computed with more precision.

Since a net can have pins in multiple regions, we term any maximal subset of pins of a net within a region as a 'net segment'. For example, if net n_i has three pins, p_1, p_2, p_3, and p_1, p_2 are in one region, while p_3 is in another, then (p_1, p_2) and (p_3) are the two net segments of n_i. There are up to three possible configurations of any net segment, defined by distribution of its *free-to-move* pins relative to the cutline going across its containing region, as shown in Fig. 6a-b. We assign a bit label to identify a subregion relative to the cutline: lower or left (upper or right) subregion has label 0 (1); see Fig. 6d. Also, when all free pins are in the lower or left (upper or right) subregion, a net segment is said to be in config. 0 (config. 1); otherwise it is in config. 2 (spanning the cutline). At any level, a net's half-perimeter length is uniquely determined by the current configurations of all its constituent net segments. When a cell is moved, all net segments connected to it are perturbed. Hence, gain of a cell c_i can be computed as $gain(c_i) = \sum_{c_i \in n_j} gain(c_i, n_{j,k})$, where $n_{j,k}$ (the k-th net segment of n_j) is connected to c_i. However, if $n_{j,k}$'s configuration is not changed due to the move of a cell connected to it, $gain(c_i, n_{j,k}) = 0$. Only those net segments whose configurations are *changed* by the current move contribute to a change in wire length.

To compute cell gains efficiently, we record a pair of gains $(g^{n_{i,j}}[0], g^{n_{i,j}}[1])$, termed the *intrinsic-gain-pair (IGP)* for each net segment $n_{i,j}$, which is computed as follows ($WL(n_i)$ is the half-perimeter wire length of n_i).

$$g^{n_{i,j}}[0] = WL(n_i)|_{n_{i,j} \text{ in config. } 2} - WL(n_i)|_{n_{i,j} \text{ in config. } 0}$$
$$g^{n_{i,j}}[1] = WL(n_i)|_{n_{i,j} \text{ in config. } 2} - WL(n_i)|_{n_{i,j} \text{ in config. } 1}$$

Even if a net segment has only one free pin, we can compute $IGP(n_{i,j})$ by assuming a virtual config. 2. As we can see, $g^{n_{i,j}}[0]$

```
gain(c_i) = 0;
side = get_side(c_i); /* get current side of c_i */
for each net segment n_{j,k} connected to c_i
    if (n_{j,k} is in config. side)
        gain(c_i) -= g^{n_{j,k}}[side];
    if (c_i is the only free cell connected to n_{j,k} in current side)
        gain(c_i) += g^{n_{j,k}}[other(side)];
```

Figure 5: *Computation of cell gain based on intrinsic-gain-pairs of incident net segments.*

($g^{n_{i,j}}[1]$) of net segment $n_{i,j}$ is the change in wire length of net n_i when $n_{i,j}$ switches from config. 2 to config. 0 (config. 1). Hence, computation of IGP of $n_{i,j}$ is dependent on the current global pin distribution of n_i. For instance, in Fig. 6d, net segment $IGP(n_{i,1})$ is computed as (0.25, 0), assuming that the horizontal dimension of each quadrant is 2 units. The *IGP*'s are then used to compute cell gains as specified in the procedure in Fig. 5.

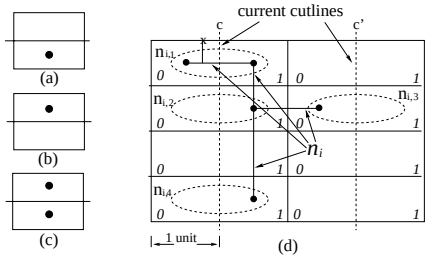

Figure 6: *Three possible net segment configurations: (a) config. 0 (b) config. 1 (c) config. 2; (d) Intrinsic-gain-pairs of net segments of net n (x is a fixed pad, dashed lines are current cutlines): IGP($n_{i,1}$)=(0.25, 0), IGP($n_{i,2}$)=(0, 0), IGP($n_{i,3}$) =(1, 0), IGP($n_{i,4}$) =(0, 0).*

After a cell is moved, however, any neighbors within the same region will have their gains updated (note that an 'intrinsic' gain pair never changes at local moves, but can be changed by moves of cells connected to the same net outside its region). Moreover, gains of those neighbors external to the current region, which are connected to other net segments of the perturbed nets, need to be updated in a two-step fashion, i.e., first update intrinsic-gain-pairs of their corresponding net segments, then recompute gains of connected cells. This corresponds to the `global_update_gain()` procedure in Fig. 4.

Inter-region update of intrinsic-gain-pairs, if done exhaustively, can be $O(d^2)$ in time for a net of degree d (consider the case in which d pins of net n_i are distributed in exactly d regions, we may need to update IGPs of all $(d-1)$ other net segments at each move of a pin). This is not acceptable if the input circuit has very-large-degree nets. However, there is redundancy in exhaustive update. For example, in Fig. 6d, when $n_{i,3}$ has one pin moved across its local cutline, and switches to another configuration, there is no need to update the IGPs of the other net segments of n_i. For large-degree nets, only those pins really close to boundaries of its current bounding box can trigger inter-region updates. In general, we can show that the overhead incurred by such updates, based on our binary search tree data structure for pins of each net, is proportional to $d(\log d)$ for a net with a large d (updates of small nets with 2 to 4 pins, which are the majority of nets in real circuits, consume constant time).

We now analyze the per-pass time requirement of our PDP algorithm. First, wire length based gains are not discrete in value, so bucket structures of standard FM cannot be used. Instead, we use binary search trees to store the gains, thus introducing an additional $\log n$ factor for updating the gain of a cell. For all local gain update, we need $\Theta(p \log n)$ time. For inter-region update, since update of intrinsic-gain-pairs takes time $\Theta(\sum(d \log d)) = O(p \log p)$, and then we have to update gains of each connected cell, we need time $O(p \log p \log n)$. Hence the total time per-pass is $O(p \log^2 n)$ (since $p = \Theta(n)$).

3.3 Tackling Move Irreversibility: Post-Processing with Horizontal and Vertical Swaps

To address the irreversibility problem of PDP, we tried several techniques for post-processing for PDP, facilitating potentially global movement of cells. The first of such is a traditional method [17], in which each row is scanned from left to right, with neighboring pairs of cells swapped if there is positive gain (in wire length), and the process is restarted from the first row, until no further improvement can be achieved.

Then, intra-row clustering is included to further improve the gain achievable through the above neighbor-swap method. Our clustering technique features an internal-to-external net count ratio (I/E). Every net that has all its pins inside the cluster is considered 'internal', and any partially connected net is then 'external'. Initially, for any single cell, the ratio is always zero, since every net is external. Starting from the leftmost cell, each row will be scanned, and the profile of I/E ratio recorded along the way. A cluster is formed whenever I/E drops, after scanning a new cell, below a pre-determined threshold. This is intuitive since a cluster is considered to be strongly connected only when the number of internal nets is high relative to external connections. Care is also taken to make the clusters more or less equal in size. We do clustered and non-clustered neighbor swaps alternately until convergence is detected. Details of clustering is shown in Fig. 7.

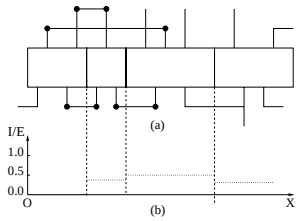

Figure 7: *Internal (thick) and external (thin) nets of a cluster, and its I/E ratio profile along the row direction*

However, this intra-row swap scheme converges to a local optimum very quickly (typically within a couple of iterations). To pull it out of local optimum, we also considered vertical (inter-row) swap of cells (given the row length limit is satisfied), with step size possibly greater than one (i.e., between non-adjacent rows). To limit computation time, we only swap cells vertically when their boundaries are horizontally overlapping each other. However, vertical swaps, unlike horizontal ones, tend to cause cell overlaps, and must be resolved by shifting at least one of overlapped subrows either to the left or right. Hence, we have to estimate, to a certain precision, the change in wire length due to such shifts. Note that if a pin is moved under the assumption that all other pins of the same net are fixed, there are only one or two pins that are really relevant to any change in wire length [13]. We developed a procedure to estimate shift-induced wire length change based on such juncture-pair modeling. Introduction of vertical swaps indeed improves our results by an additional 2 or 3 percent, but timing overhead is high due to shift-oriented estimation.

3.4 Circuit Partitioning Engine

The circuit partitioning engine invoked in our PDP program SPADE is CLIP-FM^{wl}, which is an adapted version of CLIP-FM [8] for wire length minimization. The way CLIP favors cluster-removal is that it selects next (legal) move with maximum updated part of gain, instead of aggregate FM gain[1] Hence, the gain update scheme of FM^{wl}

[1]We actually use a variation of CLIP in which the gains of cells on nets perturbed for the first time are magnified by $1/(shrink\ factor)$ where $0 \leq shrink\ factor \leq 1$. When $shrink\ factor = 0$, we get CLIP (infinite amplification), and when $shrink\ factor = 1$, we obtain regular FM.

is left intact, and CLIP-FM^{wl} has the same per-pass time complexity as FM^{wl}. However, through empirical studies we found that CLIP-FM^{wl} converges with number of passes proportional to $\log n$.

When applied to SPADE, we need to introduce the following non-trivial modifications to CLIP-FM as a circuit bi-partitioner. First of all, since there are multiple regions to be bisected simultaneously, an efficient data structure is needed to facilitate selection of best-gain, yet non-violating (with respect to all constraints under consideration) cell to move. Initially we used two 'global' binary search trees to store all free cells on either side of any cutline, only to discover later that violating cells tend to accumulate at higher-gain end of each trees, making further search for legal moves prohibitively time-consuming (since we always re-start at the best-gain node of each tree, it is obviously a quadratic-time search). Hence, in later implementations we used a two-level tree structure, i.e., cells in each region are first stored in a pair of trees locally, and all best-gain nodes of local trees are then sorted in a pair of global trees also. Then an efficient search strategy is applied, as detailed below.

Global Cell Selection Strategy: Based on the two-level tree structure, we can still access the globally best-gain cell in constant time. If a violation occurs, we first determine the type of violation involved. For vertical partitioning, the only constraint is local tolerance for each region, so we only need to select the best-gain cell in the opposite local tree to move. In some cases, e.g., the opposite local tree is empty, we need to delete the violating best-gain cell from the tree, and start the search process again. For horizontal partitioning, however, more complex decisions have to be made, since there are two types of constraints (for local region and row-set balance control, as explained in Sec. 3.5). If there is only local violation, we can proceed as in the vertical partitioning case. Otherwise, we prioritize row-set balance violation correction, and try the best-gain cell from the opposite global tree. If that cell is also violating, we search the entire opposite global tree until a feasible move is found. If no feasible moves are found, then the global tree in the side of the originally violating best-gain cell is searched.

One other element in our search strategy that contributes to higher solution quality is 'tie-breaking' for any pair of best-gain nodes in global trees. Here 'ties' specifically refer to the two best-gain cells from both global trees. The basic idea is that if both moves are feasible, then the one that follows the last move tends to continue cluster-removal and should be selected.

3.5 Maximum Row Length Control

For row-based standard cell placement problem, row length control is crucial in determining final die area utilization. Maximum row length forms a lower bound on chip width. In a general PDP process, even if we assign relatively small balance tolerance (say, 1%) to every region, the final maximum row length may still run out of control, since errors tend to accumulate. Hence, for each implicit row-set under (horizontal) partitioning, we must assign specific aggregate tolerances to it, in addition to those local tolerances of each region. This is, however, not needed in vertical partitioning.

Let us consider a row-set R under horizontal partitioning. Suppose a slice line divides R into two sub-row-sets R_0 and R_1, with r_0 and r_1 rows, respectively. To keep final length of any row in R within user-specified percentage deviation T_u from the average row length of the circuit $L_a = (\sum \text{cell widths})/r$ (r = total # of rows), the following two inequalities regarding row-set level tolerances T[0] and T[1] must hold:

$$(r_0/(r_0+r_1)+T[0])\cdot hor\_size(R) \le r_0 \cdot ML$$
$$(r_1/(r_0+r_1)+T[1])\cdot hor\_size(R) \le r_1 \cdot ML$$

where $ML = (1+T_u)L_a$. However, if we choose the row-set tolerances by setting the above inequalities as equal, then there is risk that such a budget on row length might be used up too early in higher levels of PDP, rendering extremely low (almost zero) tolerances for lower levels of partitioning. If that happens, the iterative improvement ability of the partitioner will be greatly hampered, yielding suboptimal results. Thus, we use a gradual assignment of row-set level budget by substituting ML with ML_0 and ML_1 in the above two inequalities for T[0] and T[1], respectively.

$$ML_0 = (1+[\alpha+(1-\alpha)/(1+\log_2 r_0)]T_u)L_a$$
$$ML_1 = (1+[\alpha+(1-\alpha)/(1+\log_2 r_1)]T_u)L_a$$

where $0 < \alpha < 1$. Then, local balance tolerances of regions in R can be computed based on the two row-set tolerances. Setting local tolerances for a region to be equal to its containing row-set tolerances overconstrains the problem. Hence we amplify the local tolerances by a factor of row-set tolerances that is greater than one, while still ensuring that the overall row-set tolerances are not violated. Experimental results reveal that setting α within [0.5, 0.75], combined with a horizontal tolerance amplification factor of 1.5 gives good wire length results, and makes final row length violation almost impossible.

Circuit	*cells*	*nets*	*pins*	*rows*
fract	125	147	462	6
primary1	752	902	2908	16
struct	1888	1920	5741	21
primary2	2907	3029	11219	28
biomed	6417	5742	21040	46
industry2	12142	13419	48158	72
industry3	15059	21939	65919	54
avqsmall	21854	22124	76231	80
avqlarge	25114	25384	82751	86
golem3	99932	143379	335529	192

Table 1: *Benchmark circuit statistics.*

Circuit	*SeqLP*		*SPADE*		
	WL	*run time*	*WL*	*run time*	*WL imprv.*
primary1	0.87	10	0.74	35	14.9%
struct	0.35	30	0.29	87	17.1%
primary2	3.62	66	3.13	256	13.5%
biomed	1.72	272	1.43	589	16.9%
industry2	15.23	554	11.90	2161	21.9%
industry3	43.78	787	35.37	3924	19.2%
Total	65.57	1719	52.86	7052	19.4%

Table 2: *WL (m's) and runtime (secs) comparison between SPADE and SeqLP approaches (both using 16 runs).*

4 Experimental Results

Experiments were run for ten MCNC standard cell benchmarks whose characteristics are given in Table 1. We used the same number of rows (also see Table 1) as [4, 13, 16, 18]. All experiments were conducted on a 550MHz Pentium-III Linux workstation. All reported wire length (WL) results are in meters, and run times are in seconds.

Table 2 provides a comparison of SPADE with an internally developed traditional SeqLP-based PDP program (using a somewhat better TP method than that used traditionally), both for 16 runs. Results show

Circuit	*8 runs*		*12 runs*		*16 runs*		
	WL	*time*	*WL*	*time*	*WL*	*time*	*best WL*
fract	0.022	1	0.022	2	0.024	2	0.022
primary1	0.76	19	0.76	28	0.74	35	0.72
struct	0.306	49	0.304	69	0.291	87	0.287
primary2	3.13	144	3.12	206	3.13	256	3.02
biomed	1.47	346	1.38	498	1.43	589	1.32
industry2	12.27	1340	12.20	1726	11.90	2161	11.59
industry3	37.44	2194	37.18	3118	35.37	3924	34.98
avqsmall	6.06	2205	5.92	2950	5.59	3679	5.16
avqlarge	6.59	2623	6.27	3569	6.16	4270	5.95
golem3	21.62	20196	21.05	25572	19.84	29842	19.84
Total	89.67	29117	88.21	37738	84.48	44845	82.89

Table 3: *Trade-off between SPADE run time and quality. The settings are: row length deviation up to 3%, H/V decoupling, shrink factor of 0.1, and H/V swap as post-processing. Also included, in the last column, is a set of results for best settings for each circuit (within 16 runs and 3-5% deviation).*

Circuit	*SPADE/16*	*TW7.0*	*FD-98*	*QUAD*	*Snap-On*
fract	0.024	–	–	0.034	–
primary1	0.74	0.83	0.87	0.90	0.95
struct	0.291	–	0.338	0.378	–
primary2	3.13	3.53	3.72	3.68	3.66
biomed	1.43	1.61	1.78	2.38∗	1.84
industry2	11.90	13.30	14.6	33.23∗	14.48
industry3	35.37	41.53	45.1	93.87∗	44.7
avqsmall	5.59	5.08	4.91	6.29	5.15
avqlarge	6.16	5.65	5.38	6.59	5.21
golem3	19.84	22.60	–	–	–
Total WL (8 / 8 ckts)	84.16 / 64.61	94.13 / –	– / 76.70	–	–
(7 / 6 ckts)	64.32 / 15.94	–	–	– / 17.87	75.99 / –
SPADE Imprv.	–	10.6%	15.8%	10.8%	15.3%

Table 4: *Wire length (meters) comparisons between SPADE/16 and TW7.0 [13], FD-98 [16], QUAD [4] and Snap-On [18]. The *'ed numbers for QUAD correspond to three circuits whose wire lengths are not consistent with those of the other methods, probably due to QUAD's use of circuits in the PROUD format. Our comparisons to QUAD thus do not include these three circuits.*

Circuit	*SPADE/16*	*SPADE/8*	*TW7.0*	*FD-98*	*QUAD*
fract	2	1	–	–	8
primary1	35	19	–	37	170
struct	87	49	272	40	595
primary2	256	144	–	152	1430
biomed	589	346	885	284	3875
industry2	2161	1340	3285	1283	8670
industry3	3924	2194	4504	1605	10126
avqsmall	3679	2205	4587	1741	14104
avqlarge	4270	2623	5501	2031	18950
golem3	29842	20196	–	–	–
Total (8 ckts)	15001	8929	–	7173	57920
Total (6 ckts)	14710	8766	19034	–	–
Scaled Time Ratio	1.54	1.0	1.07	0.4	1.79

Table 5: *Run time (secs) comparison of SPADE/16 and SPADE/8 to recent methods. Note that SPADE/8's wirelengths are 5.1%, 11.3% and 5.6% better than those of TW7.0, FD-98 and QUAD, respectively. The TW7.0 times are those reported in [16]. The "Scaled Time Ratio" row indicates the machine-based normalized time ratios of all methods to SPADE/8—a greater than 1 number for other methods indicate they are slower by that much factor. The scaled times were obtained according to the Heapsort benchmark in www.netlib.org, and our empirical findings that the 550 Mz Pentium III workstations we used are 3 times faster than the Sun Ultra 1 Model 170 machines (performance data for the Pentium III is not available). The Heapsort performance benchmark was chosen since placement requires some number crunching but also a more significant amount of dynamic data structure creation, access and update similar to that in Heapsort. The TW7.0 and FD-98 times are for a DEC Alphastation 250 4/266 (2 times slower than the 550 Mz Pentium III) and the QUAD times are for a SUN Ultra 1 Model 140 (3.6 times slower).*

that SPADE provides an improvement of almost 20% in wirelength over SeqLP, thus establishing that our new paradigm of SLP with global net views is superior to traditional PDP using TP. Table 3 exhibits the trade-off between run time and solution quality: results for 8, 12 and 16 runs at each level, under the same SPADE settings are shown. It can be seen that only the five largest circuits benefit appreciably from additional runs. On the average SPADE/16 obtains 5.8% smaller wirelength than SPADE/8 at the cost of 54% more runtime.

Table 4 compares SPADE/16 to four state-of-the-art methods representing four different approaches: QUAD (PDP) [4], TW7.0 (SA) [13]. FD-98 (mathematical programming) [16] and `Snap-On` [18] (two-phase placement). We obtain overall wire-length improvements of 10.8%, 10.6%, 15.8% and 15.3%, respectively, over these four techniques. Also, as shown in the table, our results are the best so far for all circuits except two, `avq_small` and `avq_large`, for which our results are worse than those of TW7.0, FD-98 and `Snap-On`, but better than those of QUAD. Besides TW7.0, we are the only ones to report results on `golem3`; SPADE/16 gives 12.2% better WL than TW7.0. Table 3's last column also shows the best SPADE results over different settings for each circuit— this gives a sense of what is achievable by a PDP-based method; it is about 2% better than the single SPADE/16 setting results reported in Tables 3 & 4. Finally, run times are compared in Table 5—`Snap-On` times are not compared as [18] reports times only for the global placement phase which is its main focus. Times for SPADE/16 are reasonable, ranging from 10 minutes for `biomed` (6.4K cells) to about 8.3 hours for `golem3` (100K cells); times for SPADE/8, which still gives significant improvements over recent methods (see caption of Table 5) are about 35% less than those for SPADE/16. The table shows (using appropriate runtime scaling for different workstations) that SPADE/8 is comparable in speed to TW7.0, about 1.8 times faster than QUAD, and 2.5 times slower than FD-98. In future work, we hope to obtain faster solutions by incorporating multilevel partitioning (we currently use a flat partitioner) and early-stop mechanisms, without sacrificing solution quality.

5 Conclusions

We presented several new PDP techniques used in our SPADE placer to rectify the drawbacks of classical PDP (mentioned in Sec. 1). The results obtained are overall significantly better than current state-of-the-art placement methods (10.8% better than QUAD [4], 10.6% better than TW7.0 [13], 15.8% better than FD-98 [16] and 15.3% better than `Snap-On` [18]). For all but two circuits in the MCNC benchmark suite, we have the best results so far. Our run-times are reasonable. We have thus demonstrated that PDP can be used effectively and efficiently to place large circuits. This is significant, since PDP has several advantages that can be exploited to meet the special needs of placement for complex deep sub-micron circuits: time efficiency, flexibility in accurately incorporating many optimization metrics like wirelength, power and timing, and flexibility in satisfying many constraints that are required for DSM chips (e.g., uniform thermal distribution, congestion control) without significantly degrading solution quality. Future work will investigate these aspects of cell placement.

Acknowledgements

We thank Bharat Gali for implementing SeqLP, and the anonymous referees for their comments, which helped improve the presentation.

References

[1] M.Breuer, "A Class of Min-cut Placement Algorithm," *Proc. 14th DAC*, pp. 284-290, 1977.

[2] A. Dunlop and B. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits," *IEEE Trans. CAD*, Vol. CAD-4, No. 1, pp. 92-98, Jan. 1985.

[3] P. Suaris and G. Kedem, "An Algorithm for Quadrisection and Its Application to Standard Cell Placement," *IEEE Trans. CAS*, Vol. 35, No. 3, pp. 294-303, Mar. 1988.

[4] D. J-H. Huang and A. B. Kahng, "Partitioning-based Standard-cell Global Placement with An Exact Objective," *Proc. ISPD*, pp. 18-25, 1997.

[5] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement," *IEEE Trans. on CAD*, pp. 1265-1278, Vol. 18, No. 9, Sept. 1999.

[6] C. M. Fiduccia and R. M. Mattheyses, "A Linear-time Heuristic for Improving Network Partitions," *Proc. 19th DAC*, pp. 175-181, 1982.

[7] S. Dutt and W. Deng, "A Probability-based Approach to VLSI Circuit Partitioning," *Proc. 33rd DAC*, pp. 100-105, 1996.

[8] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques," *Proc. ICCAD*, pp. 194-200, 1996.

[9] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey," *Integration, The VLSI Journal*, 19(1-2), pp. 1-81, 1995.

[10] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Application to VLSI Domain," *Proc. 34th DAC*, pp. 526-529, 1997.

[11] C. J. Alpert, D. J. Huang and A. B. Kahng, "Multilevel Circuit Partitioning," *Proc. 34th DAC*, pp. 530-533, 1997.

[12] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer, B.V., Deventer, The Netherlands, 1988.

[13] W-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Trans. CAD*, Mar. 1995.

[14] J.M. Kleinhans, G. Sigl, F.M. Johannes and K.J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. CAD*, vol. 10, no. 3, pp. 356-365, 1991.

[15] G. Sigl, K. Doll and F.M. Johannes, "Analytical placement: A Linear or Quadratic Objective Function?", *Proc. 28th DAC*, pp. 427-432, 1991.

[16] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," *Proc. 35th DAC*, pp. 269-274, 1998.

[17] G. Persky, "PRO: an Automatic String Placement Program for Polycell Layout", *Proc. 13th DAC*, pp. 417-423, 1976.

[18] X. Yang, M. Wang, K. Eguro and M. Sarrafzadeh, "A Snap-On Placement Tool", *Proc. ISPD-2000*, pp. 153-158.