A new Technique for Estimating Lower Bounds on Latency for High Level Synthesis[*]

Helvio P. Peixoto and Margarida F. Jacome
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin TX 78712-1084
{peixoto|jacome}@ece.utexas.edu

Abstract

In this paper we present a novel and fast estimation technique that produces tight latency lower bounds for Data Flow Graphs representing time critical segments of the application of interest. Our proposed technique can be used to compute a tighter earliest scheduling step for nodes (operations) in the Data Flow Graph and thus be used to improve the result quality of any technique requiring the computation of such ASAP values.

1 Introduction

In many CAD applications scheduling appears as a sub-problem – in High-Level Synthesis, for example, one is concerned with the generation of an RTL specification that best meets performance, area, and other constraints. Since the scheduling problem is NP-Complete,[1] the ability to generate tight lower-bounds on performance (i.e., latency) is important – it indicates, for instance, how far results produced by heuristic algorithms (i.e., upper bound estimates) might be from the optimum schedule. This paper presents a novel lower bound technique that produces tight latency lower bounds and outperforms the results produced by state-of-the-art approaches, such as the one reported in [5], when applied to a number of benchmark examples. The worst case complexity of the proposed technique is $O(n^2)$, where n is the number of operations in the application. Our method can handle multi-cycle and pipelined functional units.

2 Previous Work

In Rim and Jain,[4] an Integer Linear Programming (ILP) formulation relaxation of the general scheduling problem is proposed for generating performance estimates under resource constraints. The relaxation is based on the fact that each operation i in a DFG cannot start before time-step $\mathrm{ASAP}(i)$ and should not start later than $C - (c - \mathrm{ALAP}(i))$, where C is the number of control steps to minimize, and c is the critical path in the graph. The problem is decomposed into several sub-problems (one for each type of resource) and since they are solved independently, the bound resulting from the maximum taken over all sub-problems will provide a lower-bound to the original problem. An algorithm similar to list scheduling is proposed where the priority function is the increasing ALAP values of the operations. The complexity of the algorithm is $O(n + cC)$, where n is the number of nodes in the DFG. Experimental results demonstrate the superiority of this technique over other methods. However, one can find simple DFGs where this approach produces poor results.

A recursive technique with complexity $O(n^2 + nc^2)$ was proposed by Langerin and Cerny [2] for lower bound performance estimation. Langerin and Cerny's algorithm applies recursively the lower-bound algorithm of Rim and Jain to each node in the DFG (as for computing the ASAP), and then determines the lower bound performance of the complete schedule, again using Rim and Jain algorithm.

Tiruvuri and Chung [5] proposed yet another lower-bound technique which is based on the fact that operations can fall into three non-overlapping intervals: (a) operations that can be completed by cycle i; (b) operations that can be completed in the last j cycles; and (c) operations that should be scheduled between cycles i and j. This subdivision is based on the ASAP and ALAP values computed for each operation. By scanning all possible i and j such that $i + j \leq c$, they compute the number of cycles h necessary to perform the operations of a particular type in the (c) interval – this number is obtained by dividing the number of operations in that interval by the number of available resources for that type. The lower-bound is defined as the maximum value of $i+h+j$ among all types of operations. The complexity of the lower-bound estimator for an entire DFG is $O(n + c^2)$.

Rabay and Potkonjak [3] presented a survey on lower and upper bound estimation techniques and a thorough motivation for computing tight bounds. Most of the surveyed estimates depend on pre-computed values for the ASAP and ALAP time steps of each operation in the data flow graph. Indeed, most of the lower bounds found in the literature use such values to create a relaxed version of the scheduling problem. It follows that, by improving the estimate on the interval window in which an operation can be scheduled, one has the potential to improve any such lower bounds. So, in addition to its intrinsic value, our technique can be used in conjunction with any such lower bound estimates to improve their quality.

3 Lower Bound for Performance

A dataflow graph will be modeled by a directed acyclic graph, $G(I, E)$, where nodes I represent operations to be carried out on functional resources, and edges E represent data dependency between operations. A node $i \in I$ of type $o \in O$ has its delay denoted by $l(o)$ and the data introduction interval denoted by $l_p(o)$, in case the resources that can implement i are pipelined. If operation i is carried out in non-pipelined resources, the function $l_p(o)$ is defined as zero. For simplicity, we write $l(o) = l(\mathrm{node\_type}(i)) = l(i)$ if node i is of type o.

We now start the derivation of our lower bound on performance. The intuition behind our new lower bound, now on referred to as TASAP, is based on the simple fact that an operation i in the DFG can only start executing after all other operations in which i de-

[*]This work is supported by a National Science Foundation NSF CAREER Award MIP-9624321, NSF Award CCR-9901255 and Grant 003658-0649-1999 of the Texas Higher Education Coordinating Board Advanced Technology Program

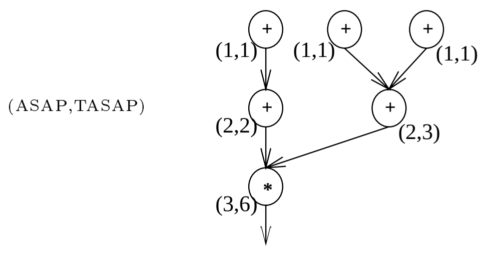

Figure 1: ASAP and TASAP values for a simple DFG, assuming a datapath with one adder and one multiplier, both taking one cycle

pends have terminated. We can write a recursive definition for the TASAP as follows:

$$\mathrm{TASAP}(i) = \begin{cases} 1, & \text{if } \mathrm{prec}(i) = \{\mathrm{source}\} \\ \max\left\{\max_{(h,i)\in E}(\mathrm{TASAP}(h) + l(h)), p(i)\right\}, & \text{otherwise} \end{cases} \quad (1)$$

where $p(i)$ is a lower bound on the time necessary to finish executing all operations that precede i and $\mathrm{prec}(i)$ is the set of immediate predecessors of node i.

The question that follows is how to compute the term $p(i)$ of Equation 1. One simple approach would be to count the number of operations of each type o, divide it by the number of available resources of type o, and finally multiply it by the latency required to execute one operation of type o. We could then assign to $p(i)$ the greatest value obtained among all the operations types. More formally:

$$p(i) = \max_{o\in O} \begin{cases} 0, \text{ if } k(i,o) \leq 0 \\ \left\lceil \frac{k(i,o)}{\mathrm{resource\_count}(o)} \right\rceil l_p(o) + l(o) - l_p(o), \text{ if } l_p(o) \neq 0 \\ \left\lceil \frac{k(i,o)}{\mathrm{resource\_count}(o)} \right\rceil l(o), \text{ otherwise.} \end{cases} \quad (2)$$

where $\mathrm{resource\_count}(o)$ is the number of resources of type o and $k(i,o)$ is the number of nodes of type o that precedes node i.

Equation 1 together with 2 is already an improvement over the traditional ASAP and can be computed in $O(n)$. Figure 1 shows a DFG and the improvement that one can obtain over the traditional ASAP for each node in the graph.

As discussed in the sequel, Equation 2 can still be improved, so as to better deal with cases where nodes of a given type are concentrated in different temporal time windows. The key observation is that when a large number of nodes (more than the number of available resources) are potentially ready to execute around a given clock step clk, the lower bound $p(i)$ would naturally (and mistakenly) assume that the execution of the nodes can be distributed from the first clock step. Our goal is to generate a $p(i)$ that properly distributes the operations in the interval defined by clk and the current step, i.e., the step at which node i can be scheduled.

We now discuss how to improve the quality of $p(i)$ by looking at successive time windows defined by the TASAP of all nodes that precede i. Given the TASAP for all nodes that precede node i, our algorithm to compute $p(i)$ examines the distribution of the nodes of a particular type in various temporal windows of different sizes. For instance, if the maximum TASAP of the immediate predecessors of node i is equal y cycles, one needs to examine y temporal windows for concentration of operations, as shown in Table 1.

Figure 2 shows the visual interpretation of the interval windows. The horizontal doted lines delineate the y possible windows intervals in which predecessor nodes of i can be scheduled for execution. For the sake of illustration, and without loss of generality, we assume that the in degree of i is two, being i_1 and i_2 its predecessors, and that $\mathrm{TASAP}(i_1) \leq \mathrm{TASAP}(i_2) = y$.

[1 to y]
[2 to y]
...
[$y-2$ to y]
[$y-1$ to y]
[y to y]

Table 1: Temporal windows

For each of such intervals, we count the number $k(i,o,x,y)$ of nodes of type o that precedes node i that can potentially be started in the interval window defined by x and y. Since node i cannot start until all nodes in any of these windows have completed execution, and all $k(i,o,x,y)$ nodes in each window [x to y] cannot start before control step x, it follows that node i cannot start before $\left(x + \left\lceil \frac{k(i,o,x,y)}{\mathrm{resource\_count}(o)} \right\rceil l(o)\right)$. If we compute this for all type o of operations and pick the greatest value, we have a lower bound on the start of node i. Note that it is not necessary to look at all combinations for the values of x and y, because some interval windows combinations dominate others. This is so because the dominated interval windows (those not listed in Table 1) cannot have more nodes than any of the windows listed in Table 1. The following equation formalizes this new lower bound $p(i)$:

$$p(i) = \max_{o\in O} \max_{1\leq x\leq y} \begin{cases} 0, \text{ if } k(i,o) \leq 0 \\ \left\lceil \frac{k(i,o,x,y)}{\mathrm{resource\_count}(o)} \right\rceil l_p(o) + l(o) - l_p(o), \text{ if } l_p(o) \neq 0 \\ \left\lceil \frac{k(i,o,x,y)}{\mathrm{resource\_count}(o)} \right\rceil l(o), \text{ otherwise.} \end{cases} \quad (3)$$

where y is the greatest TASAP for the predecessors of node i and $k(i,o,x,y)$ is the number of nodes i' of type o that precede i such that $x \leq \mathrm{TASAP}(i') \leq y$.

As it should be expected, the tighter lower bound corresponds always to $p(i)$ computed with Equation 3. In Figure 3 we show a comparison of the ASAP and the TASAP computed with $p(i)$ of Equations 2 and 3. For this example, while the plain ASAP leads to a bound of 12 cycles, the improved bound computed with Equation 3 is of 18 cycles. If DFGs contain several windows with high concentration of a particular type of operation (as the one shown in Figure 3), the proposed bound will lead to a chain of cumulative improvements over ASAP.

We have developed an efficient algorithm to compute Equation 3 recursively with Equation 1.(see Algorithm 1) For each node i we define a matrix $q(i,x,o)$ as the set of all predecessors nodes of i that can start at interval window starting at x. Initially, all elements in $q(i,x,o)$ are set to empty sets. Lines 1 to 6 compute Equation 1. Lines 7 to 11 compute $q(i,x,o)$ for each node i. Equation 3 is computed on lines 15 to 20. Lines 11 to 23 determine the concentration of operations in the several windows.

The overall complexity of Algorithm 1 is $O(n^2)$, where n is the number of nodes in the DFG. Lines 1 to 10 have three nested loops – this gives a worst case time complexity of $O(e \times n \times |O|)$,

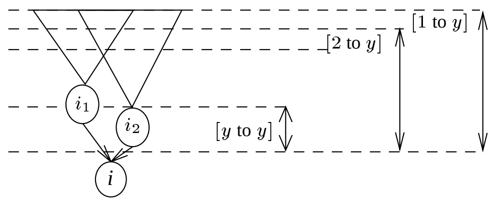

Figure 2: Pictorial description of interval windows

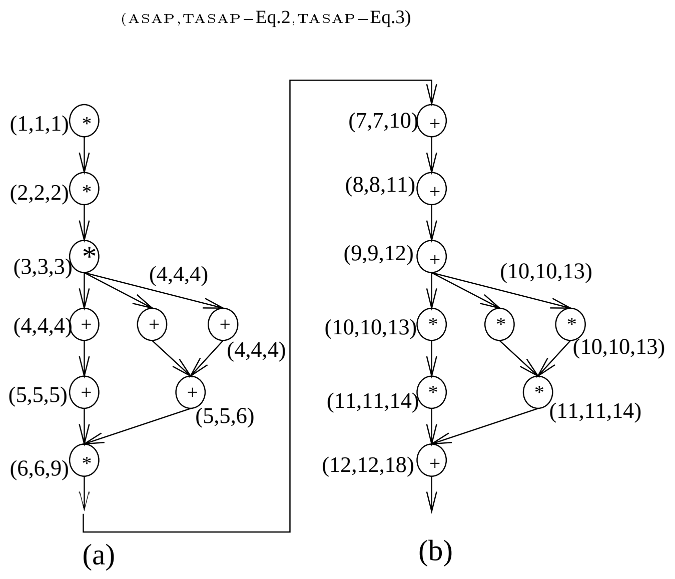

Figure 3: ASAP and TASAP computed with Equations 2 and 3. It is assumed one datapath with one adder and one multiplier, both taking one cycle

where e is the upper bound on the in-degree of nodes and $|O|$ is the number of different operation types in the DFG. Lines 11 to 23 have quadratic complexity of $O(n \times |O|)$. The recursion introduces a multiplication factor of n, giving a total complexity of $O(e \times n^2 \times |O| + n^2 \times |O|)$ for the entire DFG. Since e and $|O|$ are generally bounded by small constants, the worst case complexity of the algorithm can be rewritten as $O(n^2)$.

The proof that Algorithm 1 computes a valid lower bound is given by the following theorem:

Theorem 3.1 *Given a DFG $G(I,E)$ and a node $i \in I$, $\mathrm{TASAP}(i)$ computed with Equations 1 and 3 give a valid lower bound on the start time of node i.*

Proof If we assume for now that the lower bound on the start time of a node is given only by Equation 1, it is straight forward the proof, since $\mathrm{TASAP}(i) = \mathrm{ASAP}(i)$.

We now prove by induction on the size of the interval windows that $p(i)$ given by Equation 3 is a valid lower bound on the start time of node i.

The Base is for the case in which the start and end time of the interval windows are smaller or equal to one, i.e., all windows $[x$ to $y]$ such that $x \leq 1$ and $y \leq 1$. There is just one window in which the start and end time is smaller or equal to one: [1 to 1]. For this interval window, we determine all nodes in G of a particular type o that can be scheduled at this window, i.e., all nodes i' such that $1 \leq \mathrm{TASAP}(i') \leq 1$. Given this number and the number of available resources of type o, Equation 3 does produce a valid lower bound for windows where $x, y \leq 1$:

$$p(i) = \max_{o \in O} \begin{cases} 0, & \text{if } k(i,o) \leq 0 \\ \left\lceil \frac{k(i,o,1,1)}{\mathrm{resource\_count}(o)} \right\rceil l_p(o) + l(o) - l_p(o), & \text{if } l_p(o) \neq 0 \\ \left\lceil \frac{k(i,o,1,1)}{\mathrm{resource\_count}(o)} \right\rceil l(o), & \text{otherwise.} \end{cases} \tag{4}$$

Our Induction Hypothesis is that Equation 3 provides a valid lower bound on the start time of a node i if one considers only the interval windows $[x$ to $y]$ such that $x, y \leq m-1$, i.e., windows $[1$ to $m-1]$, $[2$ to $m-1]$, $\ldots$, $[m-2$ to $m-1]$, and $[m-1$ to $m-1]$, since all other windows are dominated by these

Algorithm 1: Tighter ASAP
Input: A DFG $G(I,E)$ and a node i.
Output: The lower bound on the start of node i

```
TASAP($G(I,E), i \in I$)
 1. if $\mathrm{pred}(i) = \mathrm{source}$
 2.   return 1
 3. else
 4.   $lb \leftarrow 0$
 5.   foreach $i_p$ such that $(i_p, i) \in E$
 6.     $lb \leftarrow \max(lb, \mathrm{TASAP}(G, i_p) + l(i_p))$
 7.     $q(i, \mathrm{TASAP}(G, i_p), \mathrm{node\_type}(i_p)) \leftarrow q(i, \mathrm{TASAP}(G, i_p), \mathrm{node\_type}(i_p)) \cup \{i_p\}$
 8.     for $x = 1$ to $x \leq \mathrm{TASAP}(G, i_p)$
 9.       foreach $o \in O$
10.         $q(i,x,o) \leftarrow q(i,x,o) \cup q(i_p,x,o)$
11.   foreach $o \in O$
12.     $k \leftarrow 0; intervalBegin \leftarrow 1$
13.     for $x = 1$ to $x \leq lb$
14.       $k \leftarrow k + |q(i,x,o)|$
15.       if $k = 0$
16.         $tmp \leftarrow 0$
17.       else if $l_p(i) \neq 0$
18.         $tmp \leftarrow \left\lceil \frac{k}{\mathrm{resource\_count}(o)} \right\rceil l_p(i) + l(i) - l_p(i) + intervalBegin$
19.       else
20.         $tmp \leftarrow \left\lceil \frac{k}{\mathrm{resource\_count}(o)} \right\rceil l(i) + intervalBegin$
21.       if $tmp < x$
22.         $k \leftarrow 0; intervalBegin \leftarrow x + 1$
23.     $lb \leftarrow \max(lb, tmp)$
24. return $lb$
```

(see Table 1). The Induction Hypothesis does address all nodes with $\mathrm{TASAP} \leq m-1$.

In the Induction Step we have to prove that for windows $[x$ to $y]$ such that $x, y \leq m$, Equation 3 holds. In order to prove the Induction Step we have to show that: (1) there is a valid expansion of the $m-1$ windows of the induction hypothesis to accommodate nodes with $\mathrm{TASAP} = m$; and (2) the new interval $[m$ to $m]$ also provides a valid lower bound. By addressing case (1) and (2), we create the set of interval windows shown in Table 1. To prove (1), note that $k(i,o,x,m) = k(i,o,x,m-1) + |M|$, where M is the set of all nodes such that $\mathrm{TASAP} = m$. Thus, the expansion of interval $[x$ to $m-1]$ to $[x$ to $m]$ is a trivial process that can only improve the quality of the bound (because now we have the chance to consider more nodes that can be scheduled before m, i.e., $k(i,o,x,m) = k(i,o,x,m-1) + |M|$). The added factor of $|M|$ does not harm the lower bound validity because the inclusion of the nodes in M into the set $k(i,o,x,m-1)$ is in fact a relaxation on the start time of the nodes in M. To prove (2) is trivial, since all nodes in M cannot start before clock cycle m. Since the nodes in M precede node i, the time necessary to execute the nodes in the window $[m$ to $m]$, added to the cycle they can start (i.e., cycle m), is a valid lower bound. Thus, the lower bound on a node i can be computed with Equation 3. □

4 Results

We have implemented ours and the algorithms in [5] so as to compare the quality of the produced results. The lower bound in [5] exhibits the same time complexity of our algorithm and has for most cases produced results superior to those of previous approaches, being thus representative of the state-of-the-art in this area. We considered several benchmark DFGs. For each DFG we considered several data-paths varying on number of resources, resource delays, and pipelining options.

Table 2 summarizes the results obtained by our algorithm and the algorithm in [5]. The first column identify the DFG. The second and third columns indicate the number or adders and multipliers, respectively. Under the Multiplier Delay column we consider various types of multipliers, namely with execution delays of one, two, and a 2 cycle pipelined implementation. For 41 cases, out of 90, the bound produced by our algorithm is tighter than the bound produced by [5] (see entries with an asterisk). In 76 cases our approach produced equal or better bounds when compared to [5].

5 Conclusions and Future Research

We have presented a simple and efficient technique for computing lower bound on the completion time of a DFG. The proposed technique can handle multi-cycle operations and pipelined functional units. For an entire DFG, our algorithm computes the lower bound in $O(n^2)$, where n is the number of nodes in the DFG. Because of the simplicity and speed of the proposed technique, one may use our technique in conjunction with any other lower bound approach that uses ASAP by substituting it with our proposed TASAP. The same principle can be used to derive a TALAP.

We are currently investigating an extension to our technique to contemplate chaining, conditional branches, loops, and prescheduled nodes.

References

[1] R. M. Garey and D. S. Johnon. *Computers and intractability, a guide to the theory of NP-Completeness*. W. H. Freeman and co., 1979.

[2] M. Langevin and E. Cerny. A recursive technique for computing lower-bound performance of schedules. *ACM Trasactions on Design Automation of Electronic Systems*, 1(4):443–456, Oct. 1996.

[3] J. M. Rabaey and M. Potkonjak. Estimating implementation bounds for real time dsp application specific circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(6):669–83, Jun. 1994.

[4] M. Rim and R. Jain. Lower-bound performance estimation for high-level synthesis scheduling problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):451–8, Apr. 1994.

[5] G. Tiruvuri and M. Chung. Estimation of lower bounds in scheduling algorithms for high-level synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 3(2):162–180, Apr. 1998.

			Multiplier Delay					
	Resources		1		2		2, pipelined	
DFG	+	*	Ours	[5]	Ours	[5]	Ours	[5]
FIR Filter	1	1	10*	9	17	17	11*	10
	1	2	10*	9	11*	10	11*	10
	1	3	10*	9	11*	10	11*	10
	2	1	9	9	17	17	10	10
	2	2	5	6	9	9	7*	6
	2	3	5	6	7*	6	7*	6
Beamforming	1	1	10	10	19	19	11	11
	1	2	8*	7	9	9	9*	8
	1	3	8*	7	9*	8	9*	8
	2	1	10	10	19	19	11	11
	2	2	5	5	9	9	6	6
	2	3	5*	4	7	7	6*	5
FFT	1	1	9*	7	11*	10	10*	8
	1	2	8*	7	9*	8	9*	8
	1	3	8*	7	9*	8	9*	8
	2	1	6	6	9	10	7	7
	2	2	5*	4	6	6	6*	5
	2	3	5*	4	6*	5	6*	5
AR Filter	1	1	17	18	33	34	18	19
	1	2	14*	13	17	18	15*	14
	1	3	14*	13	15*	14	15*	14
	2	1	17	18	33	34	18	19
	2	2	9	10	17	18	12	12
	2	3	9	9	12	12	12	12
Avenhous Filter	1	1	12	12	22*	21	13	13
	1	2	9	10	12*	11	11	11
	1	3	9	10	11	11	11	11
	2	1	12	12	22*	21	13	13
	2	2	8	8	12*	11	10	10
	2	3	8	8	10	10	10	10

Table 2: Comparative results for the proposed and state-of-the-art Lower Bounds