Algorithms and Data Structures for Fast and Good VLSI Routing

Michael Gester, Dirk Müller, Tim Nieberg, Christian Panten, Christian Schulte, Jens Vygen*

ABSTRACT

We present advanced data structures and algorithms for fast and high-quality global and detailed routing in modern technologies. Global routing is based on a combinatorial approximation scheme for min-max resource sharing. Detailed routing uses exact shortest path algorithms, based on a shape-based data structure for pin access and a two-level track-based data structure for long-distance connections. All algorithms are very fast. We demonstrate their superiority over traditional approaches by a comparison to an industrial router (on 32 nm and 22 nm chips). Our router is over two times faster, has 5 % less netlength, 20 % less vias, and reduces detours by more than 90 %.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI (very large scale integration)*; B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*

General Terms

Algorithms

Keywords

VLSI Design, Global Routing, Detailed Routing, Routing Optimization

1. INTRODUCTION

We restrict ourselves to *Manhattan routing*, meaning that all wires run parallel to the x- or y-axis. On a single routing layer, either almost all wires are horizontal or almost all are vertical (this is called the *preferred direction* of a layer; wires running orthogonally are called *jogs*). Horizontal and vertical layers alternate. This still is common design practice today.

Routing the most complex chips in the current technologies poses several challenges (cf. also [1]):

- Finding any feasible solution is often very difficult.
- The design rules tend to become more complicated with each new technology generation. While *diff-net rules*, requiring a certain minimum distance of wires that belong to different nets, are most important, *same-net rules* also require more and more attention. Ideally, the computed routing should pass all design rule checks (DRC).
- Pins often have irregular geometries and many blockages around them, which makes pin access a challenging packing problem in itself.
- Mostly for timing reasons, many nets require nonstandard wire widths, increased wire spacing, or restriction to a subset of routing layers.
- The quality of the overall routing solution is extremely important. Large detours often lead to violated timing constraints. The manufacturing yield and power consumption is significantly influenced by the routing.
- Huge instance sizes contrast with expected turn-around times of just a few hours. This requires extremely efficient algorithms and data structures.

1.1 Outline

Our global router, which we describe in Section 2, is based on an efficient fully polynomial approximation scheme for the min-max resource sharing problem. It produces a provably near-optimal fractional solution, rounds it, and removes the local congestion caused by rounding.

For detailed routing, we pre-compute routing tracks that we will use for the majority of the wires in order to pack them efficiently. The available routing space is modeled by an efficient two-level data structure (called fast grid and shape grid). A specialized version of Dijkstra's algorithm exploits this data structure to find shortest paths extremely fast even for long distances. Off-track paths are computed by a shape-based shortest path algorithm to access pins, also taking same-net rules into account. We present these routing algorithms and data structures in Section 3, and explain how we combine them with an industrial router for DRC cleanup.

The experimental results in Section 4 show that competitive numbers of DRC violations are obtained by this approach, while improving considerably wire length, via count, detours, and overall runtime.

*The authors are with the Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, D-53113 Bonn (e-mail: {gester,mueller,nieberg,panten,schulte,vygen}@or.uni-bonn.de).

DAC 2012, June 3-7,2012, San Francisco, California, USA.

2. GLOBAL ROUTING

2.1 Modeling the Global Routing Problem

Global routing is an abstraction of the actual routing problem to a coarser model of the routing space. The chip area is divided into an array of *tiles*. The size of our tiles is chosen such that approximately 50 to 100 parallel wires (of minimum width) would fit into one tile on each layer. For each tile and each routing layer we have a vertex. The vertices thus constitute a partition of the routing space. Two vertices (t, l) and (t', l') are connected by an edge if $t = t'$ and $|l - l'| = 1$, or if $l = l'$ and t and t' are two tiles that are adjacent in the preferred direction of routing layer l. This defines an undirected graph G. The edges of G are assigned capacities $u : E(G) \to \mathbb{R}_{\geq 0}$, estimating the number of standard wires that can go from any vertex to any neighbor while obeying the minimum distance requirements.

Each pin p has shapes in one or more tiles, and is represented by the respective set V_p of vertices. Each net is a set of pins, and $\mathcal{N}$ denotes the set of nets. Let $\mathcal{T}_n$ denote the set of feasible *Steiner forests* for net n, i.e., minimal edge sets $F \subseteq E(G)$ such that $F \cup \bigcup_{p \in n} K(V_p)$ connects all pins of n, where $K(V_p)$ denotes the clique on V_p. Any net n for which $\emptyset \in \mathcal{T}_n$ (e.g. because all pins are in the same tile on the same layer) can be removed.

For a net n and an edge $e \in E(G)$ let $w(n, e)$ denote the minimum required width of a wire for n along e plus the minimum required distance to any neighbor.

A major difference to most other global routers is that we allow to allocate extra space $s(n, e) \geq 0$ as this may reduce coupling capacitance and hence delay and power consumption, and may also lead to better yield.

Then the global routing task is to find for each $n \in \mathcal{N}$ a Steiner forest $T_n \in \mathcal{T}_n$ and an extra space assignment $s(n, e) \geq 0$ for $e \in T_n$ such that $\sum_{n \in \mathcal{N}: e \in T_n} (w(n, e) + s(n, e)) \leq u(e)$ for each $e \in E(G)$.

However, we are not satisfied with an arbitrary feasible solution. While a traditional objective function considers only wire length and number of vias, we can also optimize other objective functions like power consumption or expected yield, and we can also deal with constraints bounding e.g. detours of certain nets or weighted sums of capacitances on critical paths. The objective function and each of such constraints will be modeled as a *resource*. We denote by $\mathcal{R}$ the set of resources.

Each wire consumes a certain amount of some of the resources modeled by functions $\gamma^r_{n,e} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for $r \in \mathcal{R}$, $n \in \mathcal{N}$ and $e \in E(G)$. Here, $\gamma^r_{n,e}(s)$ is the estimated use of resource r if e is used by net n with allocated space $w(n, e) + s$. For all relevant resources that we consider, including electrical capacitance (with coupling), power consumption, and estimated wiring yield loss, these functions are convex (cf. [7] and [14]).

For each constraint, we have an upper bound $u^r > 0$ of how much we may use of the corresponding resource r. For the objective function, we do not have an upper bound a priori, but we can guess a value that we expect to be achievable and adapt it if needed (we could apply binary search to find approximately the optimum value, but this is usually not needed in practice).

We also model the edge capacities as resources. If $r(e)$ denotes the resource corresponding to edge e, we have $u^{r(e)} = u(e)$, $\gamma^{r(e)}_{n,e}(s) = w(n, e) + s$, and $\gamma^{r(e)}_{n,e'}(s) = 0$ for any other edge $e' \neq e$ and $s \geq 0$.

Now the task is to find for each $n \in \mathcal{N}$ a Steiner forest $T_n \in \mathcal{T}_n$ and an extra space assignment $s(n, e) \geq 0$ for $e \in T_n$ such that $\sum_{n \in \mathcal{N}} \sum_{e \in T_n} \gamma^r_{n,e}(s(n, e)) \leq u^r$ for all $r \in \mathcal{R}$.

2.2 Reduction to Min-Max Resource Sharing

Let $\chi(T) \in \{0, 1\}^{E(G)}$ denote the incidence vector of a Steiner forest T (i.e. $(\chi(T))_e = 1$ for $e \in T$ and $(\chi(T))_e = 0$ for $e \notin T$). Then

$$\mathcal{B}^{\text{int}}_n := \{(\chi(T), s) \mid T \in \mathcal{T}_n,\ s \in \mathbb{R}^{E(G)}_{\geq 0},\ s_e = 0 \text{ for } e \notin T\}.$$

represents the set of feasible solutions for net n. In our relaxation we consider the convex hull $\mathcal{B}_n := \text{conv}\left(\mathcal{B}^{\text{int}}_n\right)$ for each $n \in \mathcal{N}$. For $(x, s) \in \mathcal{B}_n$ we define

$$g^r_n(x, s) := \frac{1}{u^r} \sum_{e \in E(G): x_e > 0} x_e \, \gamma^r_{n,e}(s_e / x_e).$$

Note that we have $g^r_n(\chi(T), s) = \frac{1}{u^r} \sum_{e \in T} \gamma^r_{n,e}(s_e)$ for $(\chi(T), s) \in \mathcal{B}^{\text{int}}_n$. If each $\gamma^r_{n,e}$ is convex (as in our applications), then each g^r_n is also convex. With this notation, the global routing problem asks for an element $b_n \in \mathcal{B}^{\text{int}}_n$ for each $n \in \mathcal{N}$ such that $\sum_{n \in \mathcal{N}} g^r_n(b_n) \leq 1$ for all $r \in \mathcal{R}$. As in previous works, we now consider a fractional relaxation first and look for $b_n \in \mathcal{B}_n$ $(n \in \mathcal{N})$ approximately attaining

$$\lambda^* := \inf \left\{ \max_{r \in \mathcal{R}} \sum_{n \in \mathcal{N}} g^r_n(b_n) \,\middle|\, b_n \in \mathcal{B}_n \ (n \in \mathcal{N}) \right\}.$$

If the instance is feasible and the guess of an achievable value of the objective function is realistic (see above), then λ^* will be close to 1.

This relaxation is known as the (block-angular) Min-Max Resource Sharing Problem. In its standard formulation, the sets $\mathcal{B}_n$ are not given explicitly. Rather we assume to have an oracle, denoted by $f_n : \mathbb{R}^{\mathcal{R}}_{\geq 0} \to \mathcal{B}_n$, for $n \in \mathcal{N}$, which for $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ returns an element $b \in \mathcal{B}_n$ with $\sum_{r \in \mathcal{R}} y_r g^r_n(b) \leq \sigma \, \text{opt}_n(y)$, where $\text{opt}_n(y) := \inf_{b \in \mathcal{B}_n} \sum_{r \in \mathcal{R}} y_r g^r_n(b)$ and $\sigma \geq 1$ is a constant. In other words, we assume that we can optimize linear functions over each $\mathcal{B}_n$ efficiently with an approximation factor σ. This is indeed true in our case:

Theorem 1. [9] *The oracle functions f_n can be implemented by an approximation algorithm for the Steiner tree problem in weighted graphs.*

We use the fastest known algorithm for the Min-Max Resource Sharing Problem, due to Müller, Radke and Vygen [9]. The core algorithm works in t phases, where t has to be chosen depending on the desired approximation guarantee. In each phase, a solution for each net is computed by applying Theorem 1, and resources become more expensive as they are used. The output of the algorithm is a convex combination $\sum_{b \in \mathcal{B}^{\text{int}}_n} x_{n,b} b$ of the elements in $\mathcal{B}^{\text{int}}_n$ for each net n, i.e. $\sum_{b \in \mathcal{B}^{\text{int}}_n} x_{n,b} = 1$. See [9] for a detailed description and a proof of the following Theorem:

Theorem 2. [9] *If our oracle computes solutions within a factor σ of optimal in time θ, then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta \log |\mathcal{R}|(|\mathcal{N}|+|\mathcal{R}|)(\log\log|\mathcal{R}|+\omega^{-2}))$ time for any $\omega > 0$.*

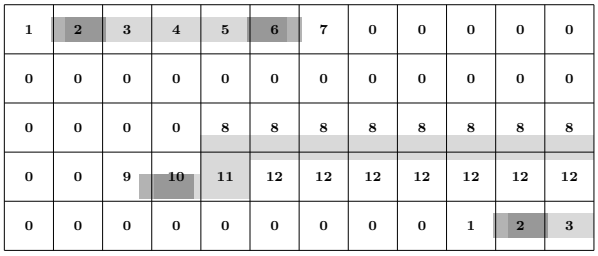

Figure 1: Shape grid cell configurations and their configuration numbers for a given wiring. Sequences of identical numbers in preferred direction are merged to intervals. In this example, we store 15 intervals with a configuration other than 0.

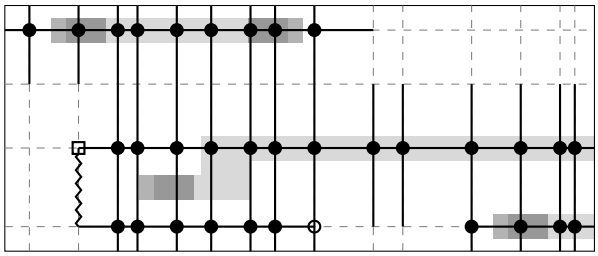

Figure 2: The fast grid: dashed lines represent x- and y-edges of the track graph, with vertices at their intersections. For some given wiretype, circles mark vertices at which no jog can start (unless ripup is allowed); if a circle is filled, then no wire in preferred direction can start at this vertex either. Thick black edges represent edges that are unusable with the given wiretype. Usability of an edge can be deduced from the vertex information in all cases except the zigzag edge: For this edge, we set a bit at one of its incident vertices (marked by a square) to indicate that usability of its incident edges must be determined by querying the shape grid.

The algorithm can be parallelized very well using *volatility-tolerant* block solvers [9] to guarantee the desired approximation ratio even with concurrent access to the same resources by multiple threads, and achieves a very good scaling with the number of processors [9].

As we need an element of $\mathcal{B}_n^{\text{int}}$ for each net n, we apply randomized rounding to the output of our resource sharing algorithm, choosing each $b \in \mathcal{B}_n^{\text{int}}$ with probability $x_{n,b}$, for each net n independently. In practice, this results only in few violations, i.e. resources $r \in \mathcal{R}$ with $\sum_{n \in \mathcal{N}} (g_n(\hat{b}_n))_r > 1$, if $\hat{b}_n$ is the solution picked by randomized rounding for $n \in \mathcal{N}$. These violations can be eliminated easily by postprocessing ("ripup and reroute") using standard heuristic techniques.

3. DETAILED ROUTING

Most of the wiring on a routing layer has the minimum possible width. Such *standard wires* can be packed best with a net-by-net routing approach if they are aligned on parallel *tracks* in preferred direction. Normally, the optimal distance of two adjacent tracks is the minimum wire width plus the minimum required distance of two wires, but in the presence of blockages, total usable track length can be maximized only with non-uniform track spacings in general. Optimum routing tracks can be computed in $O(n \log n)$ time for n axis-parallel rectangular blockages [8]. The intersection points of routing tracks with tracks projected from neighboring wiring layers define the vertices of a *track graph* in a natural way.

As most pins are not aligned with tracks, we combine a very fast on-track path search for long distance connections (cf. Section 3.3), obyeing only diff-net rules, with an off-track path search that also takes same-net rules into account and is used for pin access (cf. Sections 3.4, 3.5, and 3.6). The on-track path search operates on a two-level data structure for representing routing space, which we describe in Sections 3.1 and 3.2. Section 3.7 explains our parallelization approach for detailed routing. Finally, we describe how we combine our router with an industrial router for DRC cleanup in Section 3.8.

3.1 Shape Grid

The shape grid efficiently stores all relevant data about blockage, wire, and via shapes. The information in the shape grid allows to decide whether a wire can be placed somewhere without violating minimum distance rules. If not, it allows to find out if there is a set of shapes that can be removed such that the answer becomes positive.

The shape grid partitions the chip area on each wiring layer and on the via layers into axis-parallel rectangular *cells*, such that each of them has at most one neighbor cell to the left, right, bottom and top, respectively. The size of the cells is small enough such that shapes of different nets cannot be present in the same cell while obeying all distance rules.

In each cell, all intersections of shapes with its area are stored with coordinates relative to an anchor point (e.g. its center), plus their respective type and minimum distance requirements. Since this *cell configuration* is typically identical in a large number of cells, we store this data indirectly by a *cell configuration number* which is used as an index into a lookup table that stores the actual data.

The data volume is further reduced by grouping neighboring cells in preferred routing direction to *intervals* which are stored in an AVL-tree in each row or column of cells, depending on the preferred routing direction (see Figure 1 for an example).

Moreover, for each nonempty interval we store the net that the shapes of this interval belong to if they are removable. The type and net identifier of a shape are used to assign a *ripup level* to it. The ripup-and-reroute algorithm can then be restricted to rip out shapes of at most some specified ripup level in order to avoid re-routing of critical connections.

3.2 Fast Grid

As most of the routing is done on predefined routing tracks, we obtain faster query times by storing pre-computed data for a restricted set of locations and *wire types* (i.e. wire and via geometries with spacing requirements) based on these tracks. This data is maintained in the *fast grid* data structure. Queries for other, less frequently used, wire types are transferred to the shape grid.

If there is only on-track wiring, legality of a wire whose stick figure (usually the center line) connects two neighboring vertices of the track graph is implied by legality of the zero-dimensional stick figures at each of the vertices. Because of this, the fast grid stores information for vertices, but not for edges. This allows to group longer sequences of vertices with identical data to intervals. If data was stored for vias or jogs, more changes would occur along a track in preferred direction, resulting in a higher interval count.

If legality of a wire on an edge $\{v, w\}$ cannot be deduced from legality at v and w because off-track wires or other shapes are present in the vicinity, an extra bit at one of the vertices v and w encodes this (see Fig. 2 for an illustration). If this bit is set, a query to the fast grid is transferred to the shape grid.

We store information on the usability of up to five different wire types in the fast grid. In our experiments, this allowed to answer 97.9% of the legality queries using the information stored in the fast grid, resulting in an average overall speedup of 5.29 of on-track path search.

3.3 On-Track Path Search

The on-track path search finds a shortest path between two sets of vertices (corresponding to connected components of a net) in a graph G which results from the track graph by removing all edges whose usage would introduce a diff-net violation. We do not store G explicitly, but rather query the fast grid for usable edges as needed. If the fast grid does not store legality information for the wire type we want to use, a query to the shape grid is performed.

When routing a modern chip, millions of path searches have to be performed. Thus even a runtime linear in $|V(G)|$ would be too slow. Our on-track path search uses a Dijkstra-based algorithm with two major speed-up features:

1) Merging sequences of usable vertices on a track, and labeling whole intervals instead of single vertices. This was proposed by Hetzel [3] for the special case of equidistant routing tracks which match in all layers with the same preferred direction, generalized by Humpola [4] to arbitrary routing tracks, and further generalized by Peyer et al. [11] to more general vertex sets.

2) Using a *future cost* (similar to the A* heuristic [12]) to reduce the number of labeling steps by providing a lower bound of the distance to the target for each vertex. Peyer et al. [11] proposed a blockage-aware future cost which computes shortest paths to the target from all nodes in a supergraph of G with a simpler structure. As this also consumes non-negligible runtime, we do it only if the global routing for this connection already contains a large detour, and use a weaker future cost based on l_1-distance otherwise, which is cheaper to compute.

With edge costs

$$c(\{v, v'\}) = \begin{cases} \gamma_{\{z,z'\}} & \text{if } \{v, v'\} \text{ is a via} \\ \beta_z \cdot ||v - v'||_1 & \text{if } \{v, v'\} \text{ is a jog} \\ ||v - v'||_1 & \text{otherwise} \end{cases}$$

for two neighboring vertices $v := (x, y, z)$ and $v' := (x', y', z')$ of G, where β_z, $\gamma_{\{z,z'\}} \in \mathbb{Z}_{>0}$ are parameters that encode penalty costs for wires running in non-preferred direction and for vias, respectively, the following performance guarantee can be given:

THEOREM 3. [3, 4, 11] *A shortest path P in a track graph G with edge costs $c : E(G) \to \mathbb{N}$ as above can be found in time $O(\min\{(\Lambda + 1)|\mathcal{I}| \log |\mathcal{I}|, |V(G)| \log |V(G)|\})$, where $\mathcal{I}$ is the set of intervals representing G, and Λ is the difference of the cost of P w.r.t. c and the minimum future cost of a source vertex (i.e., Λ measures how good this lower bound is).*

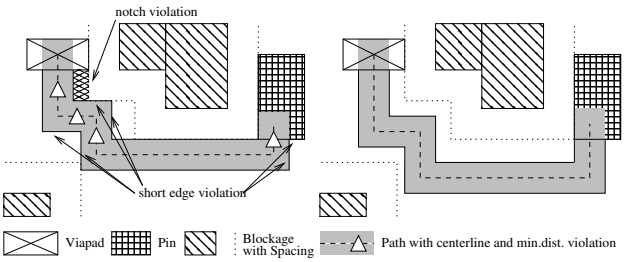

Figure 3: Example showing geometric design rule violations for geometric shortest paths without minimum segment length (left). The path is sought to connect the viapad to the pin. Taking minimum segment lengths into account, a feasible path (right) obeys these rules.

According to recent experiments on 22 nm chips, labeling and storing intervals instead of single nodes speeds up the path search by at least a factor of 6.

To support *ripup-and-reroute*, the on-track path search can be driven in a mode where vertices of the track graph that are unusable only because of minimum distance violations to modifiable shapes are flagged as usable, but high extra costs are imposed on intervals containing such vertices.

If there is unused space in a region, *wire spreading* can improve timing and manufacturing yield (both by reducing the probability of extra material defects [7], and by increasing the number of vias that can be replaced by larger, i.e. more robust, vias in postprocessing). To do this, the on-track path search imposes extra costs on intervals that should be kept free, based on congestion observed by global routing.

3.4 Same-Net Rules

Most same-net rules are in place to avoid geometric configurations with features below the lithographic capabilities, to ensure space for optical proximity correction (OPC), and to improve manufacturing yield. The list of these rules is often very long, we briefly discuss the most important ones:

- Notch violations: Even within the same path, non-adjacent segments have to obey distance requirements.
- Short edge violations: Two short adjacent edges of any polygon bounding metal area are forbidden.
- Minimum area violations: Every connected metalized polygon on a layer must have a certain minimum area, independent of its actual shape.

These rules are often violated in geometric shortest paths and Steiner trees, see also Figure 3. It is particularly important to obey them in pin access because of the often irregular geometries of pins and limited space available for fixing errors by postprocessing.

3.5 Blockage Grid for Off-Track Path Search

The main goal of the blockage grid is to support fast search for shortest (off-track) paths that avoid the most important classes of same-net rule violations. Most same-net rules can be mapped to requirements on minimum segment lengths in the constructed path [10]. Our approach to DRC-clean off-track wiring hence is as follows.

Suppose that we only have a single requirement stating that each segment of a wire must have length at least $\tau > 0$.

Call a rectilinear path τ-*feasible* if each of its segments has length at least τ and does not intersect the interior of any obstacle. We construct a data structure called *blockage grid*, which allows for finding shortest τ-feasible paths.

Starting with a Hanan grid, we add additional vertical and horizontal lines at distances $k\tau$, $k_1 \leq k \leq k_2$ and $k \in \mathbb{Z}$, from each original line, where k_1 is minimal and k_2 maximal such that each added line is at most 2τ from any of the original vertical or horizontal lines, respectively. It is not difficult to see that the total number of vertices, i.e. crossing points of lines, is then bounded by $O(n^4)$, where n denotes the number of rectangles representing the blockages. The following result shows that these vertices suffice.

THEOREM 4. [6] *Consider a set of n rectilinear obstacles in the plane, a distance $\tau > 0$, and $s, t \in \mathbb{R}^2$. If there exists a τ-feasible path from s to t, then there is also a shortest τ-feasible path from s to t all whose segments have vertices of the blockage grid as endpoints.*

In order to respect the minimum segment length τ during the path search, we construct a path-preserving digraph G on which we can run a regular shortest path algorithm. We construct G from the blockage grid as follows. G contains up to four vertices for each vertex of the blockage grid; one collecting incoming arcs of each direction. Arcs of G connect neighboring vertices of the blockage grid where this does not induce a bend together with the incoming direction. Additional arcs connect each vertex to the (at most two) nearest vertices at distance $\geq \tau$ perpendicular to the incoming direction. This way, each bend is followed by a long arc as G does not contain short arcs after bends. More sophisticated rules like short edge avoidance can be added as well by introducing corresponding edges in G or removing edges that induce a respective violation.

Storing the blockage grid coordinates only takes $O(n^2)$ space. For a multi-layer path search, we have a blockage grid for each wiring and via layer. Coordinates are also induced by blockages of neighboring layers. Vertices of two adjacent wiring layers are connected if a via is allowed here.

3.6 Off-Track Pin Access

Pins that are not aligned with routing tracks require off-track pin access [10], which is based on the off-track path search and blockage grid presented in Section 3.5. For each such pin we construct a *catalogue* of several DRC clean paths connecting it to on-track points within a small radius. Among these, we compute one primary access path for each pin such that the set of these paths for the pins of each circuit forms a *conflict-free solution* (see also Figure 4), i.e. is DRC-clean also w.r.t. diff-net rules. We add the primary access paths to the routing space as reservations before actually starting to route: this way, newly added wires do not invalidate the precomputed conflict-free solution. We do all this in a preprocessing step, but also have the functionality to construct additional paths on-the-fly when needed.

Although there may be millions of circuits placed on the chip, containing the vast majority of pins, there are only a few thousand prototype circuits in a library. Furthermore, non-circuit blockages and wires, e.g. stemming from power supply or pre-designed clock nets, often have a regular structure. The preprocessing step of constructing the catalogues for the pins exploits this.

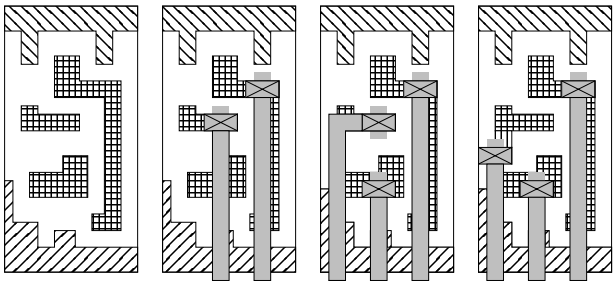

Figure 4: Examples of different pin access solutions. On the left, a circuit is given with its pins (on layer 1) and blockage structure. The following three figures show three different situations for pin access wiring (grey, on layer 2). First, a situation that may occur by a greedy approach is shown: after connecting the first two pins, the third pin is blocked. The two figures on the right show two conflict-free solutions (using on-track via positions on metal 2 as endpoints). The one on the right is superior and will be chosen by our algorithm.

We partition the set of circuits into *circuit classes*. Expanding the bounding area of a circuit and taking all shapes influencing this area and the track coordinates into account, we can identify geometrically equal situations on the chip area (up to translation, mirroring, and rotation), and we thus collect these configurations into equivalence classes. The preprocessing is then done based on these.

To anticipate the effects of local congestion, we are not interested in feasible conflict-free solutions alone, but also take (local) spreading of the paths into account. Knowing that the ongrid path-search adds wiring around the endpoints of the access paths, we evaluate a conflict-free solution based on spreading of endpoints, number of blocked tracks, directions of feasible on-track continuation, and length.

We use a branch-and-bound based enumeration technique called destructive bounding to compute a good conflict-free solution for each circuit class.

3.7 Parallelization

The parallelization approaches we use in global and detailed routing are fundamentally different, although both are shared-memory approaches. While our global router allows different threads to work in the same region, in detailed routing we partition the chip area into regions assigned to threads in order to strictly obey minimum distance rules without needing a collision detection mechanism.

Hence, in detailed routing, each thread is allowed to make only changes that do not affect regions assigned to other threads. E.g., if a wire of a certain type has large minimum distance requirements to other objects, it may not be put close to the border of the region assigned to a thread. Because this allows only a subset of connections to be closed, a sequence of partitions is defined such that in each of them the estimated workload in each region is balanced. The number of regions, and hence the number of threads used, is reduced in later partitions to allow longer connections to be closed. In addition, there is an initial critical net routing step. Nets can be considered critical here from a timing perspective, or because they use wide wires that would be hard to route when the majority of nets has already been routed.

3.8 Combination with External DRC Cleanup

The algorithms described above constitute the core parts of BonnRoute, which is the routing solution of the University of Bonn, developed within the scope of our cooperation with IBM. It is part of the BonnTools [5]. BonnRoute has been used by IBM and its customers for the design of more than thousand of the most complex chips. It focuses on near-optimum packing of wires and avoids only the most important classes of DRC violations: first, with very few exceptions, it leaves no violations of diff-net spacing rules; second, DRC violations that need additional space for fixing, e.g. minimum area errors, are avoided as much as possible. In addition, many other types of rules are handled by a post-processing step. Cleanup of the remaining DRC violations is left to an external tool. The experimental results in Section 4 demonstrate that this approach works very well in practice.

4. EXPERIMENTAL RESULTS

We compare routing results of an industry standard router (ISR) with our combined flow ("BR+ISR"), in which we use the same ISR to clean up design rule violations left by BonnRoute. ISR is a current router which is used in practice on many industry designs of former and recent technologies. In contrast to BonnRoute, it uses traditional rip-up and reroute [13] and a track assignment step to cover long distances.

All experiments were done on a 3.47GHz Intel Xeon machine. Both routers used 12 threads. Table 1 reports runtime, wire length, via counts, scenic nets and error counts, which are the sums of the numbers of DRC violations and opens (i.e. number of connected components minus number of nets). Chips 5 and 8 are 32nm designs, all others are 22nm. Out of 23:08 hours total runtime of our combined "BR+ISR" flow on these instances, BonnRoute takes only 7:12 hours. DRC cleanup takes considerably longer although only local changes are made. Nevertheless, the total runtime of our "BR+ISR" flow is less than half compared to "ISR". Global routing takes only 24 minutes of the total BonnRoute runtime, 17 minutes of which are spent in the resource sharing algorithm.

We call a net *scenic* if it has routed wiring length of at least 100 μm and a detour of at least 25% or 50%, respectively, over the length of a Steiner tree with minimum length (for nets with at most 9 terminals [2]) or approximately minimum for nets with 10 or more terminals (obtained by a Steiner tree heuristic). The Steiner trees used as baseline for defining scenic nets are of course identical in the "ISR" and "BR+ISR" rows. The table shows that the quality of the routing obtained by BonnRoute is far superior.

Chip (Nets)	Runtime	Wire len. (m)	Vias	Scenics (≥25%)	Scenics (≥50%)	Err.
1	2:20:10	3.08	1,266 k	587	103	18
(121k)	0:35:53	2.97	932 k	7		28
2	2:09:32	3.56	1,113 k	1,545	1,071	124
(126k)	1:28:38	3.40	904 k	16		194
3	2:04:40	3.07	1,240 k	695	286	14
(130k)	0:35:23	2.97	925 k	8	1	25
4	2:28:17	3.37	1,248 k	502	232	15
(135k)	0:42:19	3.24	945 k	5		3
5	2:49:58	9.91	2,936 k	2,401	1,157	50
(384k)	1:46:37	9.57	2,399 k	105	6	61
6	12:42:42	14.01	4,253 k	14,829	10,668	459
(438k)	5:28:50	12.47	3,234 k	1,568	520	506
7	8:51:05	12.31	4,039 k	3,853	2,310	154
(466k)	2:31:47	11.87	3,185 k	350	193	183
8	14:44:44	38.87	7,769 k	11,516	6,539	111
(962k)	9:59:18	37.31	6,239 k	2,619	1,285	117
Sum	48:11:08	88.18	23,864 k	35,928	22,366	945
	23:08:45	83.80	18,763 k	4,678	1,285	1,117

Table 1: Runtimes, wire length, vias, scenic nets and error counts for ISR (light gray) and our combined BR+ISR flow (dark gray)

Acknowledgment

The authors would like to thank our cooperation partners at IBM, in particular Karsten Muuss, Sven Peyer, and Gustavo Tellez. Moreover, we would like to acknowledge the contributions of former members and students of the BonnRoute team, in particular Asmus Hetzel, Christoph Albrecht, André Rohe, Sven Peyer, Jesco Humpola, Lars Bellinghausen, Corinna Gottschalk, Daniel Joachimi, Niko Klewinghaus, Felix Nohn, Thomas Petig, and Rudolf Scheifele.

5. REFERENCES

[1] C. Alpert, Z. Li, M. Moffitt, G. Nam, J. Roy, and G. Tellez. What makes a design difficult to route. In *Proc. ISPD*, pages 7–12, 2010.

[2] C. Chu and Y.-C. Wong. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 27:70–83, 2008.

[3] A. Hetzel. A sequential detailed router for huge grid graphs. In *Proc. DATE*, pages 332–339, 1998.

[4] J. Humpola. Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen. *Diploma Thesis, University of Bonn*, 2009.

[5] B. Korte, D. Rautenbach, and J. Vygen. BonnTools: Mathematical innovation for layout and timing closure of systems on a chip. *Proc. IEEE*, 95:555–572, 2007.

[6] J. Maßberg and T. Nieberg. Rectilinear paths with minimum segment lengths. *Discrete Appl. Math., to appear.*

[7] D. Müller. Optimizing yield in global routing. In *Proc. ICCAD*, pages 480–486, 2006.

[8] D. Müller. *Fast Resource Sharing in VLSI Routing*. PhD thesis, University of Bonn, 2009.

[9] D. Müller, K. Radke, and J. Vygen. Faster min-max resource sharing in theory and practice. *Math. Prog. Comp.*, 3:1–35, 2011.

[10] T. Nieberg. Gridless pin access in detailed routing. In *Proc. DAC*, pages 170 – 175, 2011.

[11] S. Peyer, D. Rautenbach, and J. Vygen. A generalization of dijkstra's shortest path algorithm with applications to VLSI routing. *J. Discrete Algorithms*, 7:377–390, 2009.

[12] P. E. Hart and N. J. Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 2:100–107, 1968.

[13] J. Salowe. Rip-up and reroute. In C. Alpert, D. Mehta, and S. Sapatnekar, editors, *Handbook of Algorithms for Physical Design Automation*, pages 615–626. CRC Press, 2009.

[14] J. Vygen. Near-optimum global routing with coupling, delay bounds, and power consumption. In *Proc. IPCO*, pages 308–324, 2004.