Lower Bounds on the Iteration Time and the Number of Resources For Functional Pipelined Data Flow Graphs

Yuan Hu, *Ahmed Ghouse* and *Bradley S. Carlson*

The Department of Electrical Engineering
State University of New York at Stony Brook
Stony Brook, NY 11794-2350
Phone: (516) 632-8474
Fax: (516) 632-8494

Technical Report 666
College of Engineering and Applied Science

Abstract— An algorithm is presented to determine two lower bounds in functional pipelined data path synthesis. Given an iteration time constraint T and a task initiation latency, the algorithm computes a lower bound on the number of functional units required to execute the data flow graph of a loop body, and given resource constraint R and a task initiation latency the algorithm computes a lower bound on the number of time steps required to execute the data flow graph. The lower bounds not only greatly reduce the size of the solution space, but also provide a means to measure the proximity of the final solution to an optimal one. Since the bounds are computed in polynomial time, this lower bound algorithm is very effective, especially for large DFGs. Experiments indicate that the lower bound is very tight. For all of the test cases the difference between our solution and the optimal solution is not greater than one.

I Introduction

In the behavioral synthesis of integrated circuits (ICs) the fundamental steps are the scheduling and allocation of operations on functional (hardware) units. The synthesis is commonly divided into a data path design and a control path design. Operation scheduling and allocation are the most important steps in behavioral synthesis [1], since they can effectively determine the time-cost trade-offs. The scheduling problem can be approached from two perspectives: *resource constrained* and *performance constrained.* Resource constrained scheduling is the task of minimizing the completion time given a fixed amount of resources, and performance constrained scheduling is the task of minimizing the amount of resources given a fixed completion time. A comprehensive survey of various scheduling and allocation techniques can be found in [2].

A behavioral description often consists of loop statements, and the loop execution usually dominates the total execution time. Optimizing the execution of the loop body is critical to the performance of the design. Scheduling a loop body is quite different from scheduling a straight-line code segment, since parallelism beyond the iteration boundaries can be exploited by executing several iterations of the loop concurrently. If the loop body is treated as a data flow graph, then the execution of the loop can be viewed as pipelining the data flow graph. The type of pipelining where the entire data flow graph is pipelined is called functional pipelining.

The research on functional pipelined data path scheduling is addressed in [3, 4, 5, 6, 7, 8, 9]. A common characteristic of these polynomial time heuristics is that they give only an *upper bound* to the minimum number of functional units or time steps needed to meet the design constraints. Although they obtain near optimal or optimal schedules on a few test examples, these algorithms in the general case have no way of knowing the proximity of their solution to the optimal solution. The lack of a tight lower bound is recognized as one of the weaknesses in current scheduling algorithms [10]. Generally, the quality of a heuristic solution for an NP-complete problem is proportional to the time spent in search for a good solution; however, a quantitative measure of the quality of a heuristic solution can only be obtained by measuring its proximity to the optimal solution. Therefore, tight lower bounds are essential to providing a good measure of the quality of an algorithm when the optimal solution is too expensive to compute. In addition, tight lower bounds can be used as a starting point for many scheduling algorithms (e.g., branch-and-bound) because tight lower bounds can greatly reduce the computation time of these algorithms.

Recent work has been done for lower bound performance estimation under resource constraints on *non-pipelined* data flow graphs [11]. In their work, a greedy algorithm is used to obtain a lower bound. The technique will not work in the case of functional pipelined data flow graphs, and it does not address the performance constrained scheduling problem.

The purpose of our paper is to present a new polynomial time algorithm to compute a lower bound on the number of functional units required to finish all the operations of a data flow graph under time constraint T, and to compute a lower bound on the number of time steps required to finish the data flow graph under resource constraint R. The lower bounds presented here can be used to develop new scheduling algorithms and to quantify the quality of existing scheduling algorithms.

The organization of our paper is as follows. Presented in Section II is the lower bound on the number of functional units for pipelined data flow graphs given the initiation latency and total iteration time. Presented in Section III is the lower bound on the number of time steps for pipelined data flow graphs given the resources and the initiation latency. Presented in Section IV are the complexity analysis and the benchmark experimental results, and the paper is concluded in Section V.

II LOWER BOUND ON THE NUMBER OF RESOURCES

The objective of this section is to find a lower bound on the number of functional units needed to execute the operations of a functional pipelined data flow graph given the task Initiation Latency (IL) and the iteration time T. As a prerequisite for computing the lower bound the as soon as possible (ASAP) and the as late as possible (ALAP) schedules[1] must be computed without pipelining under the assumption that there is an unlimited amount of resources; furthermore, the critical path must be identified. The algorithms for processing the data flow graph to find the ASAP and ALAP schedules can be found in [12]. The run time complexities of these algorithms are $O(E)$, where E is the number of edges in the graph.

The ASAP and ALAP schedules are used to compute the lower bound on the number of functional units of each type required to execute the operations of the DFG within a specified time. The final solution is a specification of the number of functional units of each operation type. For example, Fig. 1(a) is a data flow graph with two types of operations: addition and multiplication. In our method the lower bound for the number of multipliers and the lower bound for the number of adders are determined separately and combined to

[1] A schedule is represented by the completion time steps of its operations.

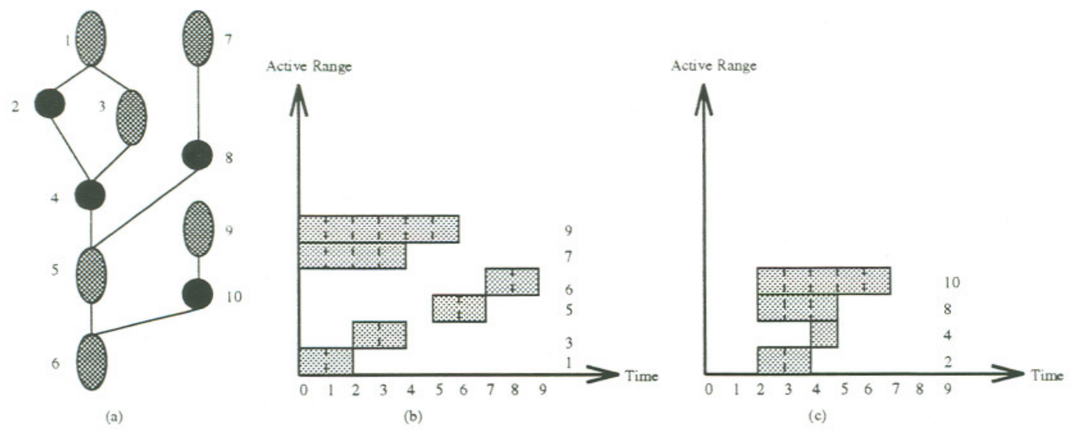

Fig. 1. (a) An example loop body data flow graph, (b) the list of active ranges for the multiplications, and (c) the list of active ranges for the additions.

Table I. The ASAP and ALAP schedules of operations in Fig.1(a).

Operations	1	2	3	4	5	6	7	8	9	10
ASAP	2	3	4	5	7	9	2	3	2	3
ALAP	2	4	4	5	7	9	4	5	6	7

form the final solution. The ASAP and ALAP schedules of each operation in Fig. 1(a) are shown in Table I. In Fig. 1(a) the operation identifier is shown adjacent to the node, an oval represents a multiplication which requires two units of execution time and a circle represents an addition which requires one unit of execution time.

Before the lower bound for functional pipelining is studied, we first discuss some of the properties that are associated with a non-pipelined loop body data flow graph. Let m be the total number of operations in the loop body data flow graph, τ_j be the time for *completing* the j^{th} operation in the graph ($j = 1, 2, ..., m$), and integer s the time required to execute an operation. An m-tuple $(\tau_1, \tau_2, ..., \tau_m)$ of completion times for the m operations in a data flow graph represents a schedule of the graph. Let $\overline{\tau_j}$ be the *completion* time of operation j in the ASAP schedule of the graph, and let $\underline{\tau_j}(T)$ be the *completion* time of operation j in the ALAP schedule. Note that the ALAP completion time $\underline{\tau_j}(T)$[2] depends on the total iteration time T.

The time interval $[\overline{\tau_j} - s, \underline{\tau_j}]$ is called the *active range* for operation j. The active range

[2] $\underline{\tau_j}(T)$ is abbreviated as $\underline{\tau_j}$ in the sequel.

for each operation can be listed in a Cartesian coordinate system as shown in Figs. 1(b) and (c). For example, in the data flow graph in Fig. 1(a) assume that the specified total loop body iteration time is $T = 9$, and that two units of time ($s = 2$) are needed to complete a multiplication operation, then (for multiplication 7) we have $\overline{\tau_7} = 2$ and $\underline{\tau_7(9)} = 4$. Thus, the active range of operation 7 is $[\overline{\tau_7} - s, \underline{\tau_7}] = [0, 4]$ as indicated in Fig. 1(b).

Given a schedule $(\tau_1, \tau_2, ..., \tau_m)$ the activity of operation j in the graph can be described by the following function.

$$f(\tau_j, t) = \begin{cases} 1 & t \in [\tau_j - s, \tau_j] \\ 0 & \text{otherwise} \end{cases}$$

$f(\tau_j, t)$ is called the *active function.* $f(\tau_j, t)$ is the actual representation of an operation in a schedule $(\tau_1, \tau_2, ..., \tau_m)$, while the active range $[\overline{\tau_j} - s, \underline{\tau_j}]$ represents the freedom of an operation among all possible schedules. Furthermore, the active function of the entire loop body data flow graph without functional pipelining can be described as the sum of the active functions of the individual operations.

$$F(\tau_1, \tau_2, ..., \tau_m, t) = \sum_{j=1}^{m} f(\tau_j, t) \; [14]$$

F is the active function of the entire data flow graph with respect to the schedule $(\tau_1, \tau_2, ..., \tau_m)$ and is an indication of how many operations are being executed *simultaneously* at an instant of time t without considering functional pipelining.

In functional pipelining different initiations of tasks may share the same control steps. If we look at the loop body data flow graph, these operations are separated by IL control steps. In other words, operations in a loop body data flow graph that are IL control steps apart will be executed in the same control step when functional pipelining is utilized. To illustrate the idea of functional pipelining we use the example in Fig. 1(a). The data flow graph is viewed as the loop body data flow graph in a functional pipelining situation. In the example successive iterations begin their executions two control steps apart, i.e., $IL = 2$; therefore, after every two units of time a new task is introduced, and operations that are two control steps apart will be executed in the same control step in the final schedule.

A partition is formed on the operations of the loop body data flow graph. Operations that are IL control steps apart belong to the same partition, and there are IL different partitions. The active function of partition i is represented by $\sum_{i=0}^{\lceil \frac{T}{IL} \rceil} F(\tau_1, \tau_2, ..., \tau_m, t + i \times IL)$, where $0 \leq t < IL$. The active function reflects the consideration of all the operations in a partition. Formally, given the task Initiation Latency IL and the iteration time T, we are to find the minimum number of functional units (n_{fp}) and an optimal schedule ($\tau_1^*, \tau_2^*, ..., \tau_m^*$) such that

$$\begin{aligned} n_{fp} &= \max_{\forall t \in [0,\mathrm{IL}]} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} F(\tau_1^*, \tau_2^*, ..., \tau_m^*, t + i \times IL) \\ &= \min_{\forall(\tau_1, \tau_2, ..., \tau_m) \in D} \max_{\forall t \in [0,\mathrm{IL}]} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} F(\tau_1, \tau_2, ..., \tau_m, t + i \times IL), \end{aligned} \tag{1}$$

where D is the schedule space.

To find a feasible schedule under these conditions is an NP-complete problem, but there is potential to find a tight lower bound by relaxing the constraints. There are four major constraints: the precedence relationships between operations, the active ranges, the task initiation latency IL and the iteration time T. We construct a relaxation by eliminating constraints for the precedence relationships, but keep the active ranges of operations and other constraints. As a result of this relaxation each operation can be considered *independently.* We define the *minimum load* for an operation j in the time interval $[t_1, t_2] \subseteq [0, IL]$, denoted $\Phi_j(t_1, t_2)$, as the minimum overlap between the active function $f(\tau_j, t)$ of operation j and the time intervals $\cup_{i=0}^{\lceil \frac{T}{IL} \rceil}[t_1 + i \times IL, t_2 + i \times IL]$. The minimum load is given by

$$\Phi_j(t_1, t_2) = \min_{\forall \tau_j} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1+i\times IL}^{t_2+i\times IL} f(\tau_j, t)dt. \tag{2}$$

The integral itself indicates how much of the active function $f(\tau_j, t)$ of operation j will be present in the time interval $\cup_{i=0}^{\lceil \frac{T}{IL} \rceil}[t_1 + i \times IL, t_2 + i \times IL]$ for a given completion time τ_j. Therefore, $\Phi_j(t_1, t_2)$ is the minimum portion of the active function which *must* be present in the time interval $\cup_{i=0}^{\lceil \frac{T}{IL} \rceil}[t_1 + i \times IL, t_2 + i \times IL]$ among *all* schedules. The collective effect of the $\Phi_j(t_1, t_2)$'s of all the operations is an indication of the minimum number of functional units needed to meet the time constraint.

In this formulation an entire partition is taken into consideration; therefore, instead of calculating the minimum load in a single time interval, we must calculate the minimum load for all the time intervals IL units apart. The computation procedure of Φ_j is illustrated in Fig. 2 where the overlap between two time intervals A and B is denoted by $|A \cap B|$. Note that in the outermost *for* loop in Fig. 2 τ_j is from $\underline{\tau_j}$ to $\underline{\tau_j} + IL$ only, because beyond $\underline{\tau_j} + IL$ the time step will belong to one of the partitions already considered. Consequently, the lower bound on the number of functional units required to complete the functional pipelined data flow graph given the initiation latency IL and the iteration time T is stated in the following theorem.

```
algorithm $\Phi_j(t_1, t_2, \underline{\tau_j}, \overline{\tau_j})$
    $\Phi_j$ = maximum integer;
    for each position of $\tau_j$ between $\underline{\tau_j}$ and $\underline{\tau_j} + IL$ do
        SUM = 0;
        for i = 0 to $\lceil \frac{T}{IL} \rceil$ do
            SUM += $|[t_1 + i \times IL, t_2 + i \times IL] \cap [\tau_j - s, \tau_j]|$;
        end_for
        $\Phi_j = min(\Phi_j, SUM)$;
    end_for
    return($\Phi_j$);
end_algorithm
```

Fig. 2. Algorithm to compute Φ_j.

Theorem 1 *Given a functional pipelined data flow graph, the task initiation latency IL and the loop body data flow graph iteration time T, the minimum number of functional units required to complete the functional pipelined data flow graph is n_{fp}. A lower bound on n_{fp} is*

$$n_{fpL} = \lceil \max_{\forall [t_1, t_2] \subseteq [0, IL]} \frac{\sum_{j=1}^{m} \Phi_j(t_1, t_2)}{t_2 - t_1} \rceil .$$

Proof: If τ_j^* represents the completion time for operation j in an optimal schedule, then from the definition of Φ_j

$$\Phi_j(t_1, t_2) = \min_{\forall \tau_j} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1 + i \times IL}^{t_2 + i \times IL} f(\tau_j, t) dt \le \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1 + i \times IL}^{t_2 + i \times IL} f(\tau_j^*, t) dt.$$

Note that τ_j^* is part of the optimal schedule for the entire data flow graph, and it is not necessarily the minimum τ_j with respect to $\Phi_j(t_1, t_2)$.

Taking the summation on both sides

$$\begin{aligned} \sum_{j=1}^{m} \Phi_j(t_1, t_2) &\le \sum_{j=1}^{m} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1 + i \times IL}^{t_2 + i \times IL} f(\tau_j^*, t) dt \\ = \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1 + i \times IL}^{t_2 + i \times IL} \sum_{j=1}^{m} f(\tau_j^*, t) dt &= \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1 + i \times IL}^{t_2 + i \times IL} F(\tau_1^*, \tau_2^*, ..., \tau_m^*, t) dt. \end{aligned}$$

Using a variable transformation with $t' = t - IL \times i$, we have $dt' = dt$ and $t = t' + IL \times i$. Therefore, from (1),

$$\sum_{j=1}^{m} \Phi_j(t_1, t_2) \le \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{t_1}^{t_2} F(\tau_1^*, \tau_2^*, ..., \tau_m^*, t + IL \times i) dt$$

$$= \int_{t_1}^{t_2} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} F(\tau_1^*, \tau_2^*, ..., \tau_m^*, t + IL \times i) dt$$
$$\leq n_{fp}(t_2 - t_1).$$

Hence, the lower bound is

$$n_{fpL} = \lceil \max_{\forall [t_1, t_2] \subseteq [0, IL]} \frac{\sum_{j=1}^{m} \Phi_j(t_1, t_2)}{t_2 - t_1} \rceil. \quad (3)$$

■

Since the total number of partitions is IL, only the time intervals $[t_1, t_2]$ that are subsets of $[0, IL]$ need to be considered. The number of sub-intervals in the time interval $[0, IL]$ is $IL(IL+1)/2$; therefore there are $IL(IL+1)/2$ different $\sum_{j=1}^{m} \Phi_j(t_1, t_2)$'s to compute in order to determine n_{fpL}.

A special case of Theorem 1 is presented in Theorem 4 of Sehwa [6]. The lower bound in [6] can be obtained from (3) by setting $t_1 = 0$, $t_2 = IL$ and $s = 1$. From the definition of $\Phi_j(t_1, t_2)$,

$$\Phi_j(0, IL) = \min_{\forall \tau_j} \sum_{i=0}^{\lceil \frac{T}{IL} \rceil} \int_{i \times IL}^{IL + i \times IL} f(\tau_j, t) dt = \int_0^T f(\tau_j, t) dt = s = 1;$$

thus,

$$n_{fpL} = \lceil \frac{\sum_{j=1}^{m} \Phi_j(t_1, t_2)}{t_2 - t_1} \rceil = \lceil \frac{m \times s}{IL} \rceil = \lceil \frac{m}{IL} \rceil.$$

Example

Assume for the example of Fig. 1 that $T = 9$ and $IL = 2$. For multiplication the maximum $\frac{\sum_{j=1}^{10} \Phi_j(t_1, t_2)}{t_2 - t_1}$ occurs at $[t_1, t_2] = [0, 1]$. Note that the maximum may not be unique. The following $\Phi_j(0, 1)$'s are needed[3] to obtain the lower bound.

$$\Phi_1(0,1) = \Phi_3(0,1) = \Phi_5(0,1) = \Phi_6(0,1) = \Phi_7(0,1) = \Phi_9(0,1) = 1$$

Thus, the lower bound is

$$n_{fpL} = \lceil \frac{\sum_{j=1}^{10} \Phi_j(0, 1)}{1 - 0} \rceil = \lceil \frac{6}{1} \rceil = 6.$$

Similarly, for addition

$$n_{fpL} = 2.$$

If we schedule the graph as shown in Fig. 3, then the lower bounds for addition and multiplication are the same as the upper bounds, and therefore, the lower bounds are optimal.

[3]All $IL(IL+1)/2$ time intervals need to be computed in order to determine the maximum.

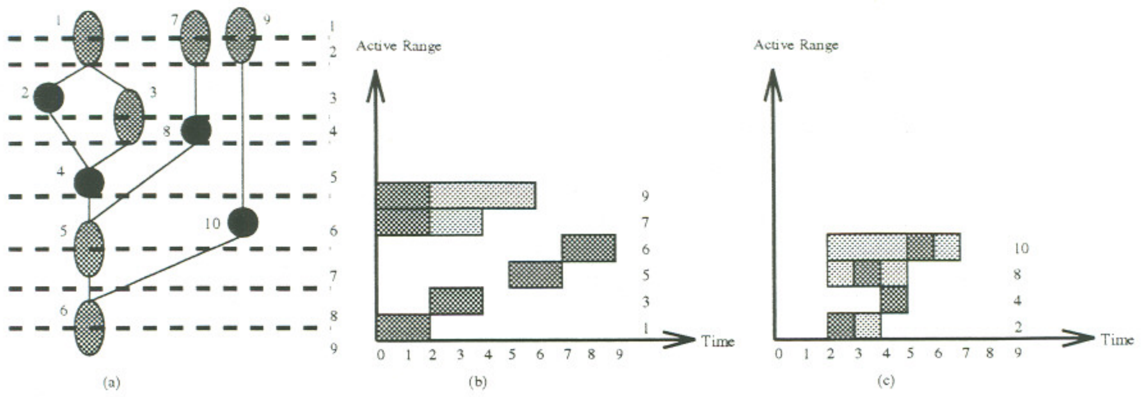

Fig. 3. (a) Schedule for the functional pipelined example, (b) the control step assignment for each multiplication, and (c) the control step assignment for each addition.

III Lower Bound on the Iteration Time

The objective of this section is to derive the lower bound on the loop iteration time given the number of resources R and initiation latency IL. The lower bound derived here can be further used in the development of other scheduling algorithms for functional pipelined data flow graphs. Some recent research on functional pipelined data flow graph scheduling [9] used the ASAP schedule under unlimited resources as their lower bound for refinement. If a better lower bound is used, the computation time will be reduced.

Given a data flow graph, a set of resources ($R_i's$) and initiation latency IL, our objective is to find the lower bound on the loop body iteration time. Care must be taken to make sure that the relationship between IL, R and the total number of operations m satisfy Lemma 2; otherwise functional pipelining is not possible.

Lemma 2 *Suppose there are m operations with execution time s in a data flow graph, and the task initiation latency is IL and the number of available resources is R. If*

$$\lceil\frac{m \times s}{IL}\rceil > R$$

then functional pipelining is not possible.

Proof: If $\lceil\frac{m\times s}{IL}\rceil > R$, then each partition needs more than R resources. Unless IL is increased, functional pipelining is impossible. ∎

In our method a lower bound is computed for each type of resource, and the final solution is the maximum of all the lower bounds of every type. For each type of operation the relationship between the lower bound on T, the number of resources R and the initiation latency IL is stated in the following theorem.

Theorem 3 *Given a functional pipelined data flow graph, the number of available resources R and the task initiation latency IL, and assuming IL, R and m satisfy Lemma 2, the minimum number of time steps required to complete the functional pipelined data flow graph is T. A lower bound on T is*

$$T_L = T_c + \Delta t,$$

where T_c is the critical path time,

$$\Delta t = \lceil \max_{\forall [t_1, t_2] \subseteq [0, IL]} \{ \frac{\sum_{j=1}^{m} \Phi_j^c(t_1, t_2)}{R} - (t_2 - t_1) \} \rceil,$$

and $\Phi_j^c(t_1, t_2)$ is defined in (2) with $T = T_c$.

Proof:

Given the time interval $[t_1, t_2] \subseteq [0, IL]$ and the number of resources R, the maximum operation load that can be executed by R resources in the time interval $\cup_{i=0}^{\lceil \frac{T_c}{IL} \rceil} [t_1 + i \times IL, t_2 + i \times IL]$ is $R(t_2 - t_1)$. On the other hand, as discussed before, the minimum operation load in the time interval $\cup_{i=0}^{\lceil \frac{T_c}{IL} \rceil} [t_1 + i \times IL, t_2 + i \times IL]$ is $\sum_{j=1}^{m} \Phi_j^c(t_1, t_2)$.

The operation load that can not be executed by R resources for the time interval $\cup_{i=0}^{\lceil \frac{T_c}{IL} \rceil} [t_1 + i \times IL, t_2 + i \times IL]$ is

$$\sum_{j=1}^{m} \Phi_j^c(t_1, t_2) - R(t_2 - t_1);$$

therefore, the minimum time increase $\delta(t_1, t_2)$ due to this uncovered operation load is

$$\begin{aligned} \delta(t_1, t_2) &= \frac{\sum_{j=1}^{m} \Phi_j^c(t_1, t_2) - R(t_2 - t_1)}{R} \\ &= \frac{\sum_{j=1}^{m} \Phi_j^c(t_1, t_2)}{R} - (t_2 - t_1). \end{aligned}$$

Note that if $\sum_{j=1}^{m} \Phi_j^c(t_1, t_2) < R(t_2 - t_1)$, then $\delta(t_1, t_2) < 0$ and R resources are sufficient to satisfy the computing requirements in the time interval $\cup_{i=0}^{\lceil \frac{T_c}{IL} \rceil} [t_1 + i \times IL, t_2 + i \times IL]$.

If we consider all time intervals, then

$$\Delta t = \max_{\forall [t_1, t_2] \subseteq [0, IL]} \{ \delta(t_1, t_2) \}.$$

Therefore, the lower bound on time is

$$T_L = T_c + \Delta t.$$

∎

There are $IL(IL + 1)/2$ different $\sum_{j=1}^{m} \Phi_j(t_1, t_2)$'s to compute in order to determine T_L.

Example

Assume in the example of Fig. 1 that the task initiation latency $IL = 2$, $R_+ = 2$ and $R_* = 6$. For addition, $\sum_{j=1}^{10} \Phi_j(t_1, t_2) - R_+ \times (t_2 - t_1)$ is always less than 0 which means 2 adders are sufficient for pipelining with $IL = 2$. Similarly, for multiplication $\delta(t_1, t_2) = 0$. Therefore, the lower bound $T_L = T_c + 0 = 9$.

IV Computational Complexity and Experiments

A *Complexity*

For Theorem 1 we have to consider $IL(IL+1)/2$ sub-intervals, and the complexity of computing Φ_j is $\lceil \frac{T}{IL} \rceil \times IL$. Therefore, the total complexity is $\frac{T}{IL} \times IL \times IL(IL+1)m = O(m(IL)^2T)$. For Theorem 3 the complexity is $O(m(IL)^2T_c)$, since only T_c is considered. The worst case performance is a conservative estimate, because normally at each step the number of computations is much less than m. The lower bounds for all types of operations can be computed together by going through $IL(IL+1)/2$ sub-intervals only once, at each step different types of operations can be computed independently and the total number of operations m is the sum of the numbers of all types of operations.

B *Experiments*

We have implemented the algorithm in a C program running on a SUN SPARCstation 2. The average CPU time for computing the lower bounds is 20ms. The lower bounds are compared with existing upper bounds found in recent research on functional pipelined scheduling algorithms [9]. Experimental results show that our lower bound is tight.

B1 *16-Point Digital FIR Filter*

The results presented here are for the functional pipelined 16-Point Digital FIR Filter example adopted from [6]. Table II tabulates the lower bound on resources under a performance constraint. The initiation latency and iteration time are given, and we compute the lower bounds for the number of adders and the number of multipliers. We also give the difference between the upper bound from [9] and our lower bound for each type of resource.

Table III lists the lower bound on time steps given the number of available resources and initiation latency. The lower bound on time is computed for addition and multiplication separately, and the maximum of these two is the final lower bound. We also compute the difference between the upper bound given in [9] and our lower bound on time steps.

B2 *Fifth Order Elliptical Filter*

The second benchmark is the fifth order elliptical filter example from [13]. It contains 26 additions and 8 multiplications. Table IV lists the results of the lower bound on the number of adders and multipliers as well as the difference between the upper and lower bounds. Table V tabulates the lower bound on time steps due to addition and multiplication, the final lower bound, and the difference between lower and upper bounds [9] on time steps.

V Conclusion

In this paper we have presented a new algorithm to compute lower bounds for the problem of performance constrained and resource constrained scheduling for pipelined data flow graphs. The lower bound is of great importance in optimal scheduling algorithms when the number of different operation types is large or the data flow graph is large. The results obtained here are not limited to the techniques used in high-level synthesis and their applications can be found in other areas of computer design where functional pipelining is utilized. Since the bounds are tight (shown by experiments and comparison to upper bounds), the lower bounds can be used to develop an efficient branch-and-bound algorithm, or they can be used in other scheduling techniques. Moreover, the lower bounds can be used to evaluate the quality of heuristic algorithms for the scheduling problem. Without the lower bound only a relative comparison between heuristics is possible.

The algorithm tends to a global optimization since it is not an iterative approach by nature, and has a complexity of $O(m(IL)^2T)$. The algorithm achieves excellent results as well as being the only one so far to provide a good lower bound for functional pipelined data flow graphs. The algorithm has the advantage of well defined bounds for optimal solutions such that it can effectively guide the design process and also provide a means to measure the quality of the solution while other methods can not. Although we do not have a theoretical proof of the tightness of our lower bounds, the experiments clearly indicate that they are very tight. Furthermore, much more information is considered during the computation of our lower bounds compared to previous work. Therefore, our bounds are tighter for arbitrary graphs. Future work may involve the development of a new scheduling algorithm using the lower bound discussed in this paper.

Table II. Lower bound on resources for 16-point digital FIR filter.

Initiation Latency	Iteration Time	n_L for Adder	n_U for Adder	$n_U - n_L$ for +	n_L For Mults	n_U for Mults	$n_U - n_L$ for *
1	6	15	15	0	8	8	0
2	6	8	8	0	4	4	0
3	6	5	5	0	3	3	0
4	6	4	4	0	2	2	0
5	7	3	3	0	2	2	0
8	10	2	2	0	1	1	0
15	16	1	1	0	1	1	0

Table III. Lower bound on iteration time for 16-point digital FIR filter.

Initiation Latency	Resource Adder	T_L Due to +	Resource Mults	T_L Due to *	Final Lower Bound	Upper Bound	Upper Bound - Lower Bound
1	15	6	8	6	6	6	0
2	8	6	4	6	6	6	0
3	6	6	3	6	6	6	0
3	5	6	3	6	6	6	0
4	4	6	2	6	6	6	0
5	3	7	2	6	7	7	0
8	2	10	1	10	10	10	0
15	1	15	1	10	15	16	1

Table IV. Lower bound on resources for fifth order elliptical filter.

Initiation Latency	Iteration Time	n_L for Adder	n_U for Adder	$n_U - n_L$ for +	n_L For Mults	n_U for Mults	$n_U - n_L$ for *
16	18	2	3	1	2	2	0
17	19	2	2	0	2	2	0
19	21	1	2	1	1	1	0

Table V. Lower bound on iteration time for fifth order elliptical filter.

Initiation Latency	Resource Adder	T_L Due to +	Resource Mults	T_L Due to *	Final Lower Bound	Upper Bound	Upper Bound - Lower Bound
16	3	16	3	17	17	18	1
16	3	16	2	18	18	18	0
17	2	18	2	18	18	19	1
19	2	18	1	21	21	21	0

References

[1] D.D. Gajski, N.D. Dutt and B.M. Pangrle, "Silicon Compilation," In *Proc. IEEE 1986 Custom Integrated Circuits Conf.*, pp. 102-110, May 1986.

[2] M.C. McFarland, A.C. Parker and R. Camposano, "The high-level synthesis of digital systems," In *Proceeding of the IEEE*, pp. 301-318, 78(2), Feb. 1990.

[3] P.G. Paulin and J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. on Computer-aided Design*, pp. 661-679, vol.8, no.6, June 1989.

[4] Miodrag Potkonjak and Jan Rabaey, "A scheduling and resource allocation algorithm for hierarchical signal flow graph," In *26th ACM/IEEE Design Automation Conference*, pp. 7-12, June 1989.

[5] Ki-soo Huang, et. al., "Scheduling and hardware sharing in pipeline data paths," In *Proceeding of IEEE*, pp. 24-27, 1989.

[6] N. Park and A.C. Parker, "Sehwa: A Software Package for Synthesis of Pipeline from Behavioral Specifications," *IEEE Trans. Computer-Aided Design,* pp. 356-370, Mar. 1988.

[7] D.J. Mallon and P.B. Denyer, "A New Approach to Pipeline Optimization," *Proc. European Conf. on Design Automation,* pp. 83-88, Mar. 1990.

[8] C.T. Hwang, Y.C. Hsu and Y.L. Lin, "Scheduling for Functional Pipelining and Loop Folding," *Proc. 28th Design Automation Conf.,* June 1991.

[9] T.F. Lee, C.H. Wu, D.D. Gajski and Y.L. Lin, "An Effective Methodology For Functional Pipelining," *Proc. 29th Design Automation Conf.,* June 1992.

[10] Roni Potasman, Joseph Lis, Alexandru Nicolau and Daniel Gajski, "Percolation Based Synthesis," *In Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 444-449, June 1990.

[11] Minjoong Rim and Rajiv Jain, "Lower-Bound Performance Estimation for High-Level Synthesis Scheduling Problem," *Proc. ICCD,* Oct. 1992.

[12] Robert Tarjan, *Data Structures and Network Algorithms*, SIAM, 1983.

[13] S.Y. Kung, H.J. Whitehouse and T. Kailath, *VLSI modern signal processing*, Prentice Hall, pp. 258-264, 1985.

[14] A.B. Barskiy, "Minimizing the number of computing devices needed to realize a computational process with a specified time," *Engineering Cybernetics*, No. 6, pp. 59-63, 1968.