Hierarchical Scheduling in High Level Synthesis Using Resource Sharing Across Nested Loops

Abhijit Ghosh* **Sandeep K. Lodha*** **Ranga Vemuri**
Department of ECECS
P.O. Box 210030
University of Cincinnati
Cincinnati, OH 45221-0030
Email: {aghosh,slodha,ranga} @ececs.uc.edu

Abstract

This paper presents a resource-constrained scheduling algorithm for hierarchical behavioral specifications containing nested loops. The algorithm attempts to share resources across levels, to schedule operations that belong to different levels of the nested loop structures in the specifications as well as operations that belong to the same level. We compare the results of scheduling using our algorithm with those obtained using traditional list scheduling with no sharing of resources among different levels of the specification. These results show an average improvement of 23.47% in terms of number of control steps.

1 Introduction

An important problem in high-level synthesis is that of scheduling operations such that the total number of resources and/or time needed to execute the operations is minimal. Behavioral descriptions that contain loops (for and while) require complex scheduling algorithms. In this paper, we present an algorithm that schedules a description in the form of a control data flow graph (CDFG) having loops with arbitrary nesting.

Various approaches to scheduling in high-level synthesis exist [1, 2, 7]. Loop scheduling has been addressed in [4, 3]. Ku and De Micheli have proposed hierarchical scheduling for real-time constraints [5, 6]. In the presence of nested loops, the behavior specification is typically modelled as hierarchical CDFG structure where the CDFG in each level contains two types of nodes: simple nodes representing simple operations such as plus and minus and complex or loop nodes representing a loop structure which is in turn modelled by a CDFG at the next level in the hierarchy. Resource constrained scheduling algorithms typically assign a complex node in a CDFG to a control step and schedule other simple nodes in that CDFG at that control step only if resources of that type are not needed in the CDFG of the complex node, i.e., a complex node is scheduled in isolation from the parent CDFG. Thus, no resource sharing is allowed between different levels of the nested loop specification. In this paper, we present a scheduling technique that alleviates this restriction.

The paper presents an algorithm that schedules a CDFG, with a given resource set such that time to execute the CDFG is minimized. The algorithm tries to share resources with complex nodes. As mentioned above, in most of the algorithms in literature, a complex node is considered as a block and scheduled separately without considering the other complex nodes that can share resources with this complex node. Moreover in these algorithms a complex node does not share resources with other simple nodes that do not belong to this complex node. Hence, these algorithms do not exploit the possibility for complete resource sharing, thus giving sub-optimal (longer execution time) results. The following examples are illustrative.

Figure 1 shows a CDFG with a simple (node **2**) and a complex node (node **3**), which is in turn made up of some simple nodes (nodes **6,7,8**). The CDFG is represented hierarchically. The source and the sink of the main CDFG are node **1** and node **4** respectively. The main CDFG is called $CDFG_1$, and the complex node (node **3**) is represented by $CDFG_{11}$. Assume that this is to be scheduled with two adders. A simple scheduling algorithm would schedule the complex node **3** in the first control step of $CDFG_1$. The complex node is in turn scheduled in 2 control steps of $CDFG_{11}$. The maximum usage of adders over all control steps of $CDFG_{11}$ is 2, leading to an extra control step of $CDFG_1$ (second control step) to schedule node **2**. Thus a simple scheduling algorithm will schedule

[1]Names of the first two authors are in alphabetical order.

0-7695-0104-4/99 $10.00 © 1999 IEEE

the design in 2 control steps of $CDFG_1$. The following observation should be noted. The complex node does not use two adders in all of its control steps. In the second control step of the complex node only one adder is being used. This makes the scheduling of node **2** along with the complex node feasible. Thus node **2** can be scheduled along with the complex node in the second control step of the complex node. Thus the $CDFG_1$ can be scheduled in 1 control step of $CDFG_1$. This leads to a reduction in the overall execution time of the design (from 3 clock cycles to 2 clock cycles by inling the complex node, assuming that the complex node executes once and the delay of a simple operation is one clock cycle). Thus in this example we schedule simple node **2** with a complex node **3**, where both nodes belong to $CDFG_1$, i.e. they are of same level (level 1). Nodes **6,7,8** are at a level 2.

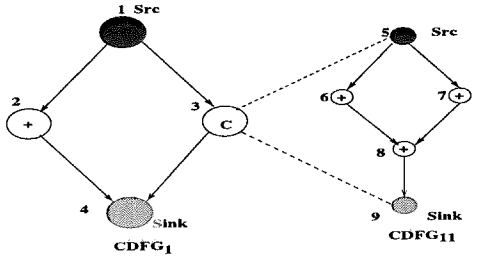

Figure 1: An Example CDFG to explain 1-Level Hierarchical Scheduling

Definition 1 *A node i is said to be scheduled in the kth control step of a $CDFG_p$ if it is scheduled in the kth control step relative to the source of $CDFG_p$.*

With reference to Figure 1 node **2** is scheduled in the second control step of $CDFG_{11}$ and node **3** is scheduled in the first control step of $CDFG_1$.

Figure 2 shows a CDFG with nested complex node. Assume that the resource set has two adders. With this resource constraint, node **2** cannot be scheduled with the operations of complex node **3**, but can be scheduled with the operations of complex node **9**, thus reducing execution times. Thus in this example we schedule a simple node along with a complex node of a higher level (one level higher), i.e., level 1 simple node (**2**) scheduled with level 2 complex node (**9**).

The above two examples illustrate that better results can be obtained if complex nodes share resources with the other simple nodes not belonging to that complex node.

Definition 2 *1-Level Hierarchical scheduling (HS_1) is defined as scheduling of simple nodes with complex nodes of same level, i.e. level k simple nodes scheduled with level k complex nodes.*

Definition 3 *K-Level Hierarchical scheduling (HS_K) is defined as scheduling of simple nodes of level N with complex nodes of level atmost $K + N$.*

Definition 4 *Max-Level Hierarchical scheduling (HS_{Max}) is defined as scheduling of simple nodes of level N with complex nodes of any level $\geq N$.*

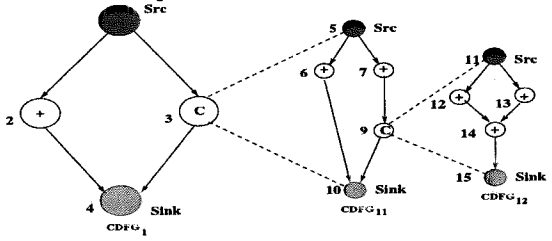

Figure 2: An Example CDFG to explain 2-Level Hierarchical Scheduling

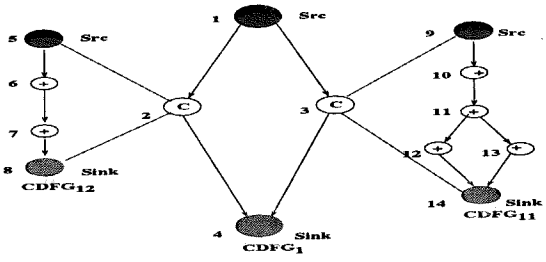

Figure 3: An Example CDFG to explain sharing of resources across complex nodes

The following example illustrates that better results can be obtained (in terms of execution times) if complex nodes share resources with the other complex nodes. Figure 3 shows a CDFG with two complex nodes. Assume that we need to schedule this CDFG with a resource constraint of two adders. A simple scheduling algorithm would schedule the main CDFG, say $CDFG_1$ in two control steps of $CDFG_1$, with $CDFG_{11}$ (in complex node **3**) scheduled in the first control step of $CDFG_1$ and $CDFG_{12}$ (in complex node **2**) scheduled in the second control step of $CDFG_1$ (assuming that this algorithm would schedule $CDFG_{11}$ in three control steps of $CDFG_{11}$). The following key observation should be noted for this example. One adder is not used in the first two control steps of $CDFG_{11}$, leading us to share this adder with $CDFG_{12}$. This leads to a schedule where $CDFG_{11}$ and $CDFG_{12}$ are scheduled in the first control step of $CDFG_1$, leading to a reduction in execution time.

Our algorithm gives better results for CDFGs having loops, where the number of iterations is unbounded. Due to unbounded nature of loops, they cannot be unrolled. Thus, the scheduling algorithms in literature would give sub-optimal results, as the possibility of resource sharing cannot be explored. In this paper we present a hierarchical scheduling algorithm that tries to maximize resource sharing. We compare our results with the List scheduling algorithm. Our algorithm reduces the execution times of the design by 23.47% on average.

The rest of the paper is organized as follows. Section 2 describes the Hierarchical scheduling algorithm in brief. This Section also describes the priority functions used. Results have been given in Section 3 and concluding remarks in Section 4.

2 Hierarchical Scheduling

2.1 Hierarchical Scheduling Algorithm

Given specification graph (in the form of a hierarchical CDFG) is traversed from source to sink and at each stage of the algorithm, two lists *Simple_Ready_List* and *Complex_Ready_List* are maintained, consisting of the ready nodes (nodes whose parents have been scheduled). Each of the complex node is associated with the corresponding CDFG. Assume that at a control step *Simple_Ready_List* has m nodes $s_1, s_2, \ldots, s_m$ and *Complex_Ready_List* has n nodes $c_1, c_2, \ldots, c_n$. Depth of a CDFG is its ASAP schedule length (a complex node is assigned to a control step). The algorithm is shown in figure 4. *slack* is an user input, where $0 \leq slack \leq 100$. *slack* allows complex nodes to be extended (beyond its depth) so that it can be scheduled with the given resource constraint. *Rlist* represents resources available at different control steps.

2.2 Priority Function

Ready List is a set of simple operations ready to be scheduled in a particular control step. All of them cannot be scheduled, due to resource constraints. So priorities need to be given to some operations.

Assume that at a particular control step say k, m operations $v_1, v_2, \ldots, v_m$ of type r are ready to be scheduled, with only $n < m$ functional units to execute operations of type r available. Thus we need to schedule n of these m ready operations. When we talk of a schedule S_k we mean assigning n operations out of the m operation to the control step k.

We do a Kernighan-Lin based search, to find the best schedule S_k. We need to partition the set of nodes $v_1, v_2, \ldots, v_m$ into two sets such that one partition consists of n nodes (Number of Resources available in this particular control step) and the other partition consists of $m - n$ nodes that are not scheduled in this control step. The various components that determines the schedule S_k are outlined here.

- Minimize excess unbalance (Number of operations that need to scheduled − Resource available) in later control step due to a schedule. This is an indication of increase in the number of control steps due to this assignment. For a particular schedule S''_k, if excess unbalance is less than another schedule S'_k, S''_k is preferred. Minimizing the excess unbalance, maximizes resource sharing.
- Maximize number of simple nodes that can be scheduled with complex nodes. This maximizes (tries to maximize) the sharing of resources with the complex nodes.
- Maximize the number of critical operations scheduled. This would try to minimize the schedule length.

```
procedure Hierarchical_Scheduler
(ReadySimpleList,ReadyComplexList,NumberOfResources)
begin
Sort (by depth) the complex nodes in ReadyComplexList
 MaxDepth ← Depth of the Complex Node having the
 highest depth
 -- Initialized all entries of Rlist with NumberOfResources
 Rlist ← Array[MaxDepth * (1 + slack/100)]
 While(There are more untried Complex nodes to be scheduled)
  Ci ← Unscheduled Complex node having the highest depth
  CSTEP ← 1
  NewReadySimpleList ← NULL
  NewReadyComplexList ← NULL
  if no complex node is scheduled till now
   Dused = di * (1 + slack/100)
  While (CSTEP ≤ Dused && Ci is not successfully scheduled)
   NewReadySimpleList ← Ready Simple operations in Ci
   NewReadyComplexList ← Ready Complex operations in Ci
   If(NewReadyComplexList=NULL)
    Schedule the ready operations using the priority functions
   Else
    -- Passing simple nodes of lower levels to be scheduled
    -- with complex nodes at higher levels in HS_Max
    In case of HS_Max NewReadySimpleList =
              NewReadySimpleList + ReadySimpleList
    Hierarchical_Scheduler
      (NewReadySimpleList,NewReadyComplexList,Rlist([CSTEP]))
   End If
   CSTEP++
  End While
  If(Ci is scheduled)
   Update the Rlist
   Remove Ci from the ReadyComplexList
   If Ci is the first complex node successfully scheduled then
    Dused = CSTEP-1
  Else
   Unschedule all the operations in Ci
  End If
 End While
 Schedule the unscheduled Simple nodes from ReadySimpleList
 with the available resources
end
```

Figure 4: *Hierarchical Scheduling Algorithm*

3 Results

We present here our results on 10 inputs (CDFGs). The attributes of the CDFG have been given in Table 1. The CDFGs are parameterized on 5 variables. They are (1) *AluOps*: Number of ALU type operations in the CDFG, (2) *MulOps*: Number of multiply type operations in the CDFG, (3) *MaxDepth*: The maximum depth of the CDFG (this includes any sub-CDFG), (4) *MaxLevel*: The maximum number of nesting levels allowed and (5) *MaxConc*: This determines the maximum level of concurrency in the CDFG, i.e., maximum number of children allowed for a parent. These attributes have been given in column 2 of

Input Graph	CDFG Parameters					ResourceSet		Slack(%)	Clock Cycles			Comparison	
	AluOps	*MulOps*	*MaxDepth*	MaxLevel	MaxConc	Alu	Mult.		HS_{Max}	HS_1	*LS*	HS_1, HS_{Max}	HS_1, LS
cdfg1	18	0	1	2	8	3	0	0	6	7	12	16.67	41.67
cdfg2	84	91	6	4	4	3	3	40	59	60	-	1.69	-
	84	91	6	4	4	5	5	50	49	49	61	0.0	24.48
cdfg3	121	85	5	3	4	2	2	80	82	83	-	1.21	-
	121	85	5	3	4	5	5	50	57	57	67	0.0	17.54
cdfg4	144	151	6	5	4	4	4	40	64	66	-	3.12	-
	144	151	6	5	4	8	8	50	51	52	64	1.96	23.07
cdfg5	146	141	6	4	4	7	7	50	59	80	-	35.59	-
	146	141	6	4	4	9	9	90	55	56	72	1.81	28.57
cdfg6	164	157	7	5	3	8	8	50	103	112	121	8.73	8.03
cdfg7	166	172	6	4	3	5	5	50	61	62	-	1.63	-
	166	172	6	4	3	11	11	20	52	52	60	0.0	15.38
cdfg8	236	256	7	4	4	8	8	80	72	73	-	1.38	-
	236	256	7	4	4	20	20	80	57	57	81	0.0	42.10
cdfg9	254	239	6	4	4	5	5	40	131	133	-	1.52	-
	254	239	6	4	4	9	9	50	113	113	132	0.0	16.81
cdfg10	358	347	7	4	4	4	4	60	135	138	-	2.2	-
	358	347	7	4	4	5	5	100	127	129	151	1.57	17.05

Table 1: Results for test CDFGs

Table 1. Resource set available is given in the 3^{rd} column. This column gives the number of ALUs and multipliers available. 4^{th} column has an entry for slack. This is explained in Section 2.1. Results for Max-Level Hierarchical scheduling(HS_{Max}), 1-Level Hierarchical scheduling (HS_1) and List scheduling (LS) are given in column 5. The Clock cycles have been calculated by inlining the complete CDFG and assuming that complex nodes are executed once. This has been done to compare the results. We compare HS_{Max} and HS_1 in the 6^{th} column. On an average, HS_{Max} outperforms HS_1 by 7.37%. In this column we also compare HS_1 with the List scheduling (LS) algorithm. We find that on an average HS_1 outperforms LS by 23.47%. In normal practice due to multiple iterative behavior of loops our algorithm will present significant improvement of execution times.

4 Conclusion

In this paper we have presented a resource constraint based hierarchical scheduling algorithm that tries to share resources with complex nodes. The algorithm tries to (1) schedule simple nodes with complex nodes and (2) tries to share resources across complex nodes. By incorporating (1) and (2) in a scheduling algorithm, we can increase resource sharing, thus reducing execution times. We have shown that HS_1 outperforms LS by 23.47% on an average. In the results presented here, we have considered only one invocation of the complex nodes (one iteration of a loop) and calculated the execution times in terms of the clock cycles. In normal practice due to multiple iterative behavior of loops our algorithm will present significant improvement of execution times. There is an overhead in the controller size compared to the controller required using LS. For our algorithm, we have a hierarchical model of the controller, where a separate controller is required for each complex node. The simple nodes are being scheduled across levels with other complex nodes (to which they do not belong) and the control signals required for these simple nodes do not depend on the invocation of these complex nodes. This leads to extra control generation to produce control signals for simple nodes scheduled across levels.

References

[1] R. Camposano and W. Wolf, "High Level VLSI Synthesis", Kluwer, 1991.

[2] Dan Gajski et al., "High Level Synthesis", Kluwer, 1992.

[3] E. Girczyc, "Loop Winding - A Data Flow Approach to Functional Pipelining", ISCAS, pp. 382-385, 1987.

[4] Goosens et al., "Loop Optimizations in Register Transfer Level Scheduling for DSP Systems", DAC, pp. 826-831, 1989.

[5] David Ku and G. De Micheli, "High Level Synthesis of ASICs under Timing and Synchronization Constraints", Kluwer, 1992.

[6] De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw Hill, 1994.

[7] Jan Vanhoof et al., "High Level Synthesis for Real Time Digital Signal Processing", Kluwer, 1993.