Force Directed Scheduling

Bin Abdul Malik, Muhammad Amjad Student of Electronic Engineering Hochschule Hamm-LippstadtLippstadt, Germanymuhammad-amjad-bin.abdul-malik@stud.hshl.de

*Abstract*—High-level synthesis raises the design abstraction level and allows rapid generation of optimized RTL hardware for performance, area, and power requirements. This paper gives an overview of one of the scheduling algorithms that has been implemented which Force Directed Scheduling. Some knowledges of high-level synthesis and unconstrained algorithms are needed to have a better understanding on the topic Force Directed Scheduling. Thus, this paper also includes some main important informations on this matter. Important to note that this paper mainly discussing on the hardware implementation.

# Introduction

The growing capabilities of silicon technology and the increasing complexity of applications in recent decades have forced design methodologies and tools to move to higher abstraction levels. Raising the abstraction levels and accelerating automation of both the synthesis and the verification processes have for this reason always been key factors in the evolution of the design process, which in turn has allowed designers to explore the design space efficiently and rapidly both in the software and hardware domain.

In the hardware domain, specification languages and design methodologies have evolved similarly. For this reason, until the late 1960s, ICs were designed, optimized, and laid out by hand. Simulation at the gate level appeared in the early 1970s and cycle-based simulation became available by 1979. Techniques introduced during the 1980s included place-and- route, schematic circuit capture, formal verification, and static timing analysis. Hardware description languages (HDLs), such as Verilog (1986) and VHDL (1987) enables wide adoption of simulation tools. Raising the hardware design’s abstraction level is essential to evaluating system-level exploration for architectural decisions such as hardware and software design, synthesis and verification, memory organization, and power management.

Apart from that, HLS also enables reuse of the same high-level specification, targeted to accommodate a wide range of design constraints and ASIC or FPGA technologies. To put it short, High-level synthesis is the process of converting a high-abstraction-level description of a design to a register-transfer-level (RTL) description for input to traditional ASIC and FPGA implementation workflows. These HDLs have also served as inputs to logic synthesis tools leading to the definition of their synthesizable subsets. During the 1990s, the first generation of commercial high-level synthesis (HLS) tools was avail- able commercially which spark a great research interest on hardware-software including estimation, exploration, partitioning, interfacing, communication, synthesis, and simulation gained momentum.

HLS tools transform an untimed (or partially timed) high-level specification into a fully timed implementation. They automatically or semiautomatically generate a custom architecture to efficiently implement the specification. In addition to the memory banks and the communication interfaces, the generated architecture is described at the RTL and contains a data path (registers, multiplexers, functional units, and buses) and a controller, as required by the given specification and the design constraints [1]. As an aid for a better understanding in terms of the flow, a high-level synthesis (HLS) design steps are shown in the figure below.

Diagram

Description automatically generated

1. High-level synthesis (HLS) design steps.

Starting from the high-level description of an application, an RTL component library, and specific design constraints, an HLS tool executes the following tasks:

1. Compiles the specification
2. Allocates hardware resources (functional units, storage components, buses, and so on)
3. Schedules the operations to clock cycles
4. Binds the operations to functional units
5. Binds variables to storage elements
6. Binds transfers to buses
7. Generates the RTL architecture

# Scheduling Algorithms

Now that we understand the flow of the HLS design steps, we can now move on next to the scheduling algorithms. As mentioned before, scheduling is one crucial step in the behavioural synthesis process. The final timing information may be completely different if two different scheduling techniques are used. Here we briefly illustrate some scheduling techniques mostly used nowadays. The main reason on why it is crucial is because it is used to determine for each assigned operation, the time at which it should be performed or executed without violating any precedence constraints. It is important to note that for a given function, an optimal solution can be constrained by area, power, performance or flexibility requirements depending on the application.

Basically, scheduling consists of determining a propagation delay for every operation of the input behavioural description and then assigning each operation to a specific control step. A control step is often equivalent to a single state of a finite-state machine. One commonly used approach is list scheduling, in which we specify a hardware constraint and use an algorithm to minimize the total execution time. The algorithm uses a local priority function to defer operations when resource conflicts occur. Another approach, called force-directed scheduling, allows us to specify a global time constraint, and the algorithm tries to minimize the resources required to meet that constraint.

A picture containing graphical user interface

Description automatically generated

1. High-level synthesis (HLS) design steps.

As we can see on the figure above, Force Directed Scheduling (FDS) is grouped into the Time Constrained Algorithms, but the algorithms basically used the combined concepts and ideas from As Soon As Possible (ASAP) and As Late As Possible (ALAP) which are listed under Unconstrained Algorithms. Therefore, as stated before, in order to understand the algorithms used in FDS, we need to first understand and have a firm grasp on the concepts of both ASAP and ALAP algorithms. A short brief explanation of List Scheduling will also be included for better understanding.

## Understanding As Soon As Possible (ASAP)

In the data flow graph (DFG) given above, each node is illustrated as operations. An operation can only be executed when it does not have parents. In other words, the highest nodes (topmost in a DFG diagram) are the operations available to be executed at the present time. If an operation is ready for running, the time step will be assigned to it as soon as possible. This means that every operation will be executed immediately without any delay. Note that the process of delaying it another time step is not possible in ASAP. We will be coming back to this delay deliberation when we discuss about ALAP. Then increase the time step and assign the currently highest nodes. That means the ASAP scheduling technique will assign all the available operations in each time step. There is no need to care about the data dependency explicitly. Since the highest nodes in DFG are the nodes available to execute, all their precedence operations have already executed. Therefore, it will follow the data dependency implicitly [3].

A picture containing diagram

Description automatically generated

1. High-level synthesis (HLS) design steps.

ASAP can be developed to work for limited resource constrained condition that always be true in real design. Not all the available nodes are assigned if there are some resource restrictions. Random selections are used because it does not have any priorities to figure out which operation has higher priority than others. Because of this defect, the result is not good enough comparing with other heuristic scheduling algorithms.

Here we use a simple example to illustrate the algorithm. The data-flow graph is shown in Fig. 3. All the operations used are addition operations in order to illustrate the basic concept of the technique. In real life situation, instead of just having addition operation, there could be a lot more operations combined together, and the resource constraints are defined in several subsets. We assume there are two adders provided here. In the first control step (CS1), there are three operations available for scheduling, namely 1, 2, and 3. Because it does not have selection category for ASAP scheduling, we pick up randomly. We select for example 1 and 2 for scheduling in CS1 and follow the algorithm to make the whole schedule. One possible solution made by ASAP is shown in Fig. 4.

A picture containing text, map, wire

Description automatically generated

1. ASAP Implementation.

## Understanding As Late As Possible (ALAP)

ALAP is interchangeable to ASAP scheduling algorithm except that it works from the bottom of the DFG towards the top. Each operation is scheduled at the last moment that must be executed. Because the assigning of control step counts downwards, the control step numbers are determined after the whole schedule is finished. ALAP is also developed for resource limitation condition as ASAP does and these two scheduling algorithms have the same drawback.

A picture containing text, map, wire

Description automatically generated

1. ALAP Implementation.

The problem with ASAP and ALAP is that they do not provide global exploration on the operation selection part. Some operations are more critical than others since they may be the predecessors of mass of operations. We should always find out the vital operations and schedule them first so as to achieve better scheduling solutions. The priority generation can be done by list scheduling, a heuristic scheduling algorithm, which will be explained afterwards.

## Understanding List Scheduling and Force Directed Scheduling (FDS)

In list scheduling, critical path is one basic definition for solving the operation selection problem. The longest length from the operation node to the node with no immediate successor is called critical path. And the length of this path is called critical path length. We can get the idea that which operation is more urgent than others according to the critical path length of the node. In other words, the critical path length is the indicator on determining which operation should have higher priority in selection under resource constraints.

Usually, the list scheduling can be based on several priority methods. The critical path length is one of them. Mobility is another mostly used priority function for list scheduling. Mobility is the number of control steps from the earliest to the latest feasible control step for an operation. The greater the mobility, the smaller the priority. This means, if there are some conflicts between several operations to be scheduled, the operations with larger mobility should be deferred to later control steps because they can be scheduled into more control steps.

Time constrained list scheduling can be derived from the original resource constrained list scheduling. Now we go back to see the data-flow graph described in Fig. 3, using list scheduling. The three available operations are 1, 2, and 3. The critical path length from that node to the end of the path is 1, 2, and 3 respectively. Because only two adders are available, the highest two operations on priority, operation 2 and operation 3, should be scheduled in CS1. All the available resources are used in CS1 so all other operations are deferred to CS2. In CS2, operation 1, 4, and 5 are available, the critical path length is found as 1, 1, and 2. So operation 5 is a compulsory one to be scheduled, and we arbitrarily pick up operation 1 as the other operation to be scheduled at the moment. And at last, the remaining two operations are available to be scheduled in CS3. The final solution is shown in Fig. 6 below.

Diagram

Description automatically generated

1. Solution using List Scheduling method.

In total, there are 3 control steps using list scheduling, while ASAP and ALAP might generate solutions using 4 control steps. Hence, we can find out that the list scheduling can consider global priority of operations so that it can solve the problem more efficiently [4].

Now we move on to force directed scheduling algorithm. It was originally proposed by Paulin and Knight in 1987 as heuristic approaches to solve both the resource-constrained and the latency-constrained scheduling problems because it uses a heuristic method to find out the solution. Paulin called force directed list scheduling the algorithm for the former problem because it is an extension of list scheduling and force directed scheduling the algorithm for the latter [3].

The force-directed scheduling (FDS) is a commonly used algorithm for scheduling. Force directed scheduling is usually treated as a time constrained scheduling technique. The main goal of the time constrained scheduling technique is to reduce the total hardware resources used but still fits the timing constraints. The latest implementation of the logarithm introduced here reduces the number of functional units, storage units, and buses required in the implementation of the design. This objective is achieved by attempting to uniformly distribute the operations onto the available resource units and by balancing the concurrency of operations assigned to them. The distribution ensures that resource units allocated to perform operations in one control step are used efficiently in all other control steps, which leads to a high utilization rate, which in turn decreases the total number of units required. The algorithm supports a comprehensive set of constraint types and scheduling modes. This balancing is done in three steps which are:

1) Determining the time frame of each operation

2) Creating a distribution graph

3) Calculating the force associated with each assignment

The forced-directed scheduling performs ASAP and ALAP scheduling first. After that, a distribution graph is achieved. Distribution graph is a plot of distribution over the schedule steps for each operation type. From the distribution graph, we can find out the possible resource utilization at every control step. We have the capability to make an even better utilization by pinching the maximum width of the graph. Force is one definition for this purpose. It is quite similar to the list scheduling using mobility as priority function.

Before describing the algorithms, we explain the underlying concepts. The time frame of an operation is the time interval where it can be scheduled. The earliest and latest start times in a frame can be computed by the ASAP and ALAP algorithms. Thus, the width of the time frame of an operation is equal to its mobility plus 1. The operation probability is a function that is zero outside the corresponding time frame and is equal to the reciprocal of the frame width inside it. Operations whose time frame is one unit wide are bound to start in one specific time step. For the remaining operations, the larger the width, the lower the probability that the operation is scheduled in any given step inside the corresponding time frame.

In force-directed scheduling, the selection of a candidate operation to be scheduled in each time step is done using the concept of force. Forces attract (repel) operations into (from) specific schedule steps. The concept of force has a direct mechanical analogy. The force exerted by an elastic spring is proportional to the displacement of its end points. The elastic constant is the proportionality factor. Paulin and Knight envisioned forces relating operations to control steps. The assignment of an operation to a control step corresponds to changing its probability. Indeed, such a probability is 1 in that step and 0 elsewhere once the assignment has been done. The variation in probability is analogous to the displacement of a spring. The value of the type distribution given by the distribution graph at that step is analogous to the elastic constant. The value of the mechanical analogy is limited as far as gaining intuition into the problem. An important point to remember is that forces are related to the concurrency of operations of a given type. The larger the force, the larger the concurrency. Thus, measuring the forces is useful as a heuristic to assign operations to time steps.

For the sake of simplicity, we assume in this section that operations have unit delays. The formalism can be extended to multi-cycle operations as well. The assignment of an operation to a step is chosen while considering all forces relating it to the schedule steps in its time frame. Forces can be categorized into two classes. The former is the set of forces relating an operation to the different possible control steps where it can be scheduled and called self-forces. The latter is related to the operation dependencies and called predecessor/successor forces. The less the force, the larger balance the solution have. In force-directed scheduling, each operator makes its own schedule for balancing the total cost of this kind of resource.

The total force on an operation related to a schedule step is computed by adding its self-force the predecessor/successor forces of all its predecessors and successors whose time frame is affected. Considering now the scheduling algorithms, the force-directed list scheduling algorithm addresses the minimum-latency scheduling problem under resource constraints. The rationale is to maximize the local concurrency (by selecting operations with large forces) while satisfying the resource bound. Note that force calculation requires the computation of the time frames, or their update, at each outer iteration of the algorithm, and when all candidate operations are critical (if their mobility is zero) and one (or more) must be deferred, thus causing a latency increase. The computational complexity of the algorithm is quadratic in the number of operations, because the predecessor/successor force computation requires considering operation pairs.

The force-directed scheduling Algorithm instead addresses the minimum resource scheduling problem under latency constraints. The algorithm considers the operations one at a time for scheduling, as opposed to the strategy of list scheduling which considers each schedule step at a time. At each iteration, the time frames, probabilities, and forces are computed. The operation with the least force is scheduled in the corresponding step. The rationale is to minimize the local concurrency related to the resource usage while meeting the latency bound that is satisfied by construction because operations are always scheduled in their time frames [2].

## Examples and Equations

In this section, we will implement all the knowledge that we gained from the concept learning. Some calculations and equations will be introduced and explained to enhance our understanding on this topic. We will be using the same three steps in balancing as stated earlier in the paper to solve the problem in the manner of FDS. Here we make a simple example to illustrate the force-directed scheduling We take a differential equation as below as an example for our problem solving:

y" + 3\*z\*y' + 3\*y = 0

This can be calculated by using the iterative algorithm,

while (z < a) repeat

zl: = z + dz;

ul: = u - (3\*z\*u\*dz) - (3\*y\*dz);

yl: = y + (u\* dz);

z: = zl ; u: = ul ; y: = yl ;

First, we determine the time frame of each operation by evaluating the ASAP and ALAP schedules. By combining results for both schedules, we can ascertain the time frame of each operation.

Diagram, engineering drawing

Description automatically generated

1. Comparison between using ASAP and ALAP approach.

Figure above illustrates the control- and data-flow graphs for the ASAP and ALAP schedules for the inner loop of the differential equation example. The nodes represent functional operations, while edges represent the data dependencies between these operations. The resulting time frames are given in the figure below.

Diagram

Description automatically generated

1. Assigning time frames from ALAP method.

The width of the box containing an operation represents the probability that the operation will be eventually placed in some time slot. We assume that the probability distribution for each operation is uniform. We chain operations by extending their time frames into the previous (or next) control step. However, before we can extend them, their combined propagation delays are added to the latch and estimated interconnection delays must be less than the clock cycle. We can extend this single-cycle method in a straightforward way to support multicycle operation.

The second step is to create the distribution graph. The reason behind this is to add the probabilities of each type of operation for each control step, or c-step, of the control-flow or data-flow graph. The resulting distribution graphs indicate the concurrency of similar operations. For each graph, the distribution in c-step is calculated by using the formula:

##### A picture containing logo Description automatically generated

Force calculation. The final step is to calculate the force associated with every feasible c-step assignment of each operation. We temporarily reduce the operation's time frame to the selected c-step. For an operation with an initial time frame that extends from c-steps t to b, the force associated with its assignment to c-step j is calculated with:

Schematic

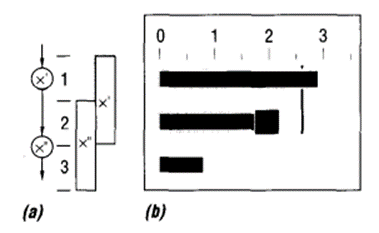
Description automatically generated

In other words, the force associated with the tentative assignment of an operation to c-step j is equal to the difference between the distribution value in that c-step and the average of the distribution values for the c-steps bounded by the operation’s initial time frame. Figure 9 illustrates this relationship, which we then use to calculate the force associated with the assignment of multiplication x’ to c-step 1. Here, Equation 2 yields

Text

Description automatically generated with medium confidence

As shown in the shaded columns in Figure 9(b), if we assign multiplication x’ to c-step 1, the distribution is not very well balanced, and multiplier costs will be higher.



1. Assigning time frames to Distribution Graph (First Approach).

Not to forget, we must also calculate the force for all predecessors and successors of the current operation whenever their time frames are affected. These additional forces are called indirect forces. The total force is the sum of the direct and indirect forces. In the force calculation in Figure 4, we did not have any indirect force because the time frame of the successor multiplication operation x’’ was not affected. However, if we assign x’ to c-step 2, we are implicitly forcing x’’ into the third control step, as the shaded bars in Figure 5a illustrate. Thus, we are exerting additional force, and the total force becomes:



A picture containing text, clock

Description automatically generated

1. Assigning time frames to Distribution Graph (Second Approach).

As shown by the shaded columns in Figure 10(b), this assignment causes a better balancing of the DG. We can see from the calculation that the calculated force is a negative value. After we have calculated the force of all operations, we assign an operation to a c-step in a way that yields the lowest force, that is, balances the concurrency of the operations most effectively. We readjust the time frames accordingly and repeat the entire process until all operations are scheduled. We consider Input/Output operations in the same way we would any regular operation. By balancing the concurrency of Input/Output operations, we minimize the number of required ports. This benefit is particularly significant for designs that limit the number of pins.

## Minimizing Storage And Interconnection Cost

Most scheduling algorithms minimize the cost of functional units but ignore the associated storage and data-transfer costs, even though scheduling has a direct effect on them. For example, the fewest buses required for a scheduled control-flow or data-flow graph is the number of concurrent data transfers in a control step. The fewest registers required is the maximum number of data arcs that cross the boundary of a control step. Figure below illustrates two simple schedules with different hardware costs.

Diagram, schematic

Description automatically generated

1. Comparison between two simple schedules with different hardware costs.

The schedule in Figure 11(a) appears to be the best of the two since it requires only one multiplier. However, the schedule in Figure 11(b) may have a lower global cost because the allocation cost for ports and buses and the storage costs are considerably lower. Since the Registers and Interconnect area consumption is reduced in solution 11(b), it might be optimal.

##### conclusion

The force -directed scheduling algorithm has been implemented in different design systems. The results have been shown to be superior to compare with list scheduling. However, its run times tend to be long for large graphs, limiting its practical use for large designs.

##### References

1. Coussy, P., Gajski, D., Meredith, M. and Takach, A., 2009. An Introduction to High-Level Synthesis. *IEEE Design &amp; Test of Computers*, 26(4), pp.8-17.
2. De Micheli, G., 1994. Synthesis and optimization of digital circuits. New York, N.Y.: McGraw-Hill.
3. Chen, K., 2022. Implementation of a Scheduling and Allocation Algorithm for Hardware Evaluation. [online] DIVA. Available at: <http://www.diva-portal.org/smash/record.jsf?pid=diva2:20244> [Accessed 14 May 2022].R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
4. Potkonjak, M., 1992. Algorithms for high level synthesis. Berkeley: Electronics Research Laboratory, College of Engineering, University of California.