VLSI - Very Large Scale Integration

Combinatorial Decision Making and Optimization
Module 1

Dahesh, Parsa (dahesh.parsa@studio.unibo.it)
Granata, Ludovico (ludovico.granata@studio.unibo.it)
Persiani, Simone (persiani.simone2@studio.unibo.it)

Academic year 2020-2021

Abstract

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. A typical example is the smartphone. The modern trend of shrinking transistor sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon die (i.e. plate). This enabled the modern cellphone to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen, and other advanced features.

1 SATisfiability

1.1 Model setup

In our proposed solution, each instance file is initially parsed in order to extract the following constants:

- $n_{circuits}$, the amount of circuits to be packed inside the grid;
- w_{grid}, the width of the grid;
- w_i and h_i with $i \in [0, n_{circuits})$, the width and height of each circuit.

Considering that one of the assigned goals is the minimization of the packing height and that –at the same time– we are interested in reducing the amount of variables and related constraints for performance reasons, we chose to limit the height of the grid to the optimal packing height as computed here:

$$h_{opt} = \frac{\sum_{i=0}^{n_{circuits}-1} w_i \cdot h_i}{w_{grid}} \tag{1}$$

h_{opt} is the packing height in case no gap is left between the circuits and each of them is aligned in such a way that they collectively reach the lowest possible height. We have found that this choice is not too big of a limitation for the solver, as each provided instance is solvable within this constraint.

1.2 The variables

We started modelling the problem by defining the Boolean variables needed to represent its domain. After having considered a couple of other options, we chose to identify the position of each circuit inside the grid with the pair of coordinates in which its bottom-left corner can be placed. This led us to the definition of the following variables:

$$v_{i,n,m}$$

, where $i \in [0, n_{circuits})$ is the index of the circuit, while n and m represent (respectively) the horizontal and vertical coordinates of the bottom-left corner of the circuit. $v_{i,n,m}$ is true if the bottom-left corner of the i-th circuit is placed at the point (n, m) inside the grid.

1.2.1 Rotation model

In order to consider also the 90°-rotated version of each circuit, the following variables have to be defined:

$$r_i$$

, where $i \in [0, n_{circuits})$ is the index of the circuit. r_i is true if the i-th circuit is rotated by 90°.

1.3 Positioning constraints

Each circuit must be placed in some position inside the grid and –at the same time– it cannot be placed in more than one position. Such a statement can be encoded in SAT by looping over all the possible (n, m) positions for the i-th circuit, in order to apply the following constraint:

$$\bigwedge_i exactly\_one(i) =$$

$$\bigwedge_i \left[\left(\bigvee_{n,m} v_{i,n,m} \right) \wedge \left(\bigwedge_{(n,m)\neq(p,q)} \neg \left(v_{i,n,m} \wedge v_{i,p,q} \right) \right) \right] \tag{2}$$

An additional constraint is implicitly enforced by limiting the possible positions to only those that result in the circuit being completely contained inside the grid. In this way, the grid boundaries are always respected:

$$\mathcal{H}_i = [0, w_{grid} - w_i], \mathcal{V}_i = [0, h_{opt} - h_i]$$

$$(n, m), (p, q) \in \mathcal{H}_i \times \mathcal{V}_i$$

1.3.1 Rotation model

In the variant of the model that takes into consideration also the possibility to rotate the circuits by 90°, the previous constraint has to be split in two as follows:

$$\neg r_i \longrightarrow \bigwedge_i exactly\_one(i) \tag{3}$$

$$r_i \longrightarrow \bigwedge_i exactly\_one(i) \tag{4}$$

In case of equation 3 the domain of possible positions is $\mathcal{H}_i \times \mathcal{V}_i$, while in case of equation 4 the domain must be revised as follows (since rotating by 90° a circuit means swapping its dimensions):

$$\mathcal{H}_i^R = [0, w_{grid} - h_i], \mathcal{V}_i^R = [0, h_{opt} - w_i]$$

$$(n, m), (p, q) \in \mathcal{H}_i^R \times \mathcal{V}_i^R$$

For each squared circuit, we force it not to be rotated in order to avoid unnecessary work for the solver:

$$\bigwedge_{w_i = h_i} \neg r_i \tag{5}$$

1.4 Non-overlapping constraints

No pair of circuits should ever overlap inside the grid. Since this kind of constraint is inherently symmetrical, we apply it to every possible pair of circuits (i, j) with $i < j$ (so to be sure not to apply the same constraint to the same two blocks twice). We chose to encode this constraint in SAT by reasoning as follows: if the i-th circuit is placed at (n, m), then there are some positions (p, q) in which the j-th circuit cannot be placed, as in such cases it would overlap with i.

$$\bigwedge_{i<j} v_{i,n,m} \longrightarrow \neg v_{j,p,q} \tag{6}$$

Such forbidden positions are determined by considering both the dimensions of the j-th circuit and those of the i-th. The resulting forbidden range is then intersected with the actual positioning domain of the j-th circuit.

$$(n, m) \in \mathcal{H}_i \times \mathcal{V}_i$$
$$p \in [n - w_j + 1, n + w_i - 1] \cap \mathcal{H}_j$$
$$q \in [m - h_j + 1, m + h_i - 1] \cap \mathcal{V}_j$$

1.4.1 Rotation model

When rotations are allowed, the previous constraint has to be split into four variants (one for each possible combination of r_i and r_j values) and both the domain for (n, m) and the forbidden ranges for (p, q) have to be re-computed:

$$\bigwedge_{i<j} v_{i,n,m} \wedge \neg r_i \wedge \neg r_j \longrightarrow \neg v_{j,p,q} \tag{7}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge \neg r_i \wedge r_j \longrightarrow \neg v_{j,p,q} \tag{8}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge r_i \wedge \neg r_j \longrightarrow \neg v_{j,p,q} \tag{9}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge r_i \wedge r_j \longrightarrow \neg v_{j,p,q} \tag{10}$$

For example, in case of equation 9:

$$(n, m) \in \mathcal{H}_i^R \times \mathcal{V}_i^R$$
$$p \in [n - w_j + 1, n + h_i - 1] \cap \mathcal{H}_j$$
$$q \in [m - h_j + 1, m + w_i - 1] \cap \mathcal{V}_j$$

1.5 Symmetry-breaking constraints

While trying to break as many symmetries as possible in the model, we had to balance the effectiveness with the complexity of the constraints to be implemented, as complex constraints can require a combinatorial amount of time to be generated. Having evaluated many other options, we settled on the following ideas:

1. positioning the biggest circuit in the bottom-left corner of the grid, namely $(n, m) = (0, 0)$;
2. when two circuits with the same horizontal dimensions are vertically stacked on top of each other, the bigger one should stay in the bottom;
3. when two circuits with the same vertical dimensions are horizontally stacked next to each other, the bigger one should stay on the left.

For point 1) is sufficient to implement this simple constraint:

$$v_{k,0,0} \tag{11}$$

$$k = \underset{i}{\operatorname{argmax}}\, w_i \cdot h_i$$

For points 2) and 3), we chose to implement a set of constraints very similar to those used for avoiding overlaps, in which we loop over every possible pair of circuits (i, j) with $i < j$ (so to be sure not to apply the same constraint to the same two blocks twice). We chose to encode such statements in SAT by reasoning as follows, assuming that the i-th circuit is placed at (n, m):

1. if the i-th circuit is bigger than (or equal in size to) the j-th circuit, then:
   (a) if $w_i = w_j$, then (p, q) cannot be such that the two circuits are vertically stacked with i on top of j;
   (b) if $h_i = h_j$, then (p, q) cannot be such that the two circuits are horizontally stacked with i on the right of j;
2. if the i-th circuit is smaller than the j-th circuit, then:
   (a) if $w_i = w_j$, then (p, q) cannot be such that the two circuits are vertically stacked with j on top of i;
   (b) if $h_i = h_j$, then (p, q) cannot be such that the two circuits are horizontally stacked with j on the right of i.

$$\bigwedge_{i<j} v_{i,n,m} \longrightarrow \neg v_{j,p,q} \tag{12}$$

For example, in case of situation 2.b) (if $n + w_i \in \mathcal{H}_j$):

$$(n, m) \in \mathcal{H}_i \times \mathcal{V}_i$$

$$(p, q) = (n + w_i, m)$$

1.5.1 Rotation model

The symmetry-breaking constraints shown above become more complex when considering also the possibility to rotate the circuits by 90°. All of the above 4 different situations must be split into other 4 subcases (one for each possible combination of r_i and r_j values):

$$\bigwedge_{i<j} v_{i,n,m} \wedge \neg r_i \wedge \neg r_j \longrightarrow \neg v_{j,p,q} \tag{13}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge \neg r_i \wedge r_j \longrightarrow \neg v_{j,p,q} \tag{14}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge r_i \wedge \neg r_j \longrightarrow \neg v_{j,p,q} \tag{15}$$

$$\bigwedge_{i<j} v_{i,n,m} \wedge r_i \wedge r_j \longrightarrow \neg v_{j,p,q} \tag{16}$$

For example, in case of situation 2.b) with $r_i \wedge \neg r_j$ (if $n + h_i \in \mathcal{H}_j$):

$$\bigwedge_{i<j} v_{i,n,m} \wedge r_i \wedge \neg r_j \longrightarrow \neg v_{j,p,q}$$

$$(n, m) \in \mathcal{H}_i^R \times \mathcal{V}_i^R$$
$$(p, q) = (n + h_i, m)$$

2 Results and Tables

Here are the tables with the final results[1] (timeout is set to 5 minutes, results are computed as the average of 3 runs):

Standard model							
Ins.	CP	SMT	SAT	Ins.	CP	SMT	SAT
1	0.295s	0.016s	0.294s	21	1.162s	timeout	timeout
2	0.297s	0.018s	0.572s	22	1.377s	timeout	timeout
3	0.300s	0.032s	2.158s	23	2.071s	8.942s	timeout
4	0.311s	0.039s	3.363s	24	0.655s	7.399s	timeout
5	0.307s	0.046s	6.707s	25	timeout	timeout	timeout
6	0.316s	0.071s	11.254s	26	2.020s	57.975s	timeout
7	0.308s	0.070s	11.510s	27	0.901s	20.394s	timeout
8	0.320s	0.100s	15.745s	28	0.956s	53.038s	timeout
9	0.317s	0.086s	23.150s	29	2.261s	65.314s	timeout
10	0.352s	0.332s	45.351s	30	timeout	timeout	timeout
11	22.743s	timeout	timeout	31	0.861s	8.249s	timeout
12	0.452s	0.688s	170.593s	32	timeout	timeout	timeout
13	0.409s	0.541s	timeout	33	0.668s	21.927s	timeout
14	0.435s	1.772s	timeout	34	timeout	timeout	timeout
15	0.456s	0.983s	timeout	35	40.135s	timeout	timeout
16	9.154s	timeout	timeout	36	43.464s	timeout	timeout
17	1.377s	5.479s	timeout	37	timeout	timeout	timeout
18	0.565s	6.516s	timeout	38	timeout	timeout	timeout
19	1.045s	timeout	timeout	39	timeout	timeout	timeout
20	1.400s	133.834s	timeout	40	timeout	timeout	timeout

[1]Run with CPU: AMD Ryzen 5 3600; GPU: NVIDIA GeForce GTX 1660 Super; RAM: 16GB; OS: Windows 10

Rotation model							
Ins.	CP	SMT	SAT	Ins.	CP	SMT	SAT
1	0.300s	0.019s	1.902s	21	timeout	timeout	timeout
2	0.311s	0.027s	3.998s	22	timeout	timeout	timeout
3	0.315s	0.047s	11.905s	23	timeout	timeout	timeout
4	0.318s	0.149s	23.888s	24	2.910s	timeout	timeout
5	0.325s	0.476s	186.306s	25	timeout	timeout	timeout
6	0.352s	0.781s	270.830s	26	timeout	timeout	timeout
7	0.446s	1.045s	timeout	27	7.723s	timeout	timeout
8	0.350s	0.826s	137.327s	28	timeout	timeout	timeout
9	18.657s	2.784s	timeout	29	timeout	timeout	timeout
10	1.774s	133.357s	timeout	30	timeout	timeout	timeout
11	timeout	timeout	timeout	31	timeout	timeout	timeout
12	2.256s	timeout	timeout	32	timeout	timeout	timeout
13	1.739s	timeout	timeout	33	timeout	timeout	timeout
14	timeout	timeout	timeout	34	timeout	timeout	timeout
15	3.192s	timeout	timeout	35	timeout	timeout	timeout
16	timeout	timeout	timeout	36	timeout	timeout	timeout
17	timeout	timeout	timeout	37	timeout	timeout	timeout
18	timeout	timeout	timeout	38	timeout	timeout	timeout
19	timeout	timeout	timeout	39	timeout	timeout	timeout
20	timeout	timeout	timeout	40	timeout	timeout	timeout