Global Weighted Scheduling and Allocation Algorithms

Houria Oudghiri and Bozena Kaminska
Ecole Polytechnique de Montréal, Electrical Engineering Department
P.O. Box 6079, Station A, Montréal, QC, H3C 3A7

Abstract

Scheduling and allocation are very complex problems in a high-level synthesis system. It was proven, in related work, that the two are NP-Complete optimization problems. In this paper, we introduce a new global approach for scheduling and allocation. The approach uses graphs to formulate the two problems and applies a partitioning procedure on these graphs to find the minimal number of cliques. The obtained cliques correspond to the time steps in scheduling and to hardware elements required in allocation. The partitioning procedure is made more efficient by weighting the graph edges by the profit to group nodes together. The procedure was programmed in C++ and experimental results are given to show its efficiency to solve both scheduling and allocation.

1. Introduction

High-level synthesis is the automatic compilation of a high-level behavioral description of a digital system into a register transfer structure. The behavioral description means the mapping from the system inputs to its outputs. This behavioral description is made in a high-level hardware description language like VHDL, ISPS, or C. The structural description refers to the set of interconnected components that make up the system. To a given behavior, there are always many corresponding structures. The major task of high-level synthesis is to find the structure that meets designer constraints like area, delay time, power consumption and testability.

The first step in high level synthesis is to compile the behavioral description into an internal representation which is always a data flow graph. The purpose of all the remaining steps is to compile this data flow graph into a structure (a data path and a control path). The main tasks to perform are the scheduling and the allocation.

In this paper, we will present a new global method for performing scheduling and allocation tasks. The method takes advantage of heuristics and exhaustive solutions. The method is based on graph theory since scheduling and allocation problems are formulated as graphs. The same formulation and the same partitioning procedure are applied to both problems. Our program is called GLOW which stands for GLObal and Weighted.

2. Related research

In this section, a taxonomy of various different techniques commonly used has been realized according to [7] but has been updated to include the most recent systems as shown in Fig.1.

GLOW is classified as a global allocation and transformational-iterative/constructive scheduling. Indeed, the global allocation which is like transformational scheduling technique has been used to solve scheduling too. The approach considers one operation at each partitioning step which trains us to classify our scheduling approach between transformational and constructive/iterative techniques. How are scheduling and allocation performed by GLOW, that is what next sections will show.

3. Weighted-graph partitioning procedure

Teng and Siewiorek were the first to apply the graph partitioning procedure FACET [3] to the allocation problem. For each type of element to be allocated, a dependency graph G=(E,V) is created, where E is the set of nodes corresponding to the elements to be allocated (operations, variables or interconnections) and V the set of edges. An edge (i, j) is created between the elements i and j if i and j can share the same hardware unit.

The next step is the partitioning of the graph into a minimum number of cliques. The graph partitioning
Scheduling
Interconnection
between microlane-
enabled scheduling
Memory [1986]
SLICER [1987]

Fig. 1: Taxonomy of the various scheduling and allocation techniques

is considered as an NP-complete problem [9]. To make the problem less complex, heuristics are used. Tseng and Siewiorek in [3] used the neighborhood property heuristic.

According to its authors, this procedure can generate sub-optimal solutions because the neighborhood property is not sufficient. A more efficient procedure was also introduced in [10] for the allocation problem, where the edges are weighted by the value of the area saved if the edge nodes are combined.

Our method is based on these two techniques and can be applied to scheduling as well as to allocation. We construct a dependency graph for each problem to be solved, i.e. scheduling, functional unit allocation, register allocation and interconnection unit allocation. The graph edges are then weighted, these weights being a general cost function. The weight of an edge \((i, j)\) is the cost or the profit involved by combining \(i\) and \(j\). This cost function can be more detailed when more low-level details are available. Indeed, when a cell library is available, cost like area and delay can be computed to show the advantage to combine two elements. The partitioning procedure used by GLOW is presented in Fig.2. When more than one edge is found with the maximum weight, we select the edge that has the maximum gain. The gain corresponds to the notion of common neighbors, but extended to weights. The gain of an edge \((i, j)\) is computed according to the following expression:

\[
\text{Gain}(i, j) = N \sum_{j=1}^{w_i} - M \sum_{k=1}^{w_j},
\]

where \(N\) is the number of edges to be preserved, \(M\) the number of edges to be deleted when \(i\) and \(j\) are combined and \(W_k\) is the weight of the edge \(k\).

4. Scheduling problem

During scheduling, each operation of the data flow graph must be assigned to a time step in such a way as to minimize the execution time (i.e. the number of time steps). Each time step will contain a list of operations. Operations that are assigned to the same time step will be executed in parallel. To each operation, two values are assigned, ASAP (As Soon As Possible) and ALAP (As Late As Possible) values. The ASAP and ALAP values determine the earliest and the latest time steps for each operation. Operations that have equal ASAP and ALAP values are on the critical path, since no choice in their scheduling is available unless the number of time steps is increased. In Fig 3, operations 0, 1, 5, 8 and 9 are all critical. For each non-critical operation, these ASAP and ALAP values define the interval of its possible schedulings.

Based on these values, an independency graph is constructed. The nodes of this graph are the operations and edges are created between each two operations that can execute in parallel (i.e. the two operations are independent or their ASAP and ALAP intervals are not disjoint). Then, the edges are weighted by a profit function. Initially, all edges are weighted by the maximum cost of the edges to be deleted.
Scheduling procedure

Begin
1. Assign to each operation the data flow graph in ASAP and ASAP values
2. Construct the independency graph between operations.
3. Assign all the critical operations to their respective time steps.
4. Update the graph according to the resulting cliques.
5. While (there is operations not yet assigned)
   a. Scan through the list of edges that contain a node corresponding to an existing clique. Select the edge which has the maximum weight (i, k)
   b. Update the graph by combining i and k.
   c. Update the ASAP value of k's successors.
   d. If there is operations that have become critical, assign them to their respective time steps.
End

9. In Fig 3, to their respective time steps defines cliques and a reduced graph.

The second part of the scheduling procedure considers the remaining operations. We consider the list of edges containing a node which belongs to an existing clique. We then select the edge that has the maximum weight (i, k) in the established list. Combining i and k means that i is assigned to the same time step as k, if k is the node already assigned. This is repeated until all operations are assigned. When an operation is assigned, we consider its successors in order to verify whether or not they have become critical.

The final schedule of the example requires 4 time steps, 2 multipliers, 1 adder, 1 subtractor and 1 comparator as hardware resources, which better than the results obtained in related works [1] and [6].

5. Allocation problem

The allocation problem is one of finding a minimum number of element sets that correspond to the required hardware units (an element is an operation, a variable or a data interconnection). The partitioning of a graph

Fig. 5: Scheduling procedure by partitioning weighted graphs

List of edges considered: \( (0, 2)^6 (0, 3)^6 \\
(1, 2)^6 (1, 3)^6 \\
(2, 3)^6 (2, 4)^3 (2, 5)^3 \\
(3, 4)^3 (3, 5)^3 (3, 6)^1 \\
(4, 5)^3 (4, 6)^3 (4, 7)^3 (4, 8)^3 \\
(5, 6)^7 (5, 7)^7 \\
(6, 7)^9 (6, 8)^9 \\
(7, 8)^3 (7, 9)^3 

The updated graph

\( (0, 2)^6 (0, 3)^6 \)
\( (1, 2)^6 (1, 3)^6 \)
\( (2, 3)^6 (2, 4)^3 (2, 5)^3 \)
\( (3, 4)^3 (3, 5)^3 (3, 6)^1 \)
\( (4, 5)^3 (4, 6)^3 (4, 7)^3 (4, 8)^3 \)
\( (5, 6)^7 (5, 7)^7 \)
\( (6, 7)^9 (6, 8)^9 \)
\( (7, 8)^3 (7, 9)^3 \)

9. In Fig 3, to their respective time steps defines cliques and a reduced graph.

The second part of the scheduling procedure considers the remaining operations. We consider the list of edges containing a node which belongs to an existing clique. We then select the edge that has the maximum weight (i, k) in the established list. Combining i and k means that i is assigned to the same time step as k, if k is the node already assigned. This is repeated until all operations are assigned. When an operation is assigned, we consider its successors in order to verify whether or not they have become critical.

The final schedule of the example requires 4 time steps, 2 multipliers, 1 adder, 1 subtractor and 1 comparator as hardware resources, which better than the results obtained in related works [1] and [6].

5. Allocation problem

The allocation problem is one of finding a minimum number of element sets that correspond to the required hardware units (an element is an operation, a variable or a data interconnection). The partitioning of a graph
into a minimum number of cliques fits in well with the allocation problem. Dependency graphs are constructed for each type of element to be allocated. Two elements are connected by an edge in the dependency graph if they can share the same hardware unit. Once the graph is constructed, its edges are weighted in such a way that each weight corresponds to the profit obtained when the two nodes of the edge are allocated to the same unit. The weighting function depends on the type of elements to be allocated. A detailed description of the register allocation, functional unit allocation and interconnection allocation can be found in [13].

As shown above, scheduling is done first and then allocation, but this does not mean that the two tasks are performed separately. Indeed, during scheduling, weights are determined considering a pre-allocation that estimates the cost of different possible allocations. The same principle is kept during allocation since each type of allocation is performed taking into account the other following types of allocation. It is why we called our algorithm global.

6. Experimentation

Our scheduling and allocation algorithms were compared to related methods by considering examples used in most papers: the differential equation which we used in the previous sections, the FACET example [3] in Fig. 7 and the algorithm of a fifth-order elliptic wave filter extracted from [11] and presented in Fig. 8. Note that the fifth-order elliptic filter is among high-level synthesis benchmarks presented in [12].

Table I shows the results for the differential equation example compared to related works [1] and [6]. HAL89 [5] found 2 multipliers, an adder, a subtractor, a comparator and 6 registers, and the same functional units have been required by SPLICER, while, with GLOW, we were able to combine the adders and the subtractors. The design has been generated in a CPU time of 4 seconds.

The results obtained by GLOW for the second example, the FACET one, are shown in Table II. In the comparison with results presented in [4], [1] and [2], we generate a better distribution for functional unit allocation since both the multiplication and the division are allocated to a separate unit from those of addition, subtraction and boolean operations. This was possible because of the notion of equivalence classes used to weight the graph edges. The Emerald [4], SPLICER [1] and Devadas and Newton system [2], all allocate the addition, the multiplication and boolean operations to the same unit which is not very practical. This is taken into account by our approach since the multiplication belongs to a different equivalence class from that of the addition and boolean operations. Only 1 second of CPU time is needed to generate the result.

The last example is the fifth-order elliptic wave filter shown in Fig. 8 [11]. Table III shows a comparison of GLOW results with the results of other systems. HAL89 result required one less adder and one more multiplier but the multiplier cost is greater than the adder cost. 12 registers were used by HAL89, while we need only 11 registers, so the data path found by GLOW costs less. We cannot compare our results to the Devadas and Newton results, second row of Table III. The reason is that the filter description used in [2] is not the same as the original one described in [11] and shown on Fig. 8. Only 7 multiplications were used in the filter description in [2], which results in allocating only 2 multipliers. In row 4 of Table III, we generate a data path for the description in [2]. The resulting data path is built up of 2 multipliers, 3 adders, 12 registers, 7 buses and executes in 17 cycles.

The functional units were allocated as in [2]. For the
The technique is for N nodes in the graph. Since edges are not oriented, there are at most \( \frac{N^2}{2} \) edges for N nodes in the graph. Thus, the partitioning procedure has a computing complexity of \( O(N^3) \) at worst. The technique is used to solve both allocation and scheduling problems.

**Fig. 8:** Algorithmic description of the elliptic filter

**Table III** Comparison of results for the elliptic filter

<table>
<thead>
<tr>
<th>System</th>
<th>No. of partitions</th>
<th>No. of registers</th>
<th>No. of Buses or Mux</th>
<th>ALUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL89</td>
<td>17</td>
<td>12</td>
<td>6 buses</td>
<td>3(+)</td>
</tr>
<tr>
<td>DEV_NEW</td>
<td>17</td>
<td>NA</td>
<td>NA</td>
<td>3(+)</td>
</tr>
<tr>
<td>GLOW1</td>
<td>17</td>
<td>11</td>
<td>6 buses</td>
<td>4(+)</td>
</tr>
<tr>
<td>GLOW2</td>
<td>17</td>
<td>12</td>
<td>7 buses</td>
<td>3(+)</td>
</tr>
</tbody>
</table>

other components, no results were presented in [2]. The elliptic filter synthesis has been performed in 2 minutes of CPU time.

7. Conclusion

In this paper, we have presented a new global approach to resolve the scheduling and allocation problems in high-level synthesis. The approach is based on a partitioning procedure which is applied to weighted graphs. Weighting graph edges makes the technique more flexible and efficient. The partitioning heuristic has a complexity which is proportional to the number of graph edges. Since edges are not oriented, there are at most \( \frac{N^2}{2} \) edges for N nodes in the graph. Thus, the partitioning procedure has a computing complexity of \( O(N^3) \) at worst. The technique is used to solve both allocation and scheduling problems, which presents the advantage to provide effort only one time to formulate the two problems and then to use the same procedure to perform them.

8. Bibliography


