第 26 卷 第 10 期 半 导 体 学 报 Vol. 26 No. 10
2005 年 10 月 CHINESE JOURNAL OF SEMICONDUCTORS Oct. ,2005

Temporal Floorplanning Using Solution Space Smoothing Based on 3D-BSSG Structure

Zheng Shuyi , Dong Sheqin , and Hong Xianlong

(Department of Computer Science & Technology , Tsinghua University , Beijing 100084 , China)

Abstract : We develop a 3D bounded slice-surface grid (3D-BSSG) structure for representation and introduce the solution space smoothing technique to search for the optimal solution. Experiment results demonstrate that a 3D-BSSG structure based algorithm is very effective and efficient.

Key words : temporal floorplanning ; FPGA ; 3D-BSSG; solution space smoothing
EEACC : 7140D ; 5120
CLC number : TN47 **Document code :** A **Article ID :** 0253-4177(2005)10-1916-09

1 Introduction

Temporal floorplanning problems come from dynamically reconfigurable FPGAs. This kind of hardware system usually consists of a reconfigurable functional unit (RFU)[1] which can be programmed during the execution of the program with varying configurations at different times[2]. Each configuration ,at a certain time , has several RFU operations (called RFUOPs or modules) which are mapped to different parts of a complete program or function.

When the RFU is configured ,not all the modules can be placed on the chip at the same time due to area limitations. Therefore ,we have to schedule these modules with a proper loading sequence in order to meet the place constraint and fulfill the whole function in less time. Naturally ,it becomes a 3D placement problem. The objective is to allocate modules in the RFU to optimize both the area and execution time without violating the temporal constraints.

Several methods to deal with such a problem have already been proposed recently. Teich *et al*. first used a component graphs to address it[3] . They assumed no dependence among scheduled modules and derived necessary and sufficient conditions for a feasible placement and proposed an enumeration scheme by using a branch-and-bound tree search algorithm to find a feasible solution. Later ,they extended their work and took into account the precedence constraints using a graph theoretic characterization of feasible solutions[4] . Bazargan *et al*. dealt with two types of placement in reconfigurable systems : online placement , where arrival time of RFUOP is determined at runtime and is not known a priori ,and offline placement in which the schedule is known at compile time[1,2,5] . In the case of online placement , they allocated the free space of RFU to an RFUOP dynamically based on greedy algorithm. In the case of offline placement , they presented an algorithm based on the simulated annealing method[14] and got a better placement than the ones generated by their online algorithm. Unlike above researchers , Reference [6] proposed a topological representation named 3-dimensional sub-transitive closure graph (3D-subTCG) to solve the temporal floorplanning problem. It is the first work that used a topological representation to han-

Received 18 February 2005 ,revised manuscript received 27 June 2005

dle the problem. They used the simulated annealing method to search the solution space and got better results than Sequence Triplet.

In this paper, we present a novel method using a three-dimensional bounded slice surface grid (3D-BSSG) structure to represent a placement in the RFU. It is developed from BSG structure, which is proposed to handle classical 2D floorplanning/placement problems[7]. In contrast with 3D-subTCG in Ref. [6], though both are topological representations, the 3D-BSSG structure is easier for coding and simpler to get a neighborhood placement in the course of searching in the solution space. Moreover, by changing the "shape" of the 3D-BSSG structure, our method is able to deal with problems concerning non-regular shaped chips and also can meet the shape demand on the placement. Based on the 3D-BSSG structure, we introduce a solution space smoothing method to achieve an optimal placement. Solution space smoothing is a special technique of multi-space search developed in recent years[8,9]. Compared with a simulated annealing algorithm, it needs few control parameters that are easy to be determined.

We adopt several benchmarks of early researchers to test our approach and the experimental results show that it is a new and efficient method for temporal floorplanning problems.

2 3D-BSSG structure

2.1 Topological structure of 3D-BSSG

The 3D-BSSG structure is a topology, defined in 3-dimension space using an *xzy*-coordinate system. In order to describe the 3D-BSSG structure conveniently, we use a physical image as well as the mathematical definition. See Fig. 2.

First we define the unit segments. On the *xyz*-coordinate system, we define, by the following formulas, three types of open surface segments (UX, UY, and UZ) associating with coordinate axis respectively. Each UX, UY or UZ is called the 3D-BSSG-unit or simply unit.

$$UX_{i,j,k} = \{(x, y, z) \mid x = i, j - 1 < y < j + 1, k - 1 < z < k + 1\}$$

$$UY_{i,j,k} = \{(x, y, z) \mid y = j, i - 1 < x < i + 1, k - 1 < z < k + 1\}$$

$$UZ_{i,j,k} = \{(x, y, z) \mid z = k, i - 1 < x < i + 1, j - 1 < y < j + 1\}$$

$(i, j, k$: integers$)$

Note that each unit is a 2 ×2 square surface segment whose subscripts i, j, k denote its center. And all units are perpendicular to the axis which they associate with. See Fig. 1.

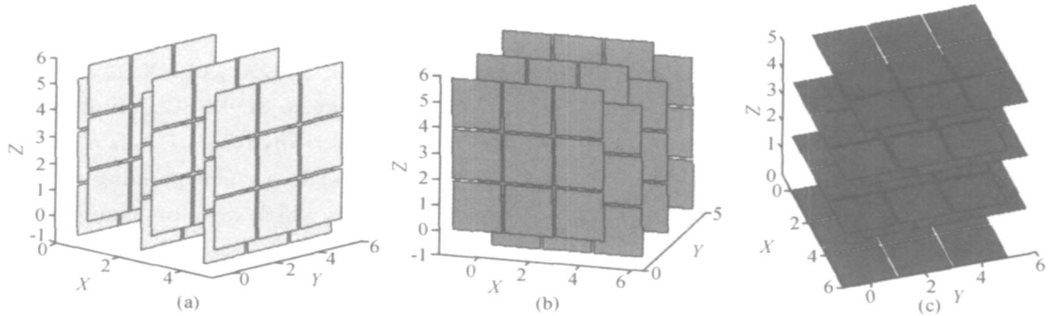

Fig. 1 3D-BSSG-unit (a) UX; (b) UY; (c) UZ

3D-BSSG is a system consisting of the set U_{BSSG} of such surface segments (See Fig. 2):

$$U_{BSSG} = \{UX_{i,j,k} | i, j, k: \text{integers}, i + j: \text{odd}, j + k: \text{odd}\} \cup \{UY_{i,j,k} | i,j,k: \text{integers}, i + j: \text{even}, j + k: \text{odd}\} \cup \{UZ_{i,j,k} | i,j,k: \text{integers}, i + j: \text{odd}, j + k: \text{even}\}$$

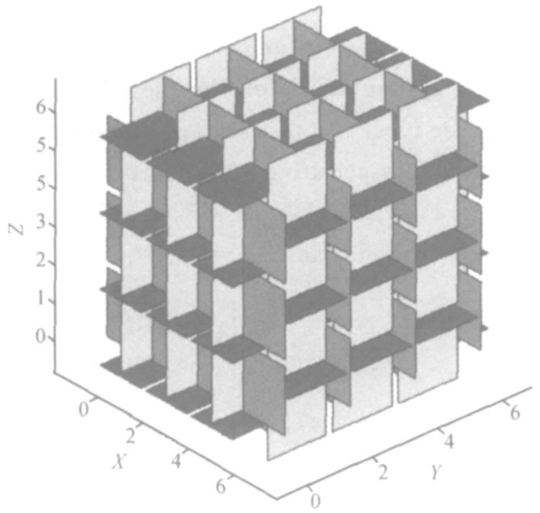

Fig. 2 Part of 3D-BSSG structure

Then we can introduce the topological relations between units. It should be mentioned that the relations defined below only exist between units in the same direction (x, y or z) of the xyz-coordinate. There is no such relation between units of different directions.

Two horizontal units $UX_{i1,j1,k1}$ and $UX_{i2,j2,k2}$ are said adjacent if $|i1 - i2| = 1$, $|j1 - j2| = 1$, and $|k1 - k2| = 1$. A horizontal unit $UX_{i1,j1,k1}$ is said right-to $UX_{i2,j2,k2}$ if $i1 - i2 = 1$, $|j1 - j2| = 1$, and $|k1 - k2| = 1$. The relation "right-to" is extended transitively: if a horizontal unit $UX_{i1,j1,k1}$ is right-to another horizontal unit $UX_{i2,j2,k2}$ and $UX_{i2,j2,k2}$ is right-to $UX_{i3,j3,k3}$, then $UX_{i1,j1,k1}$ is right-to $UX_{i3,j3,k3}$. The relations between y-direction units (behind) and z-direction units (above) can be defined analogously. For example, in Fig. 2, $UZ_{0,1,1}$ is "above" $UZ_{1,0,0}$.

A cuboid space surrounded by adjacent pairs of x-direction, y-direction, and z-direction units is called the room. If a room's left-near-bottom corner is at (i, j, k), we call it $room_{i,j,k}$. Figure 3 illustrates the boundary adjacent units of $room_{0,0,0}$.

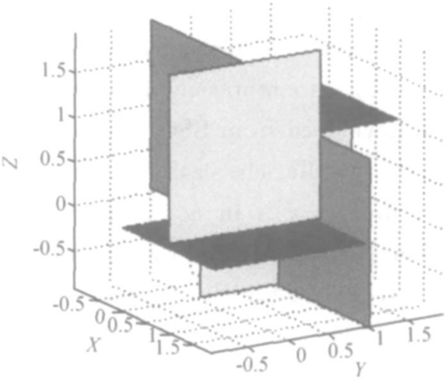

Fig. 3 Boundary adjacent units of $room_{0,0,0}$

By definition, the 3D-BSSG is an infinite grid. But for the convenience of description and coding, it is wise to bound the grid within a finite grid $3D\text{-}BSSG_{p \times q \times r}$ (p, q, r: positive integers) whose left-near-bottom corner is the origin (0, 0, 0) and the right-far-above corner is (p, q, r). We call it the domain of size $p \times q \times r$. For compactness, portions of units jutting outside the domain are cut off.

In order to show above defined topological relations among units more clearly, three directed graphs are defined to represent, respectively, the relations "right-to", "behind", and "above". They are $G_x(V_x, E_x)$, $G_y(V_y, E_y)$, and $G_z(V_z, E_z)$.

Given a domain $3D\text{-}BSSG_{p \times q \times r}$, $V_x = \{s_x, t_x\} \cup \{u_{i,j,k}\}$ where $u_{i,j,k}$ corresponds to the unit $UX_{i,j,k}$. Edges are defined as follows. s_x is a source connected to all the vertices corresponding to the left x-direction units, i. e. $UX_{0,1,0}$, $UX_{0,3,0}$, …, $UX_{0,j,k}$, … where, according to 3D-BSSG definition, j is odd, k is even and $1 \leq j \leq q$, $0 \leq k \leq r$. t_x is a sink connected from all the vertices corresponding to the right x-direction units, which are, for example of the case p: even, $UX_{p,1,0}$, $UX_{p,3,0}$, …, $UX_{p,j,k}$, … where j is odd, k is even, and $1 \leq j \leq q$, $0 \leq k \leq r$. The other edge $(u_{i1,j1,k1}, u_{i2,j2,k2})$ exists if and only if $UX_{i2,j2,k2}$ is right-to and adjacent to $UX_{i1,j1,k1}$. See Fig. 4.

The y-direction and z-direction unit adjacency graph G_y and G_z are similarly defined.

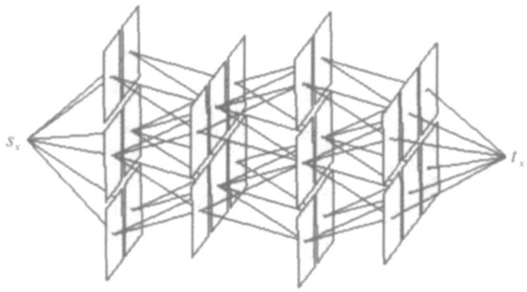

Fig. 4　Directed graph G_x

It is easy to see that each vertex of G_x has four vertices which are right-to it except the source, the sink, and part of boundary vertices. If we draw these graphs over 3D-BSSG$_{p\times q\times r}$ putting the vertices on the centers of units, each room is crossed exactly by one edge in each of G_x, G_y, and G_z. By this relation, an edge and a room are conveniently referred to by the other in such a fashion as "an edge that crossed room r", or "a room which edge e crosses".

On the basis of above definition, relations between rooms can be easily introduced. The x-direction is still taken as an example. Let $r1$ and $r2$ be two rooms. If a directed path of x-direction crosses $r1$ first and then crosses $r2$, $r2$ is said right-to r_1. It can be proved that any two rooms impossibly have relations of more than one direction. That is to say, they have either a unique relation of only one direction or no relation at all.

2.2　Obtain 3D placement from an assignment in 3D-BSSG

In the reconfigurable architecture, a module v is loaded into the device for a period of time for execution. Suppose we are given an input V, which is a set of modules v of different sizes and durations, where $|V| = n$. Assuming that $n \leq p \times q \times r$, an assignment of V is a one-to-one mapping of modules into the rooms of 3D-BSSG$_{p\times q\times r}$. A room to which no module is assigned is empty.

Weighting of unit adjacency graphs G_x, G_y, and G_z is to associate each edge e with a real number w(e) by the following formula:

If $e \in E_x$ and e crosses a non-empty room,
$w(e)$ = x-direction length of the module assigned there.

If $e \in E_y$ and e crosses a non-empty room,
$w(e)$ = y-direction length of the module assigned there.

If $e \in E_z$ and e crosses a non-empty room,
$w(e)$ = duration of the module assigned there.

Otherwise, that is, if e is either to cross an empty room or has its end-vertex on the source or sink, $w(e) = 0$.

After above weight assignment, the length of the longest path from the source of a unit adjacency graph (G_x, G_y or G_z) to each vertex of it can be calculated by performing a well-known longest path algorithm[11] which we refer to procedure: LONGEST-PATH LENGTH (G) where graph G is the input. The time complexity of LONGEST-PATH LENGTH (G) is $O(pqr)$. The purpose of computing the longest path length in the unit adjacency graphs is to determine the positions of modules.

Given an assignment of V to 3D-BSSG$_{p\times q\times r}$, we use the BSSG_To_Placement procedure described below to obtain a module's positions on the chip and its start time:

Let v be a module assigned to a room whose boundary left unit is UX_v, front unit is UY_v and bottom unit is UZ_v. Their corresponding vertices are u_{UX}, u_{UY}, and u_{UZ}. Calculate the longest paths $l_x(u_{UX})$, $l_y(u_{UY})$, and $l_z(u_{UZ})$ respectively. Since every module's x-direction length, y-direction length, and duration are all embodied in the adjacency graphs, it is not difficult to understand that the module's left-bottom corner on the chip is at $(l_x(u_{UX}), l_y(u_{UY}))$ and its start time is $l_z(u_{UZ})$. Figure 3 is a simple example that shows a placement obtained from assignment. There are six modules assigned to 3D-BSSG$_{4\times 4\times 4}$. The assignment of these modules in 3D-BSSG$_{4\times 4\times 4}$ and their corresponding placement positions are given in Table 1 along with their size and duration.

Table 1 Size, duration, assignment in 3D-BSSG$_{4\times4\times4}$ and placement position of six modules

Module	Width	Height	Duration	Assignment in 3D-BSSG$_{4\times4\times4}$	Placement position
1	1	5	4	$room_{3,4,3}$	(5,2,0)
2	5	3	4	$room_{1,3,2}$	(0,0,0)
3	2	3	3	$room_{4,2,2}$	(7,0,0)
4	2	3	1	$room_{3,2,3}$	(7,0,3)
5	2	2	3	$room_{2,2,1}$	(5,0,0)
6	2	2	1	$room_{2,2,2}$	(5,0,3)

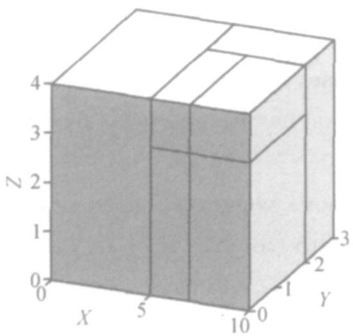

Fig. 5 Placement obtained from assignment in Table 1

Since any two rooms in topological relation (right-to, behind or above) keep the relation in the output of the procedure, we can prove that no two modules overlap.

In addition, it is apparent that the minimum bounding box of a placement is $(l_x(t_x) \times l_y(t_y) \times l_z(t_z))$.

2.3 Size and shape of 3D-BSSG

Choose an appropriate size of the 3D-BSSG domain is very important to improve the efficiency of algorithms based on it. If it is too large, the running time is not acceptable. On the other hand, if it is too small, many solutions are unlikely to be reached. Naturally, the optimal solution is very likely just in those disabled solutions.

In our experiments, assuming p, q, and r of 3D-BSSG$_{p\times q\times r}$ are equal, the range from $5\times5\times5$ to $10\times10\times10$ is proved to be a proper choice for problems whose amount of modules is less than 100.

In fact, 3D-BSSG does not always need to be a regular cuboid shape. If the chip of the temporal problem is not a regular shape, L shape for instance, the 3D-BSSG structure can also be reshaped to meet the shape demand. This kind of reshaping can easily be achieved by prohibiting modules to be assigned to some 3D-BSSG rooms which are out of the required shape.

Besides, if the ratio between chip area and duration is given to obey, the 3D-BSSG structure can also easily meet it. What we need to do is simply to adjust the p, q, and r of 3D-BSSG$_{p\times q\times r}$ until the ratio between p, q, and r is equal to the ratio asked to obey.

The above two merits of 3D-BSSG cannot be easily achieved by other 3D placement representation, such as 3D-subTCG.

2.4 Solution perturbation

In 3D-subTCG, five operations should be performed to guarantee thorough perturbation of the solution space. They are rotation, move, swap, reverse and transpositional move. To maintain the properties of a 3D-subTCG, the resulting graphs must be updated after performing reverse, move and transpositional move.

In 3D-BSSG, the solution perturbation is much easier. The only operation to perturb an assignment in 3D-BSSG is executed by swapping the contents of randomly chosen two rooms. Rotation of a module when it is swapped is also freely allowed. In each perturbation, we also perform feasibility detection as well as Ref. [6] to guarantee no violation against precedence constraints.

3 Using solution space smoothing for temporal floorplanning

The basic idea of solution space smoothing is to gradually guide the course of local search from smoothed solution spaces to rougher ones. Initially, a simplified placement instance with a smooth terrain surface is solved. Then a more complicated placement instance that has a rougher terrain surface is generated. It takes the solution of the previously solved placement as an initial placement and further improves the placement. Eventually, the original placement instance with the most complicated search space structure is solved. The solutions of the simplified problem instances are used to guide the search of more complicated ones[10].

There are various approaches to transform the original placement instance into a series of placement instances by size changing[8,10]. Here, we adopt a simple strategy as work in Ref. [10]. Let PI be the original placement instance with n modules, and PI^0 be a placement instance where all of its modules have the same size and duration as the smallest module in PI. From PI^0, we slightly change the size and duration of all the modules in PI^0 simultaneously and produce PI^1. In the same way, from PI^1 we produce PI^2, and so on. Thus, we obtain a smoothed sequence PI^0, PI^1, PI^2, PI^3, PI^4, … where the solution space changes slightly from smoothness to toughness.

In the simple placement instance, due to the same size and duration of all the modules, the solution space is flattened and has much less local minimum points. We use it as the initial smoothed solution space for our temporal problem. The size (w_{init} and h_{init}) and duration (t_{init}) can be calculated by formula below.

$$w_{init} = \min\{w_1, \cdots, w_i, \cdots, w_n\}$$

$$h_{init} = \min\{h_1, \cdots, h_i, \cdots, h_n\}$$

$$t_{init} = \min\{t_1, \cdots, t_i, \cdots, t_n\}$$

Then we can create a series of simplified placement instances by following the formula:

$$S(\alpha):$$

$$w_i(\alpha) = w_{init} + (w_i - w_{init})^{\alpha}$$

$$h_i(\alpha) = h_{init} + (h_i - h_{init})^{\alpha}$$

$$t_i(\alpha) = t_{init} + (t_i - t_{init})^{\alpha}$$

where α, which should be reduced by a function $f(\alpha)$, is a key parameter to lead the solution space smoothing process. In order to assure that the size and duration of each module changes in a slightly, gradually and monotonously increasing mode, items (w_i, h_i, t_i) in the above formula should be normalized to the range (0, 1). After this normalization, we can see that a larger α can generate a smoother solution space while a smaller α can generate a rugged one. When $\alpha \gg 1$, the size and duration of each module will be reduced to the initial one; when α drops to 1, each module will be its original one.

In fact, it is not necessary to search from a very large α. The initial value of α is proper as long as it can make the initial solution smooth enough. In most of our experiment we usually set this value to 5 and reduce it by the function $f(\alpha)$ below:

$$f(\alpha) = \begin{cases} 0.8\alpha, & \alpha > 2 \\ 0.9\alpha, & \alpha > 1.2 \\ 0.98\alpha, & \alpha > 1 \end{cases}$$

This $f(\alpha)$ is not the only choice to decrease the value of α. It could and should be adjusted to be suitable for a particular problem in order that not only the final solution quality would be improved but also the CPU time of the process would be reduced.

4 Algorithm

The cost function Φ used in our algorithm is given by

$$\Phi = V + \beta W + \gamma O$$

where V is the volume (the minimum bounding box) of the placement. We need this because not only the area of a device but also the total execution time should be considered. W is the total wirelength (the summation of a half bounding box of interconnections) and O is the reconfiguration and communication overheads, both of them can be computed in the similar way as Refs. [4,6]. β and γ are user-specified weights for different problems.

Our algorithm is detailed below:

Step 1: $\alpha := \alpha_0$; create the initial problem instance according to the smoothing function $S(\alpha)$.

Step 2: Make a initial assignment A_0 using some heuristic method and take it as the initial solution. Then, get the placement through BSSG_To_Placement procedure and calculate the cost value.

Step 3: Change A_0 to another assignment A_1 by swapping the contents of two randomly chosen rooms, then get the placement and calculate the cost value similarly as step 2. If the cost value is better, save current assignment as A_0. Otherwise, restore the assignment to A_0.

Step 4: If A_0 is considered, by some rule, as the optimal assignment for current problem instance, save the result as a current solution. Otherwise, go to step 3.

Step 5: If $\alpha = 1$, stop. The current assignment is the final solution. In order to get the corresponding final placement, simply use BSSG_To_Placement again. Otherwise, $a := f(\alpha)$; apply the smoothing function to get the next problem instance.

Step 6: Using the saved current assignment as the starting solution for the next instance and go to step 3.

In step 4, the rule of regarding an assignment as an optimal one is to check whether there is any improvement in a certain number of searches. If nothing, the assignment is taken as an optimal one.

The time complexity of out algorithm can be estimated as: $A \times (O(UNS + N_e) \times O(pqr))$, where A is the number of searched solution space, which is determined by the initial value of α and the decreasing function $f(\alpha)$, UNS is the uncertain number of searches before the search process reach the solution, N_e is the number of searches used to ensure the solution reached is an optimal or nearly optimal one and $O(pqr)$ is the time complexity of early mentioned BSSG_To_Placement procedure.

5 Results

We implemented our temporal floorplanning algorithm in the C++ programming language in the Linux environment of a PC (Intel P4 CPU and 512M memory).

Moreover, we compared our results with those of 3D-subTCG[6] and Sequence Triplet (ST) which is extended from the well known sequence pair (SP)[12] by performing three experiments. β and γ are both set to 1 in each experiment. The benchmarks we adopted in these experiments came from those used in Refs. [6,13]. Reference [6] has adapted some of them. For example, some benchmarks were added with reconfiguration and communication overheads and some were added with execution time and precedence constraints. Similar adaptation was also done by us appropriately to make the comparison fair.

In the first experiment, our objective is to minimize the volume with reconfiguration and communication overheads. In order to verify our 3D-BSSG structure itself, we tested it based on simulated annealing like Ref. [6]. As shown in Table 2, the 3D-BSSG structure based method outperforms both the ST based one and 3D-subTCG based one, which can demonstrate the effectiveness of our 3D-BSSG structure to obtain volume optimization.

Table 2 Results for volume optimization with reconfiguration and communication overheads using 3D-BSSG based simulated annealing

Circuit	# of modules	Sum of volume	ST		3D-subTCG		3D-BSSG-SA	
			Volume	Dead space	Volume	Dead space	Volume	Dead space
okp 1	50	1.24×10^8	2.16×10^8	42.6%	1.73×10^8	28.4%	1.65×10^8	24.8%
okp 2	30	8.54×10^7	1.28×10^8	33.2%	1.10×10^8	22.3%	1.09×10^8	21.7%
okp 3	30	1.23×10^8	1.85×10^8	33.1%	1.60×10^8	23.0%	1.56×10^8	20.7%
okp 4	61	2.38×10^8	4.17×10^8	42.8%	3.28×10^8	27.3%	3.07×10^8	22.3%
okp 5	97	1.89×10^8	4.48×10^8	57.7%	2.95×10^8	35.8%	2.38×10^8	20.4%
average				41.88%		27.36%		21.98%

The objective of the second experiment is the same as that of the first experiment ,but this time , we used our solution space smoothing algorithm instead of simulated annealing. Table 3 shows the results. Again ,the effectivity and efficiency of our algorithm are exhibited.

The third experiment is intended to test the 3D placement with the considerations of precedence constraints , wirelength , and reconfiguration/ communication overheads. The results of this part are also comparable ,as shown in Table 4. And the best results of 3D-ami49 is presented in Fig. 6.

Table 3 Results for same objectivity as Table 2 using 3D-BSSG based solution space smoothing

Circuit	# of modules	Sum of volume	ST		3D-subTCG		3D-BSSG-SSS	
			Volume	Dead space	Volume	Dead space	Volume	Dead space
beasley 1	10	6218	8710	28.6 %	7504	17.1 %	7504	17.1 %
beasley 2	17	11497	14664	21.5 %	12402	7.2 %	12456	7.7 %
beasley 3	21	10362	16016	35.3 %	12640	18.0 %	12166	14.8 %
beasley 4	7	10205	13800	26.0 %	13064	21.8 %	12490	18.3 %
beasley 5	14	16734	22750	26.4 %	18912	11.5 %	18994	11.9 %
beasley 6	15	11040	14994	26.3 %	13200	16.3 %	13333	17.2 %
beasley 7	8	17168	24570	30.1 %	20574	16.5 %	20574	16.5 %
beasley 8	13	83044	132275	37.2 %	98280	15.5 %	99216	16.3 %
beasley 9	18	133204	174496	23.6 %	167751	20.5 %	167751	20.5 %
beasley 10	13	493746	660480	25.2 %	575685	14.2 %	583624	15.4 %
beasley 11	15	383391	486381	24.8 %	438702	12.6 %	441186	13.1 %
beasley 12	22	646158	922080	29.9 %	823816	21.5 %	792360	18.5 %
average				27.91 %		16.06 %		15.61 %

Table 4 Results of volume and wirelength optimization using 3D-BSSG based solution space smoothing for 3D-MCNC benchmark circuits

Circuit	Sum of volume	ST			3D-subTCG			3D-BSSG-SSS		
		Volume	Wire length	Dead space	Volume	Wire length	Dead space	Volume	Wire length	Dead space
3D-apte	9.88×10^{7}	1.18×10^{8}	495.0	16.2 %	1.05×10^{8}	335.3	5.9 %	1.10×10^{8}	359.3	10.2 %
3D-xerox	4.05×10^{7}	5.27×10^{7}	613.2	23.1 %	4.42×10^{7}	602.0	8.4 %	4.52×10^{7}	607.3	10.4 %
3D-hp	1.29×10^{7}	2.06×10^{7}	387.3	37.2 %	1.50×10^{7}	158.3	13.7 %	1.47×10^{7}	153.2	12.3 %
3D-ami33	2.32×10^{6}	4.18×10^{6}	84.7	44.5 %	3.08×10^{6}	77.7	24.7 %	2.88×10^{6}	70.4	19.5 %
3D-ami49	1.32×10^{8}	2.93×10^{8}	1040.8	54.9 %	1.68×10^{8}	807.1	21.6 %	1.57×10^{8}	754.6	15.9 %
average				35.18 %			14.86 %			13.66 %

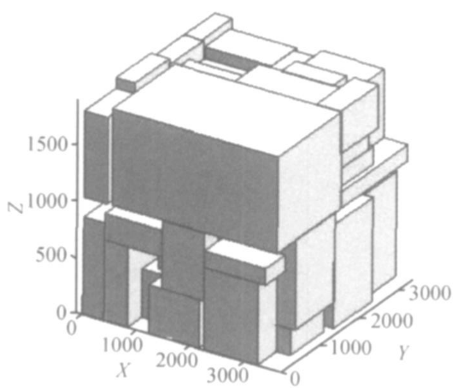

Fig. 6 Best results of 3D-ami49 volume usage is about 84. 9 %.

6 Conclusion

We have presented an effective 3D-BSSG structure based solution space smoothing algorithm to solve temporal floorplanning problems for dynamically reconfigurable FPGAs. First ,we have developed a 3D-BSSG structure to represent the placement in such problems. Then ,we used the solution space smoothing algorithm to search for the optimal solution. Compared with the simulated annealing algorithm ,it uses fewer parameters to control the search process. Experimental results have shown that our method is very effective and efficient for temporal floorplanning problems.

References

[1] Bazargan K, Kastner R, Sarrafzadeh M. Fast template placement for reconfigurable computing systems. IEEE Design & Test of Computers, 2000, 17(1): 68

[2] Bazargan K, Kastner R, Sarrafzadeh M. 3-D floorplanning: simulated annealing and greedy placement methods for reconfigurable computing systems. Design Automation for Embedded Systems-RSP '99 Special Issue, 2000

[3] Teich J, Fekete S P, Schepers J. Compile-time optimization of dynamic hardware reconfiguration. Proc of PDPTA, 1999: 1097

[4] Fekete S P, Kohler E, Teich J. Optimal FPGA module placement with temporal precedence constraints. Proc of DATE, 2001: 658

[5] Bazargan K, Sarrafzadeh M. Fast online placement for reconfigurable computing systems. IEEE Symposium on FPGAs for Custom Computing Machines, 1999: 300

[6] Yuh P H, Yang C L, Chang Y W, et al. Temporal floorplanning using 3D-subTCG. Proc Of ASP-DAC, 2003

[7] Nakatake S, Fujiyoshi K, Murata H, et al. Module placement on BSG-structure and IC layout applications. Porc of ICCAD, 1996: 484

[8] Gu Jun, Huang Xiaofei. Efficient local search with search space smoothing: a case study of the traveling salesman problem (TSP). IEEE Trans Systems Man And Cybernetics, 1994, 24(5): 728

[9] Schneider J, Dankesreiter M, Fettes W, et al. Search-space smoothing for combinatorial optimization problems. Phys A, 1997, 243: 77

[10] Dong Sheqin, Hong Xianlong, Qi Xin, et al. VLSI module placement with pre-placed modules and considering congestion using solution space smoothing. ACM/IEEE ASP-DAC, Japan, 2003: 741

[11] Lawler E. Combinatorial optimization: networks and matroids. Holt, Rinehart, and Winston, 1976

[12] Murata H, Fujiyoshi K, Nakatake S, et al. Rectangle-packing based module placement. Proc of ICCAD, 1995: 472

[13] Fekete S P, Kohler E, Schepers J. On more-dimensional packing Ⅲ: exact algorithms. ZPR Technical Report, 1997: 97

[14] Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by simulated annealing. Science, 1983, 220: 671

用基于 3D-BSSG 结构的解空间平滑算法解决时序规划问题

郑舒一 董社勤 洪先龙

(清华大学计算机科学与技术系, 北京 100084)

摘要: 可动态配置的 FPGA 电路的出现产生了时序规划问题. 如果把时间看作第三维度,那么该问题可转化为三维布局问题. 本文提出了一个全新的三维受限切面网格结构(3D-BSSG),用来表示三维布局的解;并引入解空间平滑机制来搜索最优解。实验结果证明,所设计的基于 3D-BSSG 的算法在求解时序规划问题上是十分有效的.

关键词: 时序规划; FPGA; 三维受限切面网格结构; 解空间平滑

EEACC: 7140D; 5120

中图分类号: TN47 **文献标识码**: A **文章编号**: 0253-4177(2005)10-1916-09

2005-02-18 收到,2005-06-27 定稿