Flow-based partitioning and position constraints in VLSI placement

Markus Struzyna
Research Institute for Discrete Mathematics, University of Bonn
Lennéstr. 2, 53113 Bonn, Germany
struzyna@or.uni-bonn.de

***Abstract*—This paper presents a new quadratic, partitioning-based placement algorithm which is able to handle non-convex and overlapping position constraints to subsets of cells, the *movebounds*. Our new flow-based partitioning (FBP) combines a global MinCostFlow model for computing directions with extremely fast and highly parallelizable local realization steps. Despite its global view, the size of the MinCostFlow instance is only linear in the number of partitioning regions and does not depend on the number of cells. We prove that our partitioning scheme finds a (fractional) solution for any given placement or decides in polynomial time that none exists. In practice, BonnPlace with FBP can place huge designs with almost 10 million cells and dozens of movebounds in 90 minutes of global placement. On instances with movebounds, the netlengths of our placements are more than 32% shorter than RQL's [25] and our tool is 9–20 times faster. Even without movebounds, the FBP improves the quality and runtime of BonnPlace significantly and our tool shows the currently best results on the latest placement benchmarks [16].**

I. Introduction

The traditional VLSI placement objective is to find positions of cells minimizing interconnecting wirelength in such a way that no cell overlaps any blockage or any other cell. This problem has been studied for several decades and received a boost again by the benchmarks [15], [16]. At the same time, the increasing design sizes, tight timing and wiring constraints make the placement an ongoing issue in chip design. These goals often cannot be met with classical density and blockage constraints in placement which apply to *all* cells at the same time.

Hence, there is a particular need to control the positions of *subsets* of cells in the design process efficiently: the movebounds. Movebounds appear in different chip design methodologies and applications: for particular timing and routability issues [18], for placement of different voltage domains [10] and to control clock domains [14] or IO-driven placement [28]. They can also be used as a compromise between flat and hierarchical design approaches [3]: movebounds allow to reveal the interior of hierarchical units (SoC, RLMs) but the overall hierarchical structure can be kept. Although movebounds are even part of the OpenAccess standard [20] there has not been any systematic work on movebounds in placement published until now to our best knowledge.

Unlike for the movebounded case, there are several approaches to the classical placement problem. Modern placement algorithms follow a MinCut strategy [2], [19] or an analytic approach. In the latter case one uses quadratic netlength minimization [5], [11], [21], [24], [25], [26] or approximates the half-perimeter wirelength (HPWL) by another smooth function [1], [8], [9], [13]. Overlap reduction is performed by artificial anchors and connections from repulsion forces [21], [24], [25], [26], from a precomputed cell-region matching [1], by explicit penalty methods [8], [9], [13], or by partitioning [5], [17], [27]. In this paper we present a completely new version of the global placement part of [5], the BonnPlace FBP. Our main contributions are:

- We introduce a generalized concept of inclusive and exclusive movebounds in placement. Cells associated to movebounds must be placed within the movebound area. Exclusive movebounds are in addition blockages to the other cells. Movebound areas can be non-convex and overlapping.
- We present a polynomial-time algorithm which checks whether a (fractional) placement with movebounds exists.
- We present a placement algorithm which handles several movebounds simultaneously.
- We present a new global partitioning algorithm which combines a novel global MinCostFlow model for computing directions with extremely fast and highly parallelizable local realization steps. This method is of its own interest even for placements without movebounds. The size of the MinCostFlow instance is linear in the number of windows and does not depend on the number of cells.
- We prove that our method guarantees a feasible (almost integral) partitioning if one exists, even in the presence of movebounds and for any given placement, unlike recursive partitioning approaches [5], [17], [27].
- We show how legalization of cells with different movebounds can be done simultaneously.
- We present the results of our tool on industrial instances with and without movebounds as well as on recent benchmarks.

This paper is organized as follows. In Section II we introduce the notation and formalize the concept of movebounds. Section III focuses on partitioning with movebounds. Section IV contains the new partitioning method. In Section V we present the experimental results and finally we conclude by the summary in Section VI.

978-3-9810801-7-9/DATE11/©2011 EDAA

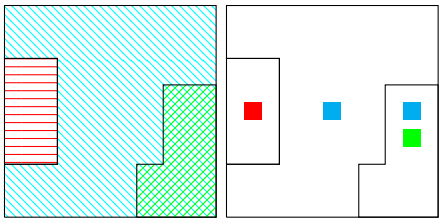

Figure 1. Three movebounds: an exclusive N, and two inclusive M,L. The area of L is contained in the area of M (left). The resulting three maximal regions (right).

II. Placement and Movebounds

Let $\mathcal{C}$ denote the set of rectangular cells and for each cell $c \in \mathcal{C}$ we write $(x, y)(c)$ for the cell's location. Given a placement $(x, y)(c)$ of c, $A_{(x,y)}(c)$ denotes the rectangle covered by the cell c and $\text{size}(c)$ its size. Given a finite set of rectangles A we refer to $\text{capa}(A)$ as the capacity of A, i.e. the total amount of space in A available to cells (respecting density and blockages). $\mathcal{A}$ is the chip area.

Definition 1. A *movebound* M is a pair $(A(M), \xi(M))$, where $A(M)$ is a finite set of axis-parallel, nonempty rectangles (the area) and $\xi(M) \in \{\text{exclusive}, \text{inclusive}\}$. For a set of movebounds $\mathcal{M}$ and a cell-movebound map $\mu : \mathcal{C} \to \mathcal{M}$, a placement $(x, y) : \mathcal{C} \to \mathcal{A}$ is legal w.r.t. movebounds if each cell c is entirely contained in $A(\mu(c))$ and for each exclusive movebound M only cells with $\mu(c) = M$ overlap $A(M)$.

One should also note that a placement without movebounds can be seen as a movebounded one, where $A(\mu(c)) = \mathcal{A}$ for any cell $c \in \mathcal{C}$. Moreover, we can assume that no exclusive movebound overlaps any other movebound as such situations can easily be detected and modified at the input. Thus, a placement is legal w.r.t. movebounds if and only if $\forall c \in \mathcal{C} : A_{(x,y)}(c) \subset \bigcup A(\mu(c))$. A necessary condition for feasibility with movebounds is hence:

$$\sum_{c \in \mathcal{C}: \mu(c) \in \mathcal{M}'} \text{size}(c) \leq \text{capa}\Big(\bigcup_{M \in \mathcal{M}'} A(M) \Big) \; \forall \mathcal{M}' \subset \mathcal{M} \quad (1)$$

As it is inconvenient to evaluate all the subsets, we partition $\mathcal{A}$ into regions (see Figure 1) :

Definition 2. A region r is a set of axis-parallel nonempty rectangles where (a) the rectangles in r do not overlap and (b) $\forall M \in \mathcal{M}$: if the area of r overlaps the area $A(M)$ then the area of r is contained in the area of $A(M)$. A movebound M *covers* r if the area of r is contained in the area of $A(M)$.

In an instance with overlapping movebounds, there might be $2^{|\mathcal{M}|}$ different intersections and potentially exponentially many regions. For the minimum number of regions one can obtain better bounds:

Lemma 1. Let l be the number of rectangles encoding $\mathcal{M}$ in $\mathcal{A}$. Then there exists a decomposition of $\mathcal{A}$ into regions with at most $\mathcal{O}(l^2)$ rectangles.

Proof. Consider the Hanan grid obtained by coordinates from rectangles in $\mathcal{M}$. The grid decomposes $\mathcal{A}$ into $\mathcal{O}(l^2)$ rectangles, which can serve as regions. □

Let $\mathcal{R}$ be a set of regions. Define the MaxFlow instance (G, u, s, t) with graph $G = (V, E)$ and $V := \{s, t\} \dot\cup \mathcal{C} \dot\cup \mathcal{R}$ and $E := (\{s\} \times \mathcal{C}) \dot\cup \{(c, r) \in \mathcal{C} \times \mathcal{R} | \mu(c) \text{ covers } r\} \dot\cup (R \times \{t\})$ and capacities $u((s, c)) := \text{size}(c) \; \forall c \in \mathcal{C}$, $u(r, t) := \text{capa}(r) \forall r \in \mathcal{R}$ and $u \equiv \infty$ elsewhere. Then, one obtains:

Theorem 1. The maximum flow in (G, u, s, t) has value $val(f) = \text{size}(\mathcal{C}) := \sum_{c \in \mathcal{C}} \text{size}(c)$ if and only if (1) holds.

Proof. (Sketch) $\Rightarrow$: Pick a subset $\mathcal{M}' \subset \mathcal{M}$, check flow condition. $\Leftarrow$: By contradiction using the MaxFlowMinCut theorem. For details see [22]. □

For a simple feasibility check one can do faster:

Theorem 2. One can decide in $\mathcal{O}(|\mathcal{C}| + |\mathcal{M}|^2 |\mathcal{R}|)$ time whether a (fractional) placement with movebounds exists.

Proof. Recall G from above. Cluster all nodes in $\mathcal{C}$ of the same movebound to one node. Using $|\mathcal{M}| \leq |\mathcal{R}|$ one can solve the (clustered) MaxFlow instance in Theorem 1 in $\mathcal{O}(|\mathcal{M}|^2 |\mathcal{R}|)$ time [7]. □

In particular, this is polynomial in the input. For the remaining part of the paper we assume that the instance is feasible.

III. Partitioning with Movebounds

Partitioning-based analytical placement tools [5], [17], [27] subdivide $\mathcal{A}$ by regular grids into sets of rectangular *windows* and subsequently apply quadratic netlength minimization (QP) and partitioning. For a set $\mathcal{W}$ of windows, partitioning is a map $\rho : \mathcal{C} \to \mathcal{W}$, such that $\sum_{c \in \mathcal{C}: \rho(c) = w} \text{size}(c) \leq \text{capa}(w) \; \forall w \in \mathcal{W}$ and $\sum_{c \in \mathcal{C}} \text{cost}(c, \rho(c))$ is minimized (usually, $\text{cost}(c, w) := \text{dist}_{L_1}(c, w)$). The procedure is repeated recursively until the window sizes are small enough. Indeed, it can be shown that an almost integral partitioning (with at most $|\mathcal{W}| - 1$ fractionally assigned cells) can be computed efficiently [4]. To improve placement quality, revisions of the partitioning are allowed by considering neighboring 2×2 or 3×3 windows (Reflow [17], Repartitioning [5], [27]). Without movebounds, the common invariant of [5], [17], [27] is that after partitioning no window contains more cell area than allowed by the window's capacity. For a feasible partitioning with movebounds, this condition translates to the requirement that (1) has to be satisfied in each window $w \in \mathcal{W}$. To this end, the idea behind Theorem 1 can be applied. For each window $w \in \mathcal{W}$, let $\mathcal{R}_w$ be a set of regions in w. Instead of computing $\rho : \mathcal{C} \to \mathcal{W}$, one computes $\rho' : \mathcal{C} \to \bigcup_{w \in \mathcal{W}} \mathcal{R}_w$ first. This can be done with the transportation algorithm [5] and modified costs: $\text{cost}((c, r)) := \infty$, wherever $\mu(c)$ does not cover r and $\text{cost}((c, r)) := \text{dist}(c, r)$ elsewhere. The partitioning can then be done in $\mathcal{O}\big(q^2 n(\log(n) + q \log(q))\big)$ time, where $q = \sum_{w \in \mathcal{W}} |\mathcal{R}_w|$. This method can already be used for recursive partitioning with movebounds as in the unconstrained case but one can do better – see Section IV.

The partitioning is also useful for legalization. While for non-overlapping movebounds legalization can be done independently, it is more complicated for the overlapping case: movement control has to be done for each movebound. Here, regions come into play again. We decompose $\mathcal{A}$ into regions $\mathcal{R}$, compute a partitioning $\rho : \mathcal{C} \to \mathcal{R}$ and for each $r \in \mathcal{R}$ legalize $\{c : \rho(c) \in r\}$ in $A(r)$ using [6]. Hence, legalization of cells (even with different movebounds) in one region is done simultaneously.

IV. Flow-Based Partitioning

Recursive partitioning approaches [5], [17], [27] have several drawbacks. For a window w, partitioning into subwindows of w assumes that a feasible partitioning in w exists, which is not always true due to rounding effects in partitioning or increased cell sizes from congestion avoidance [5]. Incremental placements are often impossible without restarting from scratch. Moreover, partitioning decisions in a window are taken locally, independently from other windows and are sensitive to slightly modified cell positions. The time-consuming reflow steps can only compensate these problems partially.

Our novel partitioning method consists of two major steps: a new global MinCostFlow model and a new local realization. The directions and the amount of movement are computed first. The realization is then done in topological order of the flow edges by local QP and Partitioning steps. For *any* initial placement of some feasible placement instance our method guarantees a feasible (fractional) partitioning.

A. MinCostFlow Model

Given a set of windows $\mathcal{W}$ induced by some regular grid Γ and a set of cells $\mathcal{C}$ with movebounds $\mathcal{M}$, we ask for a feasible partitioning $\rho : \mathcal{C} \to \mathcal{W}$ with movebounds. We assume that each cell is assigned to some window $w \in \mathcal{W}$ (from incremental placement, previous partitioning or QP). Let $\mathcal{C}_{Mw}$ denote the set of cells with movebound M in window w. In each $w \in \mathcal{W}$, there are three types of nodes:

- cell groups: $\mathcal{C}_{Mw}$ $\forall M \in \mathcal{M}$,
- transit: $\mathcal{T}_{Mw} = \{t^x_{Mw} | x \in \{N, E, S, W\}\} \forall M \in \mathcal{M}$,
- regions: r for each $r \in \mathcal{R}_w$.

We may restrict cell group and transit nodes of a movebound M to windows intersecting $A(M)$'s bounding box [22] and omit empty sets. The nodes are embedded in the plane: $(x, y)(\mathcal{C}_{Mw})$ is set to the center-of-gravity of cells in $\mathcal{C}_{Mw}$, transit nodes t^N_{Mw}, $x \in \{N, E, S, W\}$ are put to the center of the north, east, south, and west boundary of w and finally $(x, y)(r)$ is set to the center-of-gravity of the free area of the region r. We set $b(\mathcal{C}_{Mw}) := \sum_{c \in \mathcal{C}_{Mw}} \text{size}(c)$ (supply), $b(r) := -\text{capa}(r)$ (demand) and $b(\mathcal{T}_{Mw}) \equiv 0$ (transit). There are four types of edges in each window $w \in \mathcal{W}$:

$E^{cr}_{Mw} := \{(\mathcal{C}_{Mw}, r) | M \text{ covers } r \in \mathcal{R}_w\}$
$E^{tt}_{Mw} := \{(t^x_{Mw}, t^y_{Mw}) | x, y \in \{N, E, S, W\}, x \neq y\}$
$E^{tr}_{Mw} := \{(t^x_{Mw}, r) | x \in \{N, E, S, W\}, r \in \mathcal{R}_w, M \text{ covers } r\}$
$E^{ct}_{Mw} := \{(\mathcal{C}_{Mw}, t) | t \in \mathcal{T}_{Mw}\}$

See also Figure 2. The windows are connected at neighboring transit nodes by arcs connecting transit nodes of the same movebound, see Figure 3:

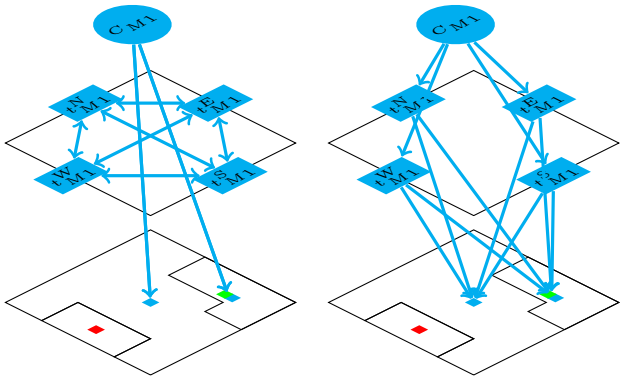

Figure 2. Example for movebound M in window 1. Edge sets are separated for better readability: E^{cr}_{M1}, E^{tt}_{M1} (left) and E^{ct}_{M1}, E^{tr}_{M1} (right).

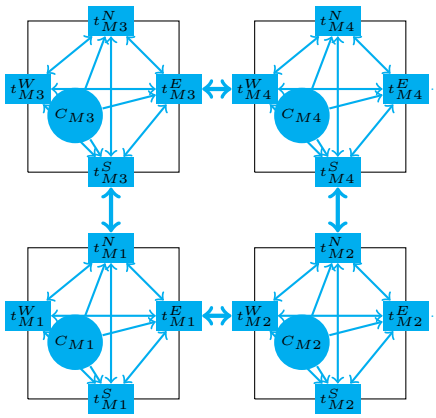

Figure 3. Example containing external edges: Subgraphs of different windows for the movebound M connected by edges between transit nodes. Region nodes are omitted here.

$$E^{\text{ext}}_{v,w,M} := \begin{cases} \{(z^S_{Mv}, z^N_{Mw}), (z^N_{Mw}, z^S_{Mv})\} & \text{if } w \text{ is below } v \\ \{(z^N_{Mv}, z^S_{Mw}), (z^S_{Mw}, z^N_{Mv})\} & \text{if } v \text{ is below } w \\ \{(z^W_{Mv}, z^E_{Mw}), (z^E_{Mw}, z^W_{Mv})\} & \text{if } v \text{ is right of } w \\ \{(z^E_{Mv}, z^W_{Mw}), (z^W_{Mw}, z^E_{Mv})\} & \text{if } w \text{ is right of } v \\ \emptyset & \text{otherwise.} \end{cases}$$

For $e \in E^{\text{ext}}_{v,w,M}$ we set $\text{cost}(e) := 0$. For $e = (u, v) \in E^{cr} \cup E^{ct} \cup E^{tt} \cup E^{tr}$, $\text{cost}(e) := \text{dist}(u, v)$. All edges are uncapacitated. The MinCostFlow instance is then (G, b, cost) with

$$V(G) := \bigcup_{w \in \mathcal{W}} \Big(\bigcup_{M \in \mathcal{M}} (\mathcal{C}_{Mw} \cup \mathcal{T}_{Mw}) \cup \mathcal{R}_w \Big) \text{ and}$$

$$E(G) := \bigcup_{w \in \mathcal{W}, M \in \mathcal{M}} (E^{cr}_{Mw} \cup E^{ct}_{Mw} \cup E^{tr}_{Mw} \cup E^{tt}_{Mw}) \cup \bigcup_{v,w \in \mathcal{W}, M \in \mathcal{M}} E^{\text{ext}}_{v,w,M}.$$

As movebounds are allowed to overlap, there might be $\mathcal{O}(|\mathcal{M}|)$ many copies of the graph as in Figure 3. In practice, $|V|$ and $|E|$ are linear in $|\mathcal{R}| + |\mathcal{W}|$ (cf. Table 1), unlike in [1], where the MinCostFlow instance is quadratic in $|\mathcal{W}|$ in the worst case. On the other hand, a feasible solution is always guaranteed:

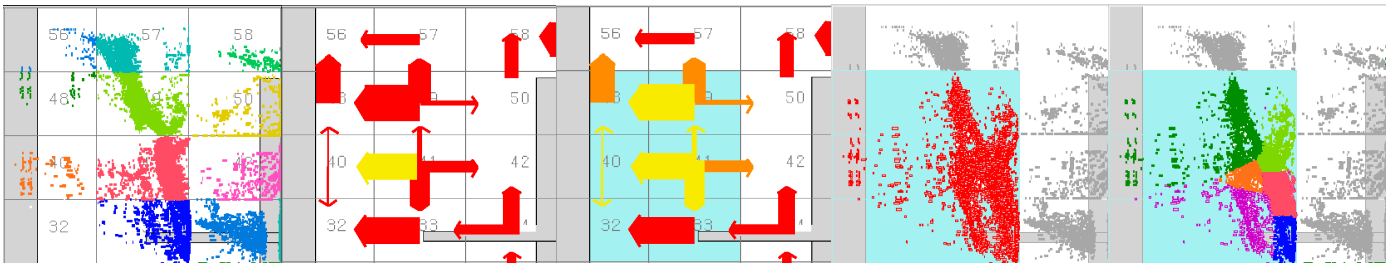

Figure 4. Realization Steps (1) Initial Solution. (2) Choose an external edge e from the flow graph (3) choose window W covering e and push flow along all other possible edges in W (4) local QP in W with fixed cells outside W (5) partitioning in W and a new solution in W.

Theorem 3 The MinCostFlow instance (G, b, cost) is feasible iff a (fractional) placement with movebounds exists.

Proof. Adding two new nodes $\{s, t\}$ and edges $e = (s, \mathcal{C}_{Mw})$ with $u(e) := b(\mathcal{C}_{Mw})$ for all $M \in \mathcal{M}, w \in \mathcal{W}$ and $e' = (r, t)$ with $u(e') = -b(r)$ for all $r \in \mathcal{R}$ to G as well as replacing any path from $\mathcal{C}_{Mw}$ to some r covered by M by an edge $(\mathcal{C}_{Mw}, r)$ leads to a MaxFlow instance as in Theorem 1. □

B. Realization

Given a solution f of the MinCostFlow instance (G, b, cost), we consider the flow-carrying graph G_f of G by deleting all edges with zero flow from G. Now, if there is no flow-carrying edge $e \in E^{\text{ext}}_{v,w,M}$ in G_f, all flow can be absorbed in the corresponding windows, thus (1) holds in each $w \in \mathcal{W}$ and the partitioning is feasible. Otherwise, for an external flow-carrying edge e cells of total size f_e which belong to movebound M have to be moved from v to w. It can be shown that proceeding in topological order of G_f, one can indeed focus on external edges only and delete them after realization [22].

We now describe *which* cells are shipped from window v to w. Here, two other novel ideas are applied. First, we do not look at cells in v only, but we consider coarser windows W with $\{v, w\} \subset W \subset \mathcal{W}$ containing v and its neighboring windows. Usually W consists of 2×3 or 3×2 windows in $\mathcal{W}$. Second, the partitioning is not based on a simple movement penalty, but a local QP (considering all cells outside W as fixed) will be computed first to obtain more connectivity information. Then a partitioning step with movebounds as in Section III is computed in W using the modified transportation algorithm. In the transportation algorithm we consider all cells in W and partition them among all regions and temporary regions resulting from transit nodes in W. The capacity of each transit/region node is set to:

$$\text{capa}(z) := \sum_{e=(\cdot,z)\in E(G_f):e \text{ realized}} f_e - \sum_{e=(z,\cdot)\in E(G_f):e \text{ realized}} f_e \quad (2)$$

and corresponds to the flow excess at this stage. It can be shown [22] that this procedure guarantees a feasible (fractional) partitioning in W, and thus in $\mathcal{W}$ using induction. The use of transit nodes as buffer regions is mandatory because a window w can appear several times during realization. Each time a window w appears in realization, the partitioning of cells in w is recomputed. Figure 4 shows a single realization example without movebounds.

For two external edges e, e' without external predecessors, the realization of e and e' can be done independently, if the coarse realization windows W of e and W' of e' do not overlap. This allows parallel processing. Using an efficient scheme, we can guarantee deterministic behavior and achieve good parallel speed-ups (up to 7.9 with 8 CPUs) on large grids [22].

V. Experimental Results

We performed two sets of experiments to evaluate the performance of our new tool on instances with and without movebounds. The first experiment set was done on chips from industry with and without movebounds on an Intel Xeon X5365 with 8 CPUs, 3.0 Ghz and 64 GB memory. Table I shows the sizes and the runtime of our new partitioning method on the largest instance (in terms of $|V|$ and $|E|$) in our testbed. Here we also provide the runtimes for the MinCostFlow computation and realization which show that even partitionings of 2.5 million cells to fifty thousand windows can be done within 15 seconds. The MinCostFlow was computed by a (sequential) NetworkSimplex algorithm, the realization was performed in parallel. On finer grids, the runtime for the MinCostFlow computation increases, but even instances with 300 000 windows can be processed quickly.

We then compared our BonnPlace FBP to the industrial version of RQL [25], a state-of-the-art force-directed tool. Both tools were allowed to run in parallel, our tool used sequential legalization [6]. Both tools used *BestChoice* [17] for clustering with cluster ratio 5. To omit different density interpretations, the target density was set to 97% for either tool. We provide the HPWL of legal placements and wall-clock runtimes. Table II shows the results without movebounds. Both tools show comparable results but BonnPlace FBP is more than 5.5 times faster on average. The differences on particular chips are however remarkable.

Instances with movebounds are summarized in Table III, in which besides the number of cells $|\mathcal{C}|$ and movebounds $|\mathcal{M}|$ the percentage of cells with movebounds and the maximum density of cells in a movebound are shown. In the last column we indicate whether the movebounds overlap (O), and (F) means that movebounds were obtained from flattening hierarchy. Tomoku, Trips and Andre had nested, overlapping movebounds which are infeasible in the exclusive case. Table

TABLE I: Sizes and runtimes of the Flow-based partitioning instances from Erhard (2 578 246 cells, 43 movebounds)

$\lvert V\rvert$ ($\times 1000$)	$\lvert E\rvert$ ($\times 1000$)	$\frac{\lvert E\rvert}{\lvert V\rvert}$	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{R}\rvert$	FBP Algorithm wall clock runtimes Flow-computation	Flow-realization
2	15	5.5	16	94	0:00:00	0:00:26
7	38	4.9	64	206	0:00:00	0:00:11
22	102	4.6	256	520	0:00:00	0:00:06
55	238	4.3	1 536	2 118	0:00:00	0:00:06
165	668	4.0	9 216	9 464	0:00:01	0:00:04
930	3 685	4.0	55 296	45 643	0:00:08	0:00:07
5 198	20 365	3.9	331 776	239 885	0:01:20	0:00:11

TABLE II: Results on instances without movebounds

Chip	$\lvert\mathcal{C}\rvert$ (10^3)	Industrial RQL [25] HPWL	Industrial RQL [25] hh:mm:ss	BonnPlace FBP HPWL	BonnPlace FBP hh:mm:ss
Dagmar	50	0.95	0:02:13	0.80	0:00:40
		100.0%	1.0	83.2%	3.3
Elisa	67	2.60	0:02:35	2.86	0:00:36
		100.0%	1.0	109.8%	4.4
Lucius	77	3.42	0:03:30	3.73	0:01:48
		100.0%	1.0	109.2%	1.9
Felix	87	8.17	0:03:05	7.68	0:00:36
		100.0%	1.0	94.0%	5.2
Paula	129	3.14	0:06:30	3.21	0:01:41
		100.0%	1.0	102.3%	3.9
Rabe	175	12.42	0:08:09	12.36	0:01:44
		100.0%	1.0	99.6%	4.7
Julia	190	10.65	0:08:39	10.84	0:02:15
		100.0%	1.0	101.8%	3.9
Max	328	17.22	0:18:00	17.99	0:06:25
		100.0%	1.0	104.5%	2.8
Roger	456	27.42	0:22:21	27.74	0:10:37
		100.0%	1.0	101.2%	2.1
Ashraf	867	61.05	0:49:03	61.53	0:09:54
		100.0%	1.0	100.8%	5.0
Patrick	1052	43.84	1:00:13	44.64	0:12:24
		100.0%	1.0	101.8%	4.9
Erhard	2578	463.76	2:24:24	413.64	0:32:45
		100.0%	1.0	89.2%	4.4
Arijan	3753	485.04	3:30:35	484.29	0:59:36
		100.0%	1.0	99.8%	3.5
Philipp	3946	358.91	4:22:42	342.37	0:54:52
		100.0%	1.0	95.4%	4.8
Tomoku	5296	356.44	7:51:32	354.37	1:10:10
		100.0%	1.0	99.4%	6.7
Trips[23]	5747	616.05	6:31:12	589.69	1:25:13
		100.0%	1.0	95.7%	4.6
Valentin	5838	671.49	8:15:01	610.40	1:36:21
		100.0%	1.0	90.9%	5.1
Andre	6794	437.01	9:16:52	448.74	1:38:04
		100.0%	1.0	102.7%	5.7
Ludwig	7500	598.40	9:19:11	603.38	1:30:30
		100.0%	1.0	100.8%	6.2
Leyla	8472	711.90	13:43:02	718.22	2:09:18
		100.0%	1.0	100.9%	6.4
Erik	9316	559.34	13:25:50	547.43	2:07:02
		100.0%	1.0	97.9%	6.3
Total		100.0%	81:44:09	99.3%	14:52:29
			1.0		5.5

TABLE III: Industrial instances with movebounds

Chip	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{C}\rvert$	% cells w/ moveb.	max mb. dens	remarks
Rabe	2	175 646	4.3%	67%	
Ashraf	206	866 777	22.0%	92%	(F)
Erhard	43	2 578 246	97.8%	74%	
Tomoku	85	5 296 120	1.2%	74%	(O)(F)
Trips[23]	114	5 747 007	99.4%	81%	(O)
Andre	43	6 794 323	3.8%	73%	(O)(F)
Ludwig	33	7 500 446	2.7%	70%	(O)(F)
Erik	39	9 316 938	84.6%	85%	(F)

TABLE IV: Results with inclusive movebounds

Chip	Industrial RQL [25] HPWL	Industrial RQL [25] hh:mm:ss	Industrial RQL [25] viol.	BonnPlace FBP HPWL	BonnPlace FBP hh:mm:ss
Rabe	16.68	00:10:22		12.44	0:01:58
	100.0%	1.0		74.6%	5.3
Ashraf	crashed			61.59	0:34:12
Erhard	549.62	04:49:16		499.31	0:39:11
	100.0%	1.0		90.8%	7.4
Tomoku	737.34	13:20:44	361	367.00	1:32:13
	100.0%	1.0		49.8%	8.7
Trips[23]	703.88	11:37:42		611.58	2:27:59
	100.0%	1.0		86.9%	4.7
Andre	1023.65	37:36:58	62	462.23	3:08:50
	100.0%	1.0		45.2%	12.0
Ludwig	1207.63	26:22:04		624.09	2:53:31
	100.0%	1.0		51.7%	9.1
Erik	879.12	36:44:33	4655	598.15	2:55:50
	100.0%	1.0		68.0%	12.5
Total w/o Ashraf	100.0%	130:41:47		64.5%	13:38:31
		1.0			9.6

TABLE V: Results with exclusive movebounds

Chip	Industrial RQL [25] HPWL	Industrial RQL [25] hh:mm:ss	Industrial RQL [25] viol.	BonnPlace FBP HPWL	BonnPlace FBP hh:mm:ss
Rabe	16.73	00:11:18		12.85	0:01:41
	100.0%	1.0		76.8%	6.7
Ashraf	93.21	02:09:14		64.44	0:15:17
	100.0%	1.0		69.1%	8.5
Erhard	633.64	04:25:35		518.64	0:30:01
	100.0%	1.0		81.9%	8.8
Andre	1144.21	47:08:58	463	494.04	1:41:37
	100.0%	1.0		43.2%	27.8
Erik	864.74	*43:20:47	5630	624.82	2:10:34
	100.0%	1.0		72.3%	19.9
Total	100.0%	97:15:52		67.1%	04:39:10
		1.0			20.9

TABLE VI BonnPlace FBP with inclusive movebounds

Chip	Wall-clock runtimes (hh:mm:ss) Global Pl.	Legalization	Total	Global/Total
Rabe	0:01:20	0:00:38	0:01:58	67.8 %
Ashraf	0:09:32	0:24:40	0:34:12	27.9 %
Erhard	0:24:52	0:14:19	0:39:11	63.5 %
Tomoku	0:47:44	0:44:29	1:32:12	51.8 %
Trips	1:14:51	1:13:08	2:27:59	50.6 %
Andre	1:35:41	1:33:09	3:08:50	50.7 %
Ludwig	1:12:23	1:41:08	2:53:31	41.7 %
Erik	1:30:24	1:25:26	2:55:50	49.2 %
Total	6:56:47	7:16:57	14:13:43	48.8%

TABLE VII: Results on ISPD 2006 Benchmarks

	Kraftwerk2 HPWL (H)	Kraftwerk2 H+DENS (H+D)	Kraftwerk2 H+D+CPU (H+D+C)	BonnPlace FBP HPWL (H)	DENS (D)	Runtime wall time (sec)	(C)	(H+D)	(H+C+D)	ratio (H+D)	ratio (H+D+C)
ad5	433.84	449.48	407.46	430.43	1.81 %	1571	-9.52%	438.22	396.50	97.5 %	97.3%
nb1	65.92	66.22	60.67	69.05	2.04 %	378	*-10.00%*	70.46	63.42	106.4 %	104.5%
nb2	203.91	206.53	185.88	200.77	1.92 %	1076	-8.16%	204.62	187.92	99.1 %	101.1%
nb3	278.51	279.57	251.62	273.48	1.15 %	709	-8.25%	276.63	253.82	98.9 %	100.8%
nb4	304.24	309.44	282.74	313.72	2.27 %	1124	*-10.00%*	320.83	288.75	103.7 %	100.2%
nb5	548.38	563.15	509.65	545.82	1.31 %	2030	*-10.00%*	552.96	497.66	98.2 %	97.7%
nb6	528.59	537.59	484.42	520.19	1.42 %	2800	-9.42%	527.59	477.62	98.1 %	98.7%
nb7	1126.58	1162.12	1056.84	1075.98	0.97 %	5461	-8.35%	1086.40	995.70	93.5 %	94.2%
										99.4 %	99.5%

IV and V show the comparison between RQL and BonnPlace FBP tool. Though RQL is able to respect movebounds in general, it produced several movebound violations (viol.) and crashed on Ashraf in the inclusive case. Our tool produced legal placements on each design. The run on Erik with exclusive movebounds (*) was done in sequential mode as the parallel one failed. Comparing the results, our tool performs better on each design, in both HPWL and runtime. On inclusive movebound instances, our placements are more than 35% shorter on average, on Tomoku and Andre even by 50%. Our tool is more than 9.5 times faster. With exclusive movebounds the gap is 32% in favor of our tool and we are more than 20 times faster. Table VI shows that our new global placement is fast and takes about 50% of the total placement runtime on movebounded instances.

In the second experiment set we ran our tool on ISPD 2006 benchmarks [16] in standard mode (using *BestChoice* with cluster ratio 2) on an Opteron 852 machine. The same machine type has been used for the contest runs. Our tool ran in parallel mode and used up to 8 CPUs in global placement, followed by sequential legalization [6]. We compare the results to the currently best tool on this benchmark set (Kraftwerk2, taken from [21]) in Table VII. For both tools, we provide the HPWL (H) with the density penalty (D) and the CPU penalty/bonus (C). The CPU bonus in italics was truncated at -10%, as in [16]: (without we would have obtained $-10.9\%, -10.3\%$ and -10.7% for nb1, nb4 and nb5 resp). On the mixed-size instance nb1, our placer produces inferior results as the large blocks are fixed at an early stage in placement.

In total, we are able to improve the currently best results on ISPD 2006 benchmarks [21], [16] significantly. Our tool benefits from its high parallelization ability during global placement, which allows speedups of up to 4.5 with 8 CPUs for the global part and preserves deterministic behavior [22]. The sequential legalization [6] leads to overall speedups between 1.5 - 3.5 with 8 CPUs and leaves room for further acceleration.

VI. Summary and Conclusions

In this paper we presented for the first time a non-trivial approach to handle non-convex and overlapping position constraints in placement, the movebounds. Our BonnPlace FBP is able to handle simultaneously dozens of movebounds and millions of cells in an efficient way and seems much more suitable to instances with movebounds than the previous tools such as RQL [25]. On instances with movebounds, our tool performs much better in terms of wirelength and runtime. The new core routine of our placer, the flow-based partitioning, compensates several drawbacks of previous partitioning algorithms and leads to highly competitive placements even on unbounded instances. BonnPlace FBP computes the currently best results on ISPD2006 benchmarks [16].

VII. Acknowledgements

I would like to thank Prof. Jens Vygen, Ulrich Brenner and the anonymous reviewers for their remarks and comments.

References

[1] Agnihotri, A.R., and Madden, P.H., Fast Analytic Placement using Minimum Cost Flow. Proc. of the ASPDAC (2007), 128–134.

[2] Agnihotri, A.R., Ono, S., and Madden, P.H., Recursive bisection placement: feng shui 5.0 implementation details. Proc. of the ISPD (2005), 230–232.

[3] Bednar, T.R., Buffet, P.H., Darden, R.J., Gould, S.W. and Zuchowski, P.S., Issues and Strategies for the physical design of system-on-chip ASICs. IBM Journal of Research and Developement 46 (6) (2002), 661–673.

[4] Brenner, U., A faster polynomial algorithm for the unbalanced Hitchcock transportation problem. Operations Research Letters 36(4) (2008), 408–413.

[5] Brenner, U., Struzyna, M., and Vygen, J., BonnPlace: Placement of leading-edge chips by advanced combinatorial algorithms. IEEE TCAD 27 (2008), 1607–1620.

[6] Brenner, U., and Vygen J., Legalizing a placement with minimum total movement. IEEE TCAD 23 (2004), 1597-1613.

[7] Gusfield, D., Martel, C. and Fernández-Baca, C., Fast algorithms for bipartite network flow. SIAM Journal on Computing 16 (2) (1987) 237–251.

[8] Chan, T.F., Cong, J., Shinnerl, J.R., Sze, K., and Xie, M., mPL6: enhanced multilevel mixed-size placement. Proc. of the ISPD (2006), 212–214.

[9] Chen, T.C., Jiang, Z.W., Hsu, T.C, Chen, H.C. and Chang, Y.W., NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. IEEE TCAD 27 (7) (2008), 1228–1240.

[10] Hu, J., Shin, Y., Dhanwada, N.R. and Marculescu, R., Architecting voltage islands in core-based system-on-a-chip designs. International Symposium on Low Power Electronics and Design (2004), 180–185.

[11] Hu, B., Zeng, Y., and Marek-Sadowska, M., mFAR: fixed-points-addition-based VLSI placement algorithm. In Proc. of the ISPD (2005), 239–241.

[12] Khatkhate, A., Li, C., Agnihotri, A. R., Yildiz, M. C., Ono, S., Koh, C., and Madden, P. H., Recursive bisection based mixed block placement. Proc. of the ISPD (2004), 84–89.

[13] Kahng, A.B and Wang, Q., A faster implementation of APlace. In Proc. of the ISPD (2006), 218–220.

[14] Nam, G.J. and Villarubia, P.G. [2009] Placement: Introduction/Problem Formulation. in Handbook of Algorithms for Physical Design Automation (2009), 277–287.

[15] Nam, G.-J., Alpert, C.J., Villarubbia, P., Winter, B. and Yildiz, M., The ISPD 2005 Placement Contest and Benchmark Suite. *http://www.sigda.org/ispd2005/contest.htm*

[16] Nam, G.-J.: The ISPD 2006 Placement Contest and Benchmark Suite. *http://www.sigda.org/ispd2006/contest.html*

[17] Nam, G.-J., Reda, S. Alpert, C.J., Villarrubia, P.G. and Kahng, A.B., A fast hierarchical quadratic placement algorithm. IEEE TCAD 25(4) (2006) 678–691.

[18] Ossler, P.J., Placement Driven Synthesis Case Studies on Two Sets of Two chips: Hierarchical and Flat. Proc. of the ISPD(2004), 190–196.

[19] Roy, J.A., Papa, D.A., Adya, S.N., Chan, H.H., Ng, A.N., Lu, J.F., and Markov, I.L., Capo: robust and scalable open-source min-cut floorplacer. Proc. of the ISPD (2005). 224–226.

[20] Si2- Silicon Intgration Initiative, The Chip Hierarchical Design System Technical Data Standard. Silicon Intgration Initiative (2009), *http://www.si2.org* .

[21] Spindler, P., Schlichtmann, U. and Johannes F.M., Kraftwerk2 - A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model. IEEE TCAD 27(8) (2008): 1398–1411.

[22] Struzyna, M., Flow-based Partitioning and Fast Global Placement in Chip Design, PhD thesis, University of Bonn (2010).

[23] *http://userweb.cs.utexas.edu/users/cart/trips/index.html*

[24] Viswanathan, N.,Pan, M. and Chu, C.C-N., FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control. Proc. of the ASPDAC, (2007), 135–140.

[25] Viswanathan, N., Nam, G., Alpert, C. J., Villarrubia, P., Ren, H., and Chu, C., "RQL: global placement via relaxed quadratic spreading and linearization". Proc. of the DAC (2007), 453–458.

[26] Vorwerk, K., Kennings, A., and Vannelli, A., Engineering details of a stable force-directed placer. In Proc. of the ICCAD (2004), 573–580.

[27] Vygen, J.: Algorithms for large-scale flat placement. Proc. of the DAC (1997), 746–751.

[28] Xiong, J., Wong, Y.-C., Sarto, E. and He, L., Constraint driven I/O planning and placement for chip-package co-design. In Proc. of APSDAC (2006), 207–212.