A Statistical-based Scheduling Algorithm in Automated Data Path Synthesis

Byung Wook Jeon and Chidchanok Lursinsap
The Center for Advanced Computer Studies of
The University of Southwestern Louisiana
Lafayette, LA 70504
Email: bwj@cacs.usl.edu and lur@cacs.usl.ed

Abstract - **In this paper, we propose a new heuristic scheduling algorithm based on the *statistical analysis* of the *cumulative frequency distribution* of operations among control steps. It has a tendency of escaping from local minima and therefore reaching a globally optimal solution. The presented algorithm considers the real world constraints such as chained operations, multicycle operations, and pipelined data paths. The result of the experiment shows that it gives optimal solutions, even though it is greedy in nature.**

1 Introduction

The high level synthesis task is to transform an abstract behavioral specification of a digital system into a register transfer level (RTL) structure that realizes the given behavior. This task can be decomposed into a number of distinct but not independent subtasks [?]. The first one is to specify the behavior of a digital system using a programming language or a hardware description language (HDL), and then to translate the description into a graph-based representation which is said to be control and data flow graph (CDFG). This subtask is followed by operation scheduling process that assigns each operation to a control step. The third phase is the resource allocation process that assigns the operations and values to hardware. Among these subtasks, operation scheduling and resource allocation stages are the main processes which are closely interrelated. The operation scheduling usually determines the major design decisions such as the number and types of functional units, clock cycle time, lifetimes of variables, and implementation styles (pipeline, chained and multicycle operations, etc.). Therefore, a good scheduler is critical to an automated data path synthesis system [?, ?, ?, ?].

The simplest scheduling scheme is to schedule operations "as soon as possible" (ASAP) or "as late as possible" (ALAP) which is done in Emerald system [?]. This scheme may produce an unrealizable scheduling if there is resource limitation. To manage this problem, ASAP scheduling with postponement of operations is proposed in MIMOLA msystem [?] and Flamel system [?]. Another approach to scheduling is the list scheduling technique in which operations are sorted in topological order using the precedences specified in CDFG, and these operations are then scheduled into control steps using some heuristic priority function such as operation mobility used in SLICER [?]. Freedom-based scheduling used in MAHA [?] determines the critical path. The operations on the critical path are scheduled first and assigned to functional units. Then the other operations are scheduled and assigned one at a time. In force-directed scheduling [?], the operations are iteratively scheduled into control steps based on the evaluation using distribution graph, which illustrates the distribution of fixed operations and unscheduled operations in each control steps.

All these schemes which are the constructive algorithms build up a schedule by adding operations one at a time until all operations have been scheduled. None of these constructive algorithms is guaranteed to find the best possible schedule. In ALPS [1], the scheduling problem is formulated as an integer linear programming (ILP) which gives a globally optimal solution. Since ILP is an exponential algorithm in nature, it is not acceptable for some large example even if an optimal solution can be generated. Practically, it is not always desirable to produce an optimal solution in the scheduling problem space because the problem itself is NP-complete. ¿From this observation, Park and Kyung [8] proposed an efficient algorithm based on the multiple exchange pair selection algorithm with cumulative gain which was proposed by Kernighan and Lin in their min-cut graph partitioning problem [2]. They devised the selection function used in their algorithm [2] by taking into account the fact that the density of functional units was the important factor for determining an operation to be moved, but no theoretical justification for it was provided. They also intentionally gave the preference to a certain operation in a particular control step.

The number of times that an operation occurs in each control step shows how the operations in the control and data flow graph *(CDFG)* are distributed among control steps. It also determines the *frequency distribution* of the operations and thus characterizes the feature of CDFG. From the basis of this statistical property, some measures can be derived which may be used for selecting a good candidate operation to be moved. The theoretical justification for these is quite simple but strong enough to devise a new selection function. Based on this observation, we propose a new scheduling algorithm using *statistical analysis* of the cumulative *frequency distribution* of operations among control steps. The presented algorithm considers the real world constraints such as chained and multicycle operations, and pipelined data paths. The essential method being used in our algorithm is also based on the multiple exchange pair selection algorithm [2].

We will start by describing the scheduling process in the next section. The scheduling process consists of two phase: one is the prephase for the scheduling phase and the other is the scheduling phase. The prephase transforms a given CDFG into the intermediate form containing information necessary for the scheduling phase. The scheduling phase selects a set of trials, and accepts even locally undesirable movements if they belong to a set of movements which maximizes the object function as a whole. This will be followed by the extention of algorithm to the real world constraints such as chained operations, multicycle operations, and pipelined data paths. Then, the experimental results for the examples used in several other systems will be given. Finally, concluding remarks will be made in the last section.

2 Scheduling Process

The presented scheduling algorithm takes into account the *frequency distribution* of operations occurred in each control step. The essential method being used is a modified version of the algorithm in [2]. This algorithm selects a set of trials, and accepts even locally undesirable movements if they belong to a set of movements which maximize the object function as a whole. Even though this algorithm may prone to get stuck in a local minimum rather than finding the global optimum, it has a *hill climbing* property that can escape from local minima and therefore reach a globally optimal solution. In the practical point of view, it is not always necessary to produce an optimal solution as long as a near optimal one can be obtained.

The scheduling process consists of two phases; one is a prephase for the scheduling phase and the other a scheduling phase. We will now describe the prephase that transforms a given *CDFG* into the intermediate form containing information necessary for the scheduling phase. Then, the selection function which is the kernel of our scheduling phase is derived. This will be followed by description of the scheduling algorithm which improves the solution iteratively. To illustrate our scheme, we will use the example given in [11]. Figure 1 illustrates the *CDFG* of this example.

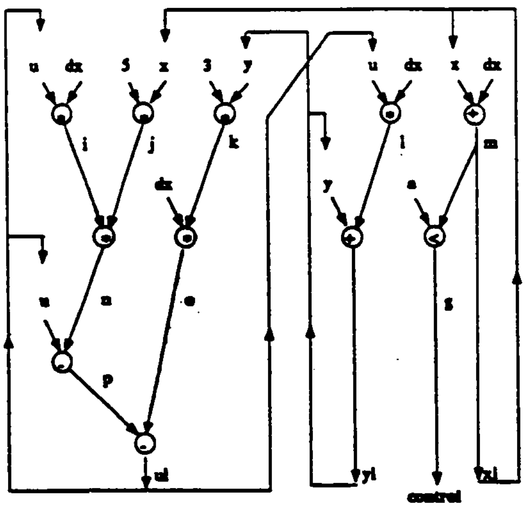

Figure 1: The data flow graph for the example

2.1 Prephase for Scheduling

This phase transforms a given *CDFG* into a constraint graph *(CG)* to identify the precedence relationships that are due to both data flow and control flow dependencies, and the timing behavior of each operation. Each node corresponds to an operation and has weight which specifies the minimal number of cycles required to execute the corresponding operation. Two nodes in *CG* are connected by a directed edge if and only if there is a precedence relation between two operations. The weight of an edge indicates the number of control steps between two operations. This transformed graph will be the input to the scheduling phase. Similar descriptions of *CG* used in our system can be found in the literature [3, 8].

2.2 Selection Function

The number of times that an i-type operation occurs in each control step illustrates how the operations are distributed among control steps in a given *CDFG* and therefore characterizes the feature of that *CDFG* . This implies that if one operation in control step j is moved to another step k, the *CDFG* may have different properties after this movement. That is, the distribution of operations in a given *CDFG* can be changed after the movement, which implies that the concurrency of operations may be affected. We may also compute measures such as the average number of operations for each control step and the spread of operations among control steps. For the purpose of our scheduling process, we primarily consider the two most important of such quantities, which are the *mean* and the *variance* of the number of operations, to describe the balanced distribution of operations. Based on these measures, we will explore how to balance the concurrency of the operations among control steps in a given *CDFG* .

Roughly speaking, the average number is a measure of the minimum number of operations needed to uniformly distribute them among control steps and the variance measures the spread or dispersion of operations in the corresponding control step. We first want to introduce a measure for the average number of operations. The most common such measure is the arithmetic mean. The *mean* of the number of operations of i-type operation is denoted by M_i and defined by the formula

$$M_i = \lceil \frac{N_i}{N} \rceil$$

where N_i denotes the total number of i-type operations and N the minimum number of control steps required to perform a given *CDFG.*

Now we want to introduce a measure for the spread or variation of the number of operations to distinguish between two *CDFGs.* We choose a quantity that measures the deviation from the mean M_i in each control step and then take square of such quantity to derive another measurement. This quantity of i-type operation in control step j can be defined by

$$V_i(j) = (n_i(j) - M_i)^2$$

where $n_i(j)$ is the number of i-type operations in control step j. By using this quantity, another measurement which is said to be the *variance* of the distribution of operations between two control steps can be introduced by

$$\begin{aligned} V_i(j,k) &= V_i(j) + V_i(k) \\ &= (n_i(j) - M_i)^2 + (n_i(k) - M_i)^2 \end{aligned}$$

where $V_i(j,k)$ denotes the *variance* of i-type operation at control steps j and k.

It can be seen that as the value of $V_i(j,k)$ approaches to zero, the number of operations at both control steps i and j tends to be balanced uniformly. That is, it measures the *degree* of the *balance* of operations between two control steps. Then, the *variance* of the distribution after the movement of an operation O_{i_p} from step j to k can be also defined by

$$\begin{aligned} V_i'(j,k) &= V_i'(j) + V_i'(k) \\ &= (n_i(j) - 1 - M_i)^2 + (n_i(k) + 1 - M_i)^2 \end{aligned}$$

where $V_i'(j,k)$ denotes the *variance* of i-type operation at control steps j and k after the movement.

From the above results, we can derive another measurement, the *Change of Variance (CV),* which is the difference between variances before and after the movement. The *CV* of an operation is defined by the formula

$$C_{i_p}(j,k) = V_i(j,k) - V_i'(j,k)$$

where $C_{i_p}(j,k)$ denotes the value of the *CV* when p-operation of i-type operations is moved from control step j to k.

We can observe that if the value of this function is greater than zero, it is preferable to move O_{i_p} from step j to k. The negative value indicates that the movement is not desirable. Clearly, from the values of this function, we can make a decision whether or not the movement of an operation is preferable. That is, the balance of the concurrency of operations can be achieved by decreasing the number of operations in the control step where the value of *CV* is maximal. This provides the *selection function* used in our scheduling algorithm to choose a good candidate operation to be moved.

2.3 Scheduling Algorithm

The goal of our scheduling algorithm is to reduce the number of functional units required but not to lengthen the total execution time. This can be achieved by balancing the concurrency of the operations assigned to the functional units, which implies the high utilization of the units and in turn minimizes the number of units required. The balance of concurrency can be specified by the degree of the distribution of operations appeared in each control step. Therefore, we can accomplish this by distributing those as uniformly as possible without violating any constraints.

We now describe our scheduling algorithm on the basis of this observation. For the simplicity, we temporarily assume that each operation executes in one control step and all operations are either arithmetic or logic operations only. The objective function is the cost function whose arguments are each type of functional units and the number of those in each control step. The *ASAP* and *ALAP* schedules are shown in Figure 2 and Figure 3, respectively. Figure 4 depicts the final scheduled graph for the example given in Figure 1.

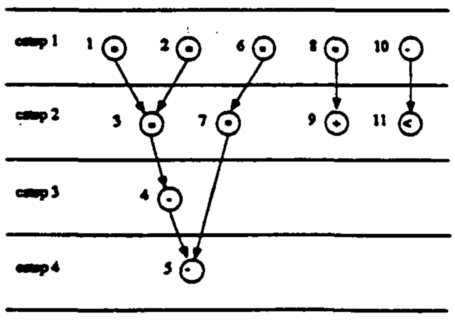

Figure 2: The ASAP schedule for the example

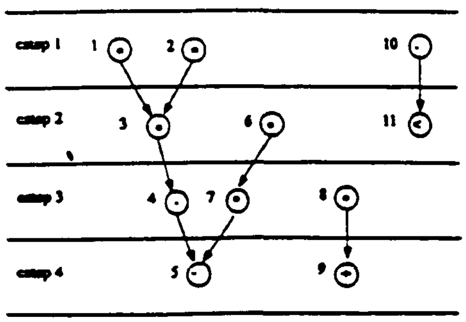

Figure 3: The ALAP schedule for the example

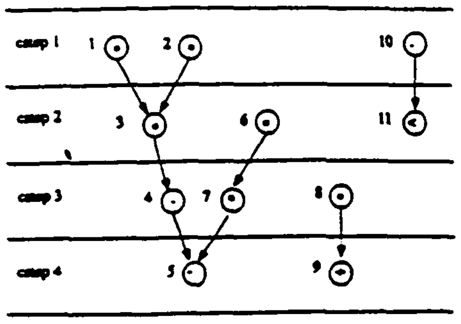

Figure 4: The scheduled data flow graph for the example

STEP 1 Determine the critical path and compute the mobility of each operation using both *ASAP* and *ALAP* schedules. In the following steps, the operations in the critical path are not considered because each of them can not be moved to the other control step. We consider either an *ASAP* or an *ALAP* schedule as an initial feasible solution.

STEP 2 Set counter to zero. Find the movable operations from the feasible solution determined in the previous iteration. If there is no movable operation, then go to *STEP5.* Otherwise, the following steps are performed.

STEP 3 Select the control step k for an operation O_{i_p} such that the selection function $C_{i_p}(j,k)$ has the maximum value and then move O_{i_p} to control step k. O_{i_p} denotes an i-type operation which is numbered as p in *CDFG.* Then, that operation is locked at the control step k temporarily.

STEP 4 Compute the gain which is the change of the value of the objective cost function when the operation O_{i_p} is moved from control step j to k. Each of these gains computed in this step is stored somewhere for the later use. Increment counter and go to *STEP2.*

STEP 5 Find a sequence of operations such that its cumulative gain is maximum among those sequences generated up to now. If there is no improvement for the objective cost function, the scheduling process is terminated. Otherwise, the current feasible solution determined during the previous iteration is modified and go to *STEP2.*

We will now illustrate how the algorithm works using the example shown in Figure 1. For the simplicity, it will be temporarily assumed that the available functional units are multipliers and ALUs, and that multiplier cost is four and ALU cost is one. We also assume that ALU is capable of performing addition, subtraction, and comparison. Let us consider the ASAP schedule depicted in Figure 2 which is chosen to be an initial feasible solution. It shows that the movable operations are O_{mul_7}, O_{alu_9}, and $O_{alu_{11}}$. We then compute $C_{i_p}(j,k)$ for each of these operations as described before. For example, if the operation O_{mul_7} is moved from control step 2 to 3 in Figure 2, then we can compute the selection function as follows:

$$\begin{aligned} M_{mul} &= \lceil \tfrac{6}{4} \rceil \\ &= 2 \\ C_{mul_7}(2,3) &= \{(2-2)^2 + (0-2)^2\} - \{(1-2)^2 + (1-2)^2\} \\ &= 2 \end{aligned}$$

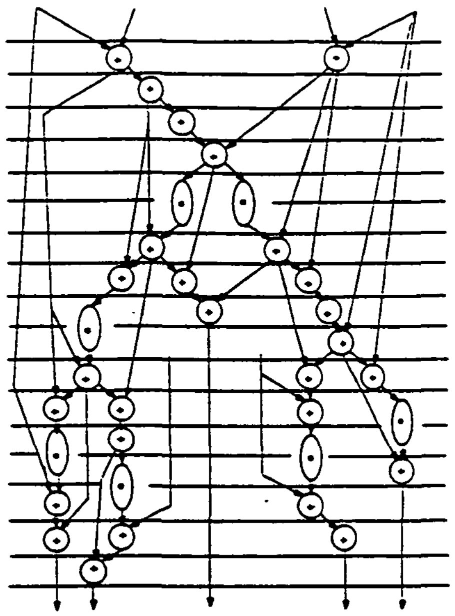

Figure 5: Fifth-order wave digital eliptical filter example

5 Conclusion

We have presented a new heuristic scheduling algorithm based on multiple exchange pair selection algorithm using the statistical analysis on the cumulative frequency distribution of the number of operation among control steps. The presented algorithm considers the real world constraints such as chained and multicycle operations, and pipelined data paths. The theoretical justification of this statistical method used in our selection function is simple but is strong enough to choose a good candidate operation to be moved. The proposed algorithm has a *hill climbing* property that can escape from local minima and reach a globally optimal solution, even though it is greedy in nature and therefore may prone to get stuck in a local minimum rather than finding the global optimum.

The experimental result shows that this algorithm can generate the optimal solutions for the examples used in the literatures [4, 10, 11].

References

[1] Cheng-Tsung Hwang, Jiah-Hurng Lee, and Yu-Chin Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Tran. Computer-Aided Design*, 10(4):464–475, April 1991.

[2] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Syst.Tech. J.*, 49(2):291–308, 1970.

[3] David C. Ku and Giovanni De Micheli. Constrained Resouce Sharing and Conflict Resolution in Hebe. In *Integration*, volume 12, pages 131–165. Elsevier, December 1991.

[4] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI Modern Signal Processing.* Prentice Hall, 1985.

[5] P. Marwedel. A New Synthesis Algorithm for MIMOLA Software System. In *Proceedings of the 23th IEEE/ACM Design Automaton Conference*, pages 271–277, July 1986.

[6] M. C. McFarland, S. J. Parker, and Raul Camposano. The High-Lvel Synthesis of Digital System. *Proceeding of IEEE,* 78(2):301–318, February 1990.

[7] B. M. Pangrle and D. D. Gajski. Slicer: A state synthesis for intelligent silicon compilation. In *Proceedings of the IEEE International Conference on Computer Design*, pages 42–5, October 1987.

[8] In-Cheol Park and Chong-Min Kyung. Fast and Near Optimal Scheduling in Automatic Data Path Synthesis. In *Proceedings of the 28th IEEE/ACM Design Automaton Conference*, pages 680–685, July 1991.

[9] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-7(3), March 1988.

[10] A. C. Parker, G. T. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23th IEEE/ACM Design Automaton Conference*, pages 461–66, July 1986.

[11] P. G. Paulin and J. P. Knight. Force-directed Scheduling in Automatic Data Path Synthesis. In *Proceedings of the 24th IEEE/ACM Design Automaton Conference*, pages 195–202, July 1987.

[12] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-8(6):661–79, June 1989.

[13] H. Trickey. Flamel: A high-level hardware compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-6(2):259–69, March 1987.

[14] C. J. Tseng and D. P. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-5(3):379–95, July 1986.