# Optimal Routing-Conscious Dynamic Placement for Reconfigurable Devices

Ali Ahmadinia[1], Christophe Bobda[1], Sándor P. Fekete[2], Jürgen Teich[1], and Jan C. van der Veen[2]

[1] Department of Computer Science 12, University of Erlangen-Nuremberg, Germany
{ahmadinia,bobda,teich}@cs.fau.de
[2] Department of Mathematical Optimization, Braunschweig University of Technology, Germany, {s.fekete,j.van-der-veen}@tu-bs.de.

**Abstract.** We describe algorithmic results for two crucial aspects of allocating resources on computational hardware devices with partial reconfigurability. By using methods from the field of computational geometry, we derive a method that allows correct maintainance of free and occupied space of a set of $n$ rectangular modules in optimal time $\Theta(n \log n)$; previous approaches needed a time of $O(n^2)$ for correct results and $O(n)$ for heuristic results. We also show that finding an optimal feasible communication-conscious placement (which minimizes the total weighted Manhattan distance between the new module and existing demand points) can be computed in $\Theta(n \log n)$. Both resulting algorithms are practically easy to implement and show convincing experimental behavior.

**Keywords:** Reconfigurable computing, field-programable gate array (FPGA), module placement, occupied space manager (OSM), routing-conscious placement, Manhattan metric, line sweep technique, optimal running time, lower bounds.

## 1 Introduction

One of the cutting-edge aspects of reconfigurable computing is the possibility of *partial* reconfiguration of a device. In this paper we resolve two crucial issues for this task:

1. Given a set of $n$ rectangular modules that have been placed on a chip, identify all feasible positions for a new module.
2. Given a set of $n$ rectangular modules that have been placed on a chip, a new module, and demands for connecting it to existing sites, find a feasible position for the module that minimizes the total weighted distance to the given sites.

**Related Work.** The first of the above issues is the task of maintaining free space. Bazargan et al. [3] describe how to achieve this by maintaining the set of all maximal free rectangles; as this set can have size $\Omega(n^2)$, the complexity is quadratic. Alternatively, they propose partitioning free space into only $O(n)$ free rectangles; the price for this improved complexity is the fact that no feasible placement may be found, even though one exists. Walder et al. [7] have suggested ways to reduce this deficiency and did report on experimental improvement, but their $O(n)$ procedure is still a heuristic approach that may fail in some scenarios. Thus, there remains a gap between $O(n^2)$ methods that report

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 847–851, 2004.
© Springer-Verlag Berlin Heidelberg 2004

an accurate answer, and $O(n)$ heuristics that may fail in some scenarios. Ahmadinia et al. [1] suggested maintaining occupied space instead of free space, but (depending on the computational model) their approach is still quadratic.

The more difficult task of routing-conscious placement has received less attention: Clearly, *optimal* placement of a new module has to go beyond feasible placement. For configurable computing, this second aspect has only been treated very recently, in work by Ahmadinia et al. [1], who suggest a heuristic to find a feasible placement for a new module with small total weighted Euclidean distance to a set of demand points. However, according to [5], using Manhattan distances is more appropriate.

**Our Results.** We resolve both of the above issues:

- We give a $O(n \log n)$ method to provide an occupied space manager (OSM). This approach uses a plane-sweep approach from computational geometry.
- We give a matching lower bound of $\Omega(n \log n)$ for locating a maximal free rectangle between a set of $n$ modules, showing that our method has optimal complexity.
- We show that our OSM can be extended to find a feasible position that minimizes total weighted Manhattan distance to existing sites. The resulting algorithm still has an optimal running time of $\Theta(n \log n)$.
- We describe implementation details to illustrate that our method is fast and easy.
- We provide experimental data to demonstrate the practical usefulness of our results.

In Section 2, we present our optimal OSM. Section 3 describes optimal routing-conscious placement, followed by implementation details and experimental data in Section 4. See [2] for a full version of our paper.

## 2 The Occupied-Space Manager

Our occupied-space manager is based on a modification of the well-known algorithm ContourOfUnionOfRectangles (CUR) [6] that finds the contour of a union of axis-parallel rectangles. As the number of contour segments is linear in $n$, we achieve a running time of $O(n \log n)$. Note that we do *not* require the contour to be connected, i.e., our approach works even if there are holes in the arrangement.

As shown in Figure 1, we shrink the area of the chip and simultaneously blow up the existing modules by half the width and half the height of the new module. Then placing the new module $m$ reduces to finding free space for a point.

In general finding the contour of a set of axis-aligned rectangles can be done by using the CUR algorithm as described in [6]. Our algorithm is a modification of CUR and returns a linear number of vertical and horizontal line segments. The building blocks of CUR are an algorithmic technique from computational geometry called *plane sweep* and a data structure called *segment tree*. See [4] for in-depth introductions.

The crucial part of our algorithm are two plane sweeps: one horizontal sweep that discovers all the vertical contour segments and one vertical sweep that finds all horizontal segments.

For the horizontal sweep we add for each of the modules in $M'$ $(x'_i, Open, y'_i, y'_i+h'_i)$ and $(x'_i + w'_i, Close, y'_i, y'_i + h'_i)$ to a list $L$. After sorting this list lexicographically all elements are processed. In case of an $Open$ event the corresponding contour points are

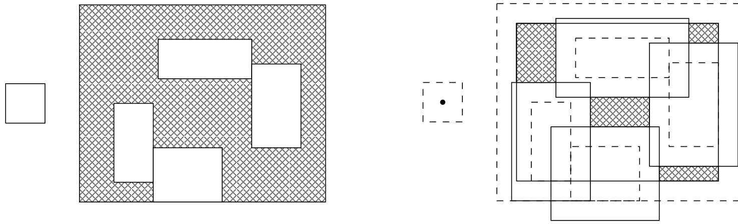

**Fig. 1.** (Left) A set of existing modules, and an additional module. (Right) Expanding existing modules and shrinking chip area and the new module reduces free-space management to placing a single point.

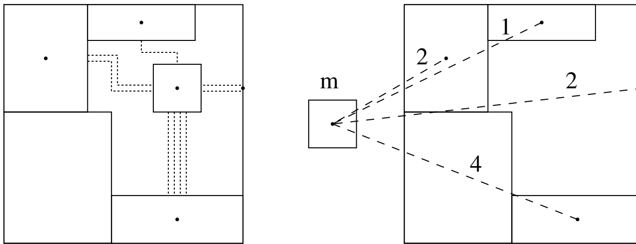


**Fig. 2.** (Left) Physical chip (Right) Communication model with $k = 4$: The numbers on the connections are the $b_{im}$.

retrieved from the segment tree and the segment $[y'_j, y'_j + h'_j]$ is added to the tree. For a $Close$ event the segment $[y'_i, y'_i + h'_i]$ is removed from the tree and the corresponding contour points are retrieved.

In the CUR algorithm we would construct the horizontal contour segments from the vertical segments. In our setting we would not find free space of height 0. So we need to do another vertical plane sweep to discover all horizontal segments.

In the algebraic tree model of computation, there is a lower bound of $\Omega(n \log n)$ on the complexity of deciding the maximum size of a free rectangle between $n$ existing rectangles. In summary, we get the following result:

**Theorem 1.** *The complexity of* FINDCONTOURSEGMENTS *is* $\Theta(n \log n)$.

## 3 Routing-Conscious Placement

An appropriate measure for the cost of communication between modules is their Manhattan distance, weighted by the relative amount of communication. See Figure 2. Finding an optimal feasible placement under this measure can still be achieved in time $\Theta(n \log n)$, making use of local optimality properties, our OSM, and another application of plane sweep techniques.

When placing an additional module $m$ at position $(x, y)$, with existing modules placed at $(x'_i, y'_i)$ and buswidth $b_{im}$ for the communication path needed to create a

routing unit between modules $i$ and $m$, we consider the objective function

$$\min\{\sum_{i=1}^{k} b_{im}\|(x'_i, y'_i) - (x, y)\|_1 : (x'_m, y'_m) \in F' \setminus \bigcup_{m'_i \in M'} m'_i\}.$$

In the Manhattan metric, this can be reformulated to

$$c(x', y') = \sum_{i=1}^{k} b_{im}|x_i - x'| + \sum_{i=1}^{k} b_{im}|y_i - y'|.$$

This means we may consider two separate minimization problems, one for each coordinate. If we ignore feasibility, both minima are attained in the respective weighted medians. As we already sort the coordinates for performing plane sweeps, the running time for this step is not critical. If the median is in the occupied space there are only two other types of points where the global optimum could be located.

One type of point can be found by intersecting the contour of the occupied space with the median axes $l_x = \{(x_{med}, y) : y \in [0, H]\}$ and $l_y = \{(x, y_{med}) : x \in [0, W]\}$. In these points the $x$- or $y$-coordinate of the gradient vanishes. We cannot move in the direction of a better solution because that way is blocked by either a vertical or a horizontal segment of the contour.

The other type of points are some of the vertices of the contour. These points are the intersections of horizontal and vertical segments forming an interior angle of $\frac{\pi}{2}$ pointing in the direction of the median. In these points neither of the gradients vanishes, but local improvement is blocked by contour segments.

By inspecting all $O(n)$ local optima one finds the global optimum. Using incremental plane sweep techniques, evaluation of all local optima can be achieved in time $O(n \log n)$. Again, see [2] for details.

As the lower bound on feasible placement still applies, we get the following:

**Theorem 2.** *A feasible position with minimum communication cost can be computed in time $\Theta(n \log n)$.*

## 4 Experimental Results

The running time of our algorithm is not only good in theory, but also quite practical (as constants are small) and easy to implement. Here we show some results of our implementation. See Table 1 for an overview and [2] for details.

We have randomly generated different kinds of benchmark instances with 100 modules. The instances differ in module size and distribution of the sizes. We benchmarked a g++ 3.2 compiled c++ implementation of our algorithm against the algorithms described in [1] and [3]. Shown in the first set of columns in the table is a comparison of overall running times for 100 modules for each instance in milliseconds on a 2.53GHz Intel Pentium 4. Remarkably, our algorithm has clearly the fastest running times, even though it computes a much better solution. This illustrates the superiority of a plane-sweep apporach. Clearly, the difference in running times will increase for even larger instances. The second set of columns compares the average routing cost per module. Note that in

**Table 1.** Experimental results for the different benchmark instances. Overall running time, average routing cost for each module, and rejection rate are shown for the different algorithms. RCP denotes the algorithm described in this paper, NAOP refers to the algorithm as described in [1] and KFF is the algorithm KAMER combined with First Fit as presented in [3].

| | | Running Time (ms) | | | Routing Cost | | | Rejection Rate | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | RCP | NAOP | KFF | RCP | NAOP | KFF | RCP | NAOP | KFF |
| Uniform | 5-10% | 173 | 197 | 204 | 1403 | 3641 | 9522 | 0% | 0% | 0% |
| Uniform | 10-15% | 162 | 208 | 194 | 1747 | 5490 | 14311 | 0% | 1% | 0% |
| Uniform | 15-20% | 160 | 172 | 158 | 2044 | 7250 | 19791 | 2% | 5% | 1% |
| Uniform | 20-25% | 156 | 181 | 161 | 1987 | 7061 | 20159 | 10% | 12% | 9% |
| Uniform | 5-25% | 168 | 224 | 215 | 1721 | 6741 | 21347 | 5% | 8% | 5% |
| Increasing | 5-25% | 196 | 252 | 243 | 1931 | 6914 | 21910 | 8% | 14% | 6% |
| Decreasing | 25-5% | 175 | 232 | 228 | 611 | 2311 | 11712 | 0% | 3% | 4% |

[1], placement is done according to a weighted Euclidean distance, and optimization is only done heuristically. As a consequence, the objective values are markedly higher. [3] does not take routing cost into account and places by some bin-packing like heuristic that tries to minimize rejection rate. As a result, communication cost is one order of magnitude higher than for our method. The third set of columns compares the average number of modules that had to be rejected due to lack of space on the chip, which is one of the objectives in [3]. Even though this figure is not considered by our algorithm, the total number of rejected modules for our algorithm is precisely the same as for [3]. Again, our results dominate the ones for [1] by a clear margin.

In summary, our algorithm is faster, better and more robust against rejection than the method described in [1]. It is also faster, much better and as robust against rejection as the approach described in [3].

## References


1. A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich. A new approach for on-line placement on reconfigurable devices. In *Proc. IPDPS-2004, RAW-2004, IEEE-CS Press*, 2004.
2. A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. van der Veen. Optimal routing-conscious placement for reconfigurable computing. Manuscript (submitted), 2004.
3. K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *In IEEE Design and Test - Special Issue on Reconfigurable Computing*, January-March:68–83, 2000.
4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
5. J. Mache and V. Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proc. Third Joint Conference on Information Sciences, Sessions on Parallel and Distributed Processing*, volume 3, pages 223–226, 1997.
6. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.
7. H. Walder, C. Steiger, and M. Platzner. Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing. In *Proc. IPDPS-2003, RAW-2003, IEEE-CS Press*, page 178.