Branch-and-Bound Style Resource Constrained Scheduling using Efficient Structure-Aware Pruning

Mingsong Chen*, Saijie Huang, Geguang Pu*
Shanghai Key Lab of Trustworthy Computing
East China Normal University, Shanghai, China
Email: {mschen, sjhuang, ggpu}@sei.ecnu.edu.cn

Prabhat Mishra
Department of Computer & Information Sci. & Eng.
University of Florida, Gainesville, FL, USA
Email: prabhat@cise.ufl.edu

***Abstract*—Branch-and-bound approaches are promising in pruning infeasible search space during the resource constrained scheduling (RCS). However, such heuristic approaches only compare the estimated upper and lower bounds of an incomplete schedule to the length of the best feasible schedule at that iteration. This paper proposes an efficient pruning technique which can identify the fruitless search space based on the detailed structural scheduling information of the obtained best feasible schedule. The proactive nature of our pruning technique enables the pruning of the space which cannot be identified by the state-of-the-art branch-and-bound techniques. The experimental results demonstrate that our approach can drastically (up to two orders-of-magnitude) reduce the overall RCS time under a wide variety of resource constraints.**

I. Introduction

High-Level Synthesis (HLS) enables rapid generation of RTL hardware designs to satisfy performance, cost, area and power requirements [1]. In HLS, Electronic System Level (ESL) behavior descriptions are converted into *Data Flow Graphs* (DFGs), which are used as the intermediate representation by HLS algorithms. HLS involves two major tasks: scheduling and allocation. Scheduling refers to the assignment of operations to control steps (*c-steps*), and allocation binds the computation operations in DFGs to hardware resources. Currently, the scheduling problem is a major challenge in HLS, because it needs to make the trade-off among various constraints and explore a large number of possible alternatives to find an optimal or near-optimal design. In this paper, we focus on HLS under resource constraints, called Resource Constrained Scheduling (RCS). Given a DFG and a pre-defined set of functional or non-functional resources with specified overheads, RCS tries to find a schedule with minimum overall *c*-steps. Essentially, RCS is a scheduling problem with constraints of computation precedence and resource limits [9].

Since RCS is a NP-Complete problem, instead of enumerating all possible solutions, various approaches [3], [14] are proposed to efficiently prune infeasible or inferior candidates during the RCS search. Branch-and-bound (B&B) approaches [3] are widely used in existing RCS approaches to prune the search space. The B&B methods update the upper bound information of the schedule dynamically when encountering a better design candidate during the search. It utilizes such information to prune the inferior candidates which is worse than the best candidate obtained so far. Although the existing B&B algorithms are efficient in pruning the search space, they only investigate the upper and lower bound information of the scheduling to guide the search. In fact, its performance can be significantly improved by further pruning the search space using other potential avenues besides the upper and lower bounds.

In this paper, we propose an efficient algorithm for the RCS problem based on B&B methods, which investigate the DFG structure information of the up-to-date optimal scheduling. By comparing the scheduling time of partial operations, our approach can discard the incomplete schedules whose estimated lower-bound length equals to the length of the up-to-date optimal schedule. Therefore it can prune the search space in a proactive manner. Moreover, due to early detection of fruitless search, our approach can efficiently avoid deep recursive search that leads to drastic reduction in scheduling time.

This paper is organized as follows. Section II introduces the related work of RCS. Section III presents the related background and motivates the need for structure-aware pruning. Section IV proposes our approach which considers the structure information of the pruning techniques in detail. Section V presents our experimental results. Finally, Section VI concludes the paper.

* Corresponding authors.

II. Related Works

Unlike non-optimal HLS heuristic methods (e.g., list scheduling [3], force directed scheduling [5]) which can achieve near-optimal schedules with less overhead, this paper focuses on how to quickly obtain an optimal RCS result in HLS. As an early popular approach, *Integer Linear Programming* (ILP) models [2], [6], [7] are widely used in HLS scheduling. However, the number of variables in ILP models increases very fast with the size of DFGs. Therefore, solving tight resource constrained problems using ILP models may need prohibitively long time.

The *execution interval analysis* approach is another widely investigated approach for HLS scheduling. The basic idea is to perform the lower bound estimation before real scheduling. Since it can prune inferior designs, the overall scheduling time can be reduced. In [4], Ohm et al. presented a comprehensive technique for lower bound estimation. Shen and Jong [8] proposed a stepwise refinement algorithm for resource estimation based on execution interval analysis. Their approach can handle loop folding and conditional branches at the same time. Therefore it can produce a tight bound quickly. However, most of these methods can only give near-optimal solutions rather an optimal one.

To achieve an optimal resource constrained HLS schedule, an obvious and time-consuming way would be to enumerate all the possible designs. To effectively avoid unnecessary searching during scheduling, Narasimhan and Ramanujam [3] presented a B&B approach called BULB using both lower- and upper-bound information to prune the search space. Consequently, the overall scheduling time can be drastically reduced. It is important to note that, in [3], the lower-bound cost of the subproblem cannot be added to the partial schedule, which may lead to worse lower-bound values of the complete problem. Hansen and Singh [13] proposed an efficient B&B approach to reduce the scheduling time under multi-resource constraints. In [14], Wu et al. presented a novel in-place search algorithm based on a systematic offspring generation algorithm, which requires only a constant storage space during the traversal of the search tree.

Our approach is inspired by the work of [3], which can efficiently prune the search space in a B&B manner. To the best of our knowledge, our approach is the first attempt to utilize the structure information of the best schedule searched so far to prune the search space. A comparison between the BULB method [3] and our approach is presented in Section V.

III. Background and Motivation

A. Graph-Based Notations of RCS Problem

RCS employs DFGs (Data Flow Graphs) to describe the dependence of operations. A DFG is a DAG (Directed Acyclic Graph) $G=(V,E)$, where V is a set of vertices (nodes) designating functional operations with different types, E is a set of directed edges describing operation dependencies between nodes. For any two nodes $v_i, v_j \in V$, $\langle v_i,\ v_j\rangle \in E$ indicates that the operation of v_i must complete before the start of the operation of v_j. Consider the example DFG in Figure 1 that consists of 5 nodes and 4 directed edges. In a DFG, each v_i is tied with an operation op_i, and $type(op_i)$ indicates the type of functional unit that will be occupied by op_i. $delay(op_i)$ is used to denote time delay of op_i. An operation without any predecessors is an *input operation*, and an operation without any successors is an *output operation*.

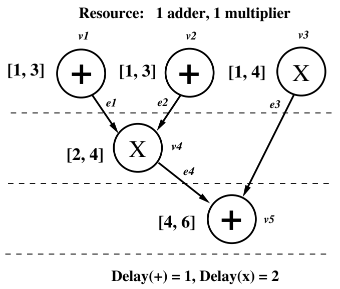

Fig. 1. An example of an HLS DFG

In RCS, the start time of an operation is regarded as the first *c-step* of its execution. The *As-Soon-As-Possible* (*ASAP*) notation estimates the start time of operation op_i. Since resource constraints cannot be taken into account before the real scheduling, in most approaches, $ASAP(op_i)$ estimates the earliest *c-step* when op_i can be started under the precedence constraints. Alternatively, the *As-Late-As-Possible* time of operation op_i, denoted by $ALAP(op_i)$ estimates the latest *c-step* when the operation op_i can be started. The notation $[ASAP(op_i), ALAP(op_i)]$, which is introduced in [3], indicates the execution intervals of operation op_i.

Besides basic graph notations, various graph theory notations are used to enable the RCS analysis. In this paper, we use G^r to represent the *transpose graph* of G by reversing all edge orientation. $G' = (V', E')$ is a *sub-graph* of $G = (V, E)$ if both $V' \subseteq V$ and $E' \subseteq E$. The sub-graph including nodes v_i and all its direct and indirect predecessors is denoted by $G_{pre}(v_i)$. The sub-graph with v_i as its source node is denoted as $G(v_i)$. The *length* of a path in a DFG indicates the number of nodes along the path. The *weighted length* of a path is the sum of operation delays of the nodes along the path, while the delay is determined by the type of nodes. The path in G with the longest length is called its *critical path* and the weighted path with the longest length is called *weighted critical path*. We use $CP_w(G)$ to denote the length of the weighted critical path of G, and $CP_l(G)$ for the length of the critical path of G. During the recursive B&B search of G, the scheduling order $\rho(G)$ of all operations is determined by the length of weighted critical path $CP_w(G(v_i))$ $1 \le i \le N$ in a non-ascending manner. As an example shown in Figure 1, either $\rho_1(G)$ = $<v_1,\ v_2,\ v_3,\ v_4,\ v_5>$ or $\rho_2(G)$ = $<v_1,\ v_2,\ v_4,\ v_3,\ v_5>$ can be a candidate of scheduling orders for G, since $CP_w(G(v_1)) = 4$, $CP_w(G(v_2)) = 4$, $CP_w(G(v_4)) = 3$, $CP_w(G(v_3)) = 3$, and $CP_w(G(v_5)) = 1$.

Level is an important notion to describe the structure of the RCS. In a DFG, the level of a node v, denoted by $Level(v)$, indicates the longest length from input nodes to v, that is, $Level(v) = CP_l(G_{pre}(v))$. For the example in Figure 1, the nodes are partitioned into 3 levels, and the level of v_5 is 3. When the level information of each node is fixed before the scheduling, we mark each node with an index from small level to large level. For a node v, $L_l(v)$ and $L_h(v)$ denote the smallest and largest indices of the nodes within the same level of v respectively. For example, the DFG in Figure 1 is already marked using the level information. For the nodes v_1 and v_2, we can get $L_l(v_1) = L_l(v_2) = 1$ and $L_h(v_1) = L_h(v_2) = 3$.

B. Scheduling Based on ASAP and ALAP

In RCS, *c-step* is the basic time unit. An operation will occupy a specific number of continuous *c-steps* for execution on corresponding functional unit during the scheduling. A *feasible scheduling* for a DFG tries to dispatch operations under the operation dependence constraints posed by the DFG and the limited resources by the implementation requirement. As described in Definition 3.1, a schedule for a given DFG is an assignment function S which dispatches each operation op_i at *c-step* $S(op_i) \in Z^+$. Let S be a feasible schedule, its length $le(S)$ is the largest finished time of all the operations, i.e., $le(S) = \max\{S(op_i) + delay(op_i) \mid op_i \in V\}$. A schedule is *optimal* if it is the up-to-date best with smallest length during the scheduling exploration. Among all possible schedules, the schedule with minimal length is regarded as the *global optimal schedule*.

Definition 3.1: Let $G = (V, E)$ be a DFG of a behavior specification, and OP be the set of operations corresponding to V, where $|V| = |OP| = N$. Assuming that the target implementation supplies M types of functions, $\Sigma = \{\pi_1, ..., \pi_M\}$, and $num(\pi_i)$ indicates the number of functional units of type π_i $(1 \le i \le M)$. A function $S : OP \rightarrow Z^+$ is a feasible schedule of G, iff it satisfies all the following conditions:

(1) If $\langle op_i, op_j\rangle \in E$ where $1 \le i, j \le N$, then $S(op_i) + delay(op_i) \le S(op_j)$ holds.

(2) For any time t and any operation of type π_j, $|\{op_i \mid type(op_i) = \pi_j \wedge ([S(op_i),\ S(op_i) + delay(op_i)] \bigcap [t, t]) \neq \emptyset\}| \le num(\pi_j)$. ■

Here condition (1) gives the precedence constraint posed by given DFG, and condition (2) indicates the resource constraints during the scheduling of DFG operations. We use $(op_i, S(op_i))$ to denote the scheduling pair for operation op_i. For example, the binary relation $\{(op_1, 1), (op_2, 2), (op_3, 2), (op_4, 4), (op_5, 6)\}$ is a feasible schedule with length 7 for the DFG in Figure 1. And the binary relation $\{(op_1, 1), (op_2, 2), (op_3, 1), (op_4, 3), (op_5, 5)\}$ is an optimal schedule with length 5.

C. Motivation

Based on the permutation of *c-steps* on the $[ASAP, ALAP]$ of all operations, B&B approaches [3], [13] can prune useless search space effectively. Besides $[ASAP, ALAP]$ intervals which restrict the search space of each operation, B&B approach uses other two important data structures to prune infeasible schedules: i) the optimal schedule S_{opt} which keeps the up-to-date optimal schedule, and ii) current schedule S which holds current (incomplete) schedule.

Figure 2a) illustrates how the B&B search makes the pruning. Since it is impossible to determine the optimal schedule length before all the enumerations are done, to restrict search space, the upper bound and lower bound of the optimal schedule are estimated. We use ω to record the upper-bound length of S_{opt}. Initially, ω equals to the length of a feasible schedule determined using the list scheduling approach. Then ω decreases dynamically when a faster feasible schedule is found during the RCS exploration. *GlobalLow* is the lower bound length of S_{opt}, which is calculated using the approach proposed in [10]. In Figure 2, *optimal* indicates the length of the global optimal schedule, which is in the range [*GlobalLow*, ω]. The current schedule S, which is an incomplete enumeration, also has two bound estimations. We use *lower* and *upper* to denote the lower-bound estimate and upper-bound estimate for S respectively. Note that ω is the length of a complete schedule S_{opt}, but in S, not all the operations are scheduled. Therefore, the *lower* and *upper* of S also can be considered as the lower-bound and upper-bound respectively for the schedules which have the same *c-steps* on each scheduled operations of S. In the B&B approach, if *lower* is larger than ω, the dispatching of the unscheduled operations will not continue, since *optimal* should be in the range $[GlobalLow, \omega]$. However, if *upper* is smaller than ω, a schedule which is better than S_{opt} is found. Consequently, S_{opt} will be replaced by the new schedule.

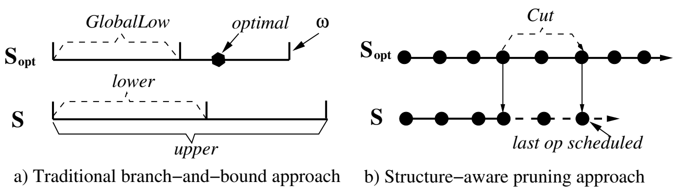
GlobalLow
optimal
ω
S_opt
lower
S
upper
a) Traditional branch−and−bound approach
Cut
S_opt
S
last op scheduled
b) Structure−aware pruning approach

Fig. 2. Pruning scenarios in B&B approach

From the above discussion, we can find that in B&B approach, the *lower*, *upper* and ω are the three main factors involved in pruning the useless search space. The pruning performance highly depends on the estimation algorithms for the factors *lower*, *upper* and the value of ω. However, the other useful information of the S_{opt} has not been fully investigated. For example, the structural scheduling information of S_{opt} has not been exploited by the pruning strategies of B&B approach. Figure 2b) shows the basic idea of the structure-aware pruning. In this figure, each black dot represents a functional operation, and the arrowed line indicates the dispatching order. Since the S_{opt} contains the useful scheduling exploration information, by comparing partial operations (i.e., the operations on a *cut*, which will be introduced in Definition 4.1) between S_{opt} and S, the further search of optimal schedule based on S can be terminated under some condition. For example, let $\rho_1(G)$ of the design shown in Figure 1 be the operation searching order, and assume that $S_{opt} = \{(op_1, 1)$, $(op_2, 2)$, $(op_3, 1)$, $(op_4, 3)$, $(op_5, 5)\}$ is the best schedule searched so far. If only the operations op_1, op_2, op_3 have been dispatched in the current schedule S such that $S(op_1) = 1$, $S(op_2) = 2$, and $S(op_3) = 2$, the subsequent recursive search of the current schedule can be terminated. This is because that op_1, op_2, and op_3 form a *cut* of G which enables the level-bound pruning heuristic proposed in Section IV-A. This is extremely useful when $\omega \in [lower, upper]$, since it extends the pruning capability of the traditional B&B approaches. Furthermore, it also can be used to stop the fruitless search earlier, which can prevent the deep recursive search in a proactive manner.

IV. Structure-Aware Pruning

Section III-C described the limitations of traditional B&B heuristics and motivated the need for efficient structure-aware pruning techniques to significantly improve the pruning performance. This section first presents our level-bound pruning heuristics that can utilize structural details. Next, we present our RCS algorithm using the proposed heuristics.

A. Level-Bound Pruning Heuristics

To further improve the pruning performance, we proposed a novel structure-aware pruning technique called *level-bound heuristic* which fully utilizes the precedence relation information of DFG operations.

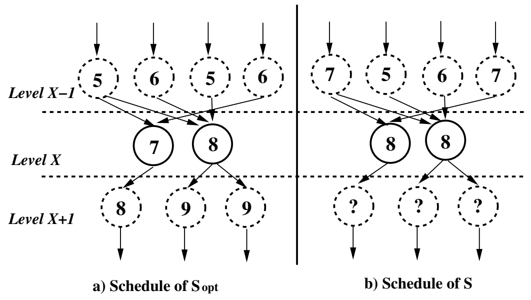

Fig. 3. Comparison of Two Schedules

Figure 3 depicts the basic idea of our level-bound pruning method, where a) presents an optimal schedule searched so far of a DFG G, and b) shows an incomplete schedule of G. Each node in the figures is marked with a number, which indicates the dispatch time of the corresponding operation. Suppose that in S the nodes of levels lower than or equal to level X have been scheduled and the other nodes (i.e., in levels larger than X) have not been determined. For traditional B&B approaches, they only compares the value between $le(S_{opt})$ and the value of both the *lower* and *upper* of S for pruning. Note that, in Figure 3, the dispatching time of all S_{opt}'s operations in the X_{th} level is better than the one of S. Such structural scheduling information (i.e., the *c-steps* of partial operations and corresponding operation precedence relations) can be used to prune the search space.

Definition 4.1: For a given DFG G, a *cut* is an edge set $EG = \{e_1, ..., e_n\}$ where all the following conditions hold:

1) For each edge e_i $(1 < i \leq n)$, there exists a path from some source node (i.e., a node without any incoming edges) to e_i.
2) For each source node in G^r, there exists a path from the source node to some $e \in EG$.

For a given *cut*, $Pre(CUT)$ denotes its precedent node set; $Imme(CUT)$ denotes the adjacent input nodes of the edges in the cut; and $Post(CUT)$ denotes the set of descendant nodes. A cut that separates the nodes in level X and the nodes in the levels lower than X is called an X_{th}*-level cut* of G. Assuming that cut_X is the X_{th}*-level cut* of G, $Imme(cut_X)$ is the X_{th} *complete level* of G. ■

In RCS, checking two schedules node by node is neither efficient nor necessary. Alternatively, we introduce the *cut* notation which enables the fast comparison and pruning during RCS. A *cut* divides a DFG into two disjoint parts where the first part is the set of all the precedent nodes of the cut while the second part is the set of all the descendant nodes of the cut. Consider the example in Figure 1, $\{e_3, e_4\}$ is the second level cut of G. However, $\{e_4\}$ is not a cut, since there is no path from e_4 to v_3 in G^r. In G, $Pre(\{e_3, e_4\}) = \{v_1, v_2, v_3, v_4\}$, $Imme(\{e_3, e_4\}) = \{v_3, v_4\}$ and $Post(\{e_3, e_4\}) = \{v_5\}$. The second complete level of G is $\{v_3, v_4\}$.

For a given DFG G and some given *cut*, if we have two schedules (i.e., S_1 and S_2) which have just finished the scheduling of the node set $Pre(CUT)$, our level-bound heuristics can be used to determine whether some incomplete schedule can be abandoned only based on the comparison of the scheduling information of the same *cut* of S_1

and S_2. Before proving the correctness of our level-bound heuristics (Theorem 4.1), we define *local optimal schedule* which is essentially based on the result of an incomplete schedule.

Definition 4.2: Let OP be the node set of a given DFG G. Assume that S is an incomplete schedule of G and node set $OP_{sch} = \{op_{i_1}, op_{i_2}, ..., op_{i_m}\}$ is a subset of scheduled operations. A *local optimal schedule* $S_{l,OP_{sch}}$ is a complete schedule which has fixed values for OP_{sch} and the optimal scheduling for the unscheduled operations (i.e., $OP \backslash OP_{sch}$). That means $S_{l,OP_{sch}}$ is a global optimal schedule in the search space $\Pi_{k=1}^{m}[S(op_{i_k}), S(op_{i_k})] \times \Pi_{op_j \in OP \backslash OP_{sch}}$ $[ASAP(op_j), ALAP(op_j)]$. ■

The local optimal schedule $S_{l,OP_{sch}}$ denotes that when a set of operations OP_{sch} are assigned with fixed *c-steps* in advance, it searches for the "global" optimal solution from the rest of the nodes. Obviously, $S_{l,OP_{sch}}$ is an optimal schedule but may not be the globally optimal schedule of the given DFG.

Theorem 4.1: Assume that S_1 and S_2 are two incomplete schedules of a given DFG $G = (V, E)$, and CUT is a cut of G. Let $S_{l_1,Pre(CUT)}$ and $S_{l_2,Pre(CUT)}$ be two different local optimal schedules based on CUT, and $Imme$ (CUT) be an operation set $\{op_{i_1}, op_{i_2}, ..., op_{i_k}\}$. Assume that $\omega_1 = le(S_{l_1,Pre(CUT)})$ and $\omega_2 = le(S_{l_2,Pre(CUT)})$. We can conclude that $\bigwedge_{1 \le j \le k}(S_1(op_{i_j}) \le S_2(op_{i_j}))$ && $\bigvee_{1 \le j \le k}(S_{opt}(op_{i_j})! = S(op_{i_j})) \Longrightarrow \omega_1 \le \omega_2$.

Proof: From the definition of local optimal schedule, we know that $S_{l_1,Pre(CUT)}$ is obtained from the search space $\Pi_{op_i \in Pre(CUT)}$ $[S_1(op_i),\ S_1(op_i)] \times \Pi_{op_j \in Post(CUT)}$ $[ASAP_{S_1}(op_j),\ ALAP_{S_1}(op_j)]$ while $S_{l_2,Pre(CUT)}$ is obtained from the search space $\Pi_{op_i \in Pre(CUT)}$ $[S_2(op_i),\ S_2(op_i)] \times \Pi_{op_j \in Post(CUT)}$ $[ASAP_{S_2}(op_j),\ ALAP_{S_2}(op_j)]$. Ignoring the scheduled nodes in $Pre(CUT)$, we schedule nodes in $Post(CUT)$ independently. Based on the scheduled nodes in $Pre(CUT)$, the ASAP values on the $Post(CUT)$ nodes of S_1 and S_2 can be dynamically updated. Due to the fact that if $\bigwedge_{1 \le j \le k}(S_1(op_{i_j})$ $\le S_2(op_{i_j}))$ && $\bigvee_{1 \le j \le k}(S_{opt}(op_{i_j})! = S(op_{i_j}))$, for each $op \in Post(CUT)$, we can get that $ASAP_{S1}(op)' \le ASAP_{S2}(op)'$ and schedule S_1 at time $S_1(op)$ has equal or more resource than schedule S_2 at time $S_2(op)$. For each $op \in Post(CUT)$, we do not change the ALAP value, which means $ALAP_{S1}(op) = ALAP_{S2}(op)$. Let $le(S_{1,Post(CUT)})$ and $le(S_{2,Post(CUT)})$ be the lengths of the global optimal schedules in the search space $\Pi_{op \in Post(CUT)}$ $[ASAP_{S_1}(op)'$, $ALAP_{S_1}(op)]$ and $\Pi_{op \in Post(CUT)}$ $[ASAP_{S_2}(op)', ALAP_{S_2}(op)]$, respectively. Based on the fact that $\Pi_{op \in Post(CUT)}$ $[ASAP_{S_1}(op)', ALAP_{S_1}$ $(op)] \supset \Pi_{op \in Post(CUT)}$ $[ASAP_{S_2}(op)', ALAP_{S_2}(op)]$, a local optimal schedule achieved in $\Pi_{op \in Post(CUT)}$ $[ASAP_{S_2}(op)', ALAP_{S_2}(op)]$ can also be achieved in $\Pi_{op \in Post(CUT)}[ASAP_{S_1}(op)', ALAP_{S_1}(op)]$. That means $le((S_{1,Post(CUT)})) \le le((S_{2,Post(CUT)}))$. Therefore, based on the CUT, if $\bigwedge_{1 \le j \le k}(S_1(op_{i_j}) \le S_2(op_{i_j}))$ and $le((S_{1,Post(CUT)})) \le le((S_{2,Post(CUT)}))$, we can conclude that $\omega_1 \le \omega_2$ holds. ■

In RCS, dynamically deciding whether an edge set is a cut or not is a time-consuming job. Since the static information of complete level structures introduced in Definition 4.1 implies the cut information, our approach adopts it for the *level-bound pruning* checking. Instead of comparing two schedules based on all scheduled operations, in our level-bound heuristics, the comparison is triggered only when all operations in one level have been scheduled.

Definition 4.3: For a given DFG G, S is an incomplete schedule and S_{opt} is the best schedule so far. Let OP_k be the operation set of the k_{th} complete level. Assume that all the operations in set OP_k have been scheduled. Based on Theorem 4.1, the *level-bound pruning* can be enabled when the following conditions hold.

1) $\forall op_i,\ op_i \in OP_k \to S(op_i) > 0$;
2) $\forall op_i,\ op_i \in OP_k \to S_{opt}(op_i) \le S(op_i)$;
3) $\exists op_i,\ op_i \in OP_k \to S_{opt}(op_i) < S(op_i)$. ■

In Definition 4.3, the condition 1) indicates that all the operations in k_{th} complete level are dispatched. The condition 2) means that all the operations of S in k_{th} complete level have no smaller *c-steps* than the corresponding operations of S_{opt}. The condition 3) denotes that at least one operation in k_{th} complete level has worse *c-step* than the corresponding operation in S_{opt}. The above conditions indicate that the comparison only needs to check all the operations in level k. For example in Figure 3, when the last operation in level X is scheduled, the level-bound checking will be invoked. Since all operations of S in X_{th} complete level have larger or equal *c-steps* than the ones in S_{opt}, the level-bound pruning can be enabled in this case.

For a given DFG, assume that the i_{th} complete level contains k operations, i.e., $OP_k = \{op_{i_1}, op_{i_2}, \ldots, op_{i_k}\}$, and all the operations in the i_{th} complete level have been scheduled. Let S_{opt} be the optimal schedule so far, and let S be the current schedule. When the *level-bound pruning condition* holds for the i_{th} complete level, there exists an operation op_{i_j} $(1 \le j \le k)$ such that $S(op_{i_j}) > S_{opt}(op_{i_j})$. In other words, the recursive procedure of B&B approach backtracks at least to the operation op_{i_j} with value $S_{opt}(op_{i_j}) + 1$. Let OP be the operation set which contains both OP_k and its precedent operations, and their *c-steps* are the same as the schedule S. According to Theorem 4.1, the local optimal schedule $S_{l,OP}$ cannot be better than S_{opt}. Since S_{opt} keeps the up-to-date best schedule along the B&B approach recursion, the current schedule S can be pruned.

Algorithm 1: Our Level-Bound Pruning Algorithm

```
Input:  i) $S_{opt}$, which is the best schedule searched so far
       ii) $S$, which stores the current incomplete schedule
      iii) $op_i$, which is the last dispatched operation of $S$
Output:  Whether $S_{opt}$ outperforms $S$.
LevelBound($S, S_{opt}, op_i$)  begin
    if  $i \neq L_h(op_i)$ then
        1. return $false$;
    end
    2. $OP = \{op_{i_1}, \ldots, op_{i_k}\} = (level(op_i))_{th}$ complete level;
    if $\bigwedge_{1 \le j \le k}(S_{opt}(op_{i_j}) \le S(op_{i_j}))$ &&$\bigvee_{1 \le j \le k}(S_{opt}(op_{i_j}) \neq S(op_{i_j}))$ then
        3. return $true$;
    end
    4. return $false$;
end
```

Algorithm 1 describes the implementation of our level-bound pruning approach. Step 1 checks whether all the operations in current level have been scheduled. Step 2 obtains the $(level(op_i))_{th}$ complete level. Step 3 asserts that S_{opt} outperforms S based on the level-bound condition, and step 4 asserts otherwise. Note that, to save time, the level bound based approach will be invoked only when all the operations in the $(level(op_i))_{th}$ complete level are dispatched.

B. HLS Scheduling using Our Approach

Compared to B&B approach which uses ω value only for pruning, our level-bound based heuristic employs the structure information of both current and optimal schedules. The level-bound pruning approach can be combined with B&B approach to explore more pruning chances, which can improve the RCS performance drastically.

Algorithm 2 presents our RCS approach that uses both B&B and level-bound heuristics. Step 1 initializes the variable *sucTimes* which indicates the times of the successful occupation of the resource by op_i. If op_i is the final unscheduled operation in $level(op_i)$ and $LevelBound(S, S_{opt}, op_i)$ returns $true$, it means that the current schedule stored in S is worse than S_{opt}. Then step 2 sets the global Boolean variable $jump$ to be $true$, which indicates that the level-bound checking may lead to some backtrack of more than one operations. Step 3 stops the c-step enumeration of op_i. If $LevelBound(S, S_{opt}, op_i)$ returns false, the operation op_i will be scheduled. Step 4 indicates that the operation successfully gets the intended resource. Steps 5 and 6 calculate the *lower* and *upper* for the current schedule. If *upper* is smaller than ω, then ω and S_{opt} will be updated in steps 7 and 8. Changing ω value will trigger the checking of early termination condition (i.e., achieving a schedule whose length equals to *GlobalLow*) in step 9. Step 10 does the runtime space shrinking by update the ALAP values of operations. If *lower* is smaller than ω, current operation will be scheduled. Step 11 assigns a *c-step* to operation op_i. Step 12 reserves resources required by the operation. Then op_{i+1} is processed recursively in step 13. When the search backtracks, the resource occupied by op_{i+1} is released in step 14. Steps 15 and 16 deal with the backtracks caused by the level-bound checking. When *sucTimes* equals to 1, it means that the current *step* is the smallest for op_i based on the scheduled operations. Since all the subsequent operations in the dispatching order have been enumerated, the search by increasing *step* by 1 is fruitless. Therefore, step 15 indicates that the search of op_i can be stopped. Step 16 continues to enumerate the following c-steps of the current operation. Finally, the algorithm returns one global optimal schedule and its length.

Algorithm 2: Structure-Aware B&B Pruning

```
Input: i) $D$, which is an HLS DFG with resource constraints;
       ii) $OP = \{op_1, \ldots, op_N\}$ in dispatching order;
       iii) $S_{opt}$, which is a feasible schedule of length ω;
       iv) $S$, which stores the current incomplete schedule;
       v) $jump = false$, which is a global Boolean variable;
Output: An optimal HLS schedule and its length ($S'$, ω')
OurMethod($D$, $OP$, $i$, $N$, $S_{opt}$, S, ω) begin
    if $i \leq N$ then
        1. $sucTimes = 0$;
        for $step = ASAP(op_i)$ to $ALAP(op_i)$ do
            if $LevelBound(S, S_{opt}, op_i)$ then
                2. jump = $true$;
                3. return ($S_{opt}$, ω);
            end
            if $Precedence(op_i) \wedge ResAvaible(step,\ type(op_i))$ then
                4. $sucTimes{+}{+}$;
                5. $lower = LBound(op_i)$;
                6. $upper = le(ListScheduling(OP, op_i))$;
                if $upper < ω$ then
                    7. $ω = upper$;
                    8. $S_{opt} = ListScheduling(OP, op_i)$;
                    if $ω == GlobalLow(D)$ then
                        9. Terminate ($S_{opt}$, ω);
                    end
                    10. $UpdateALAP()$;
                end
                if $lower < ω$ then
                    /* Dispatch the current operation */
                    11. $S(op_i) = step$;
                    12. $ResOccupy(step, type(op_i), delay(op_i))$;
                    13. $OurMethod(D, OP, i+1, N, S_{opt}, S, ω)$;
                    14. $ResRestore(step, type(op_i), delay(op_i))$;
                    if $jump$ then
                        if $sucTimes{=}{=}1$ then
                            15. return ($S_{opt}$, ω);
                        else
                            16. jump = $false$;
                        end
                    end
                end
            end
        end
    end
    return ($S_{opt}$, ω);
end
```

V. Experimental Results

To evaluate the effectiveness of our approach, we conducted various experiments with different kinds of resource constraints. We collected the following benchmarks from the *MediaBench* benchmark [11], which is a standard DSP benchmark suite: i) *ARFilter* with 28 nodes and 30 edges, ii) *Cosine 1* with 66 nodes and 76 edges, iii) *Collapse* with 56 nodes and 73 edges, and iv) *Feedback* with 53 nodes and 50 edges. We also used the benchmark *FDCT* with 42 nodes and 52 edges from [16]. By using the C programming language, we implemented both the BULB method and our approach that integrates the structure-aware pruning heuristic. For comparison, we also derived *ILP models* for RCS using IBM ILOG CPLEX CP Optimizer [12], which adopts the *branch-and-cut* heuristic method for efficient searching. All the experimental results were obtained on a Linux machine with Intel 2.0GHz processors and 3 GB RAM.

TABLE I
THE SETTINGS OF THE FUNCTIONAL UNITS

Functional Unit	Operation Class	Delay (unit)	Power (unit)	Energy (unit)	Area (unit)
ADD/SUB	+/-	1	10	10	10
MUL/DIV	*	2	20	40	40
MEM	LD,STR	1	15	15	20
Shift	<<,>>	1	10	10	5
Other	…	1	10	10	10

In this experiment, we consider functional unit constraints as well as non-functional constraints (i.e., power and area). Table I lists the corresponding settings for various types of functional operations used in the experiment.

TABLE II
SCHEDULING RESULTS UNDER FUNCTIONAL UNIT CONSTRAINTS

Design		ILP [12]	BULB	Ours	Speedup
name	# of +, ×	(sec.)	(sec.)	(sec.)	
ARFilter	1, 3	Timeout	0.34	0.14	2.43
	1, 4	Timeout	0.86	0.26	3.31
	1, 5	Timeout	0.85	0.26	3.27
	2, 3	2.32	0.01	<0.01	>1.00
Collapse	2, 1	Timeout	Timeout	234.76	>15.33
	2, 2	Timeout	Timeout	Timeout	NA
Cosine	1, 2	Timeout	105.67	23.38	4.52
	2, 2	Timeout	611.75	65.91	9.28
	3, 3	Timeout	0.02	<0.01	>2.00
Feedback	4, 4	Timeout	171.67	154.94	1.11
	4, 5	Timeout	Timeout	Timeout	NA
	5, 5	Timeout	5.53	4.96	1.11
FDCT	1, 2	Timeout	38.05	23.52	1.62
	2, 2	Timeout	210.22	18.67	11.26
	2, 3	Timeout	21.26	4.12	5.16
	2, 4	Timeout	4.31	2.00	2.16
	2, 5	Timeout	0.99	0.61	1.62
	3, 4	Timeout	0.64	0.51	1.25
	4, 4	Timeout	0.13	0.02	6.50

* Timeout means that the scheduling time is larger than 3600 seconds.

Table II presents the experimental results carried out with different functional unit constraints on the five benchmarks. The first column of the table has two sub-columns. The first sub-column indicates the name of the benchmarks. The second sub-column presents the functional unit constraint for the design. We use notation "x, y" to denote that only x adders and y multipliers are adopted for the RCS. For example, in the first row of $ARFilter$ design, "1, 3" denotes that one adder and three multipliers are used for the given design. Due to the space limit, we did not provide the number of other functional unit types. The second column presents the ILP solving time using the CPLEX CP Optimizer. In all benchmark items, only one of them (i.e., ARFilter design with 2 adders and 3 multipliers) can achieve the optimal result within the time limit. The third column presents the scheduling timing using the BULB approach [3]. The fourth column shows the results using our structure-aware approach. The last column indicates overall speedup using our approach over the BULB approach, i.e., $\frac{BULB}{Our\ Approach}$.

From this table, we can find that our approach can not only outperform the state-of-the-art ILP solver with the branch-and-cut heuristic, but also it can drastically improve the RCS performance using the B&B style searching. For instance, when handling the design *Collapse* under the constraint of two adders and one multiplier, BULB method needs more than 3600 seconds for scheduling, which is not acceptable in many scenarios. However, our approach requires only 234.76 seconds. Generally, during RCS, the cost of the detection and the check of level-bound information is negligible. However, it can effectively prune the space where the search is fruitless. Therefore, in most cases, our approach can achieve a significant improvement.

The area and the power are two key issues in hardware design. The scheduling under these two constraints can be considered as a variant of the time-minimum resource scheduling [13]. Since both area and power can be treated as special kinds of resources, our proposed structure-aware pruning can also be used to promote the scheduling performance under such non-functional constraints. We did the experiment with the designs given in Table II using these two constraints. Due to the space limit, we only present the results for the FDCT design. All the other designs have the similar conclusion based on their results.

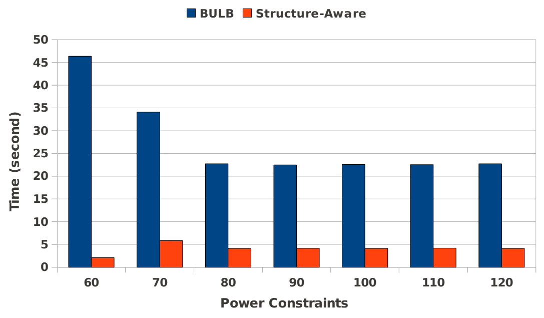

Fig. 4. Scheduling results with an area of 140 units

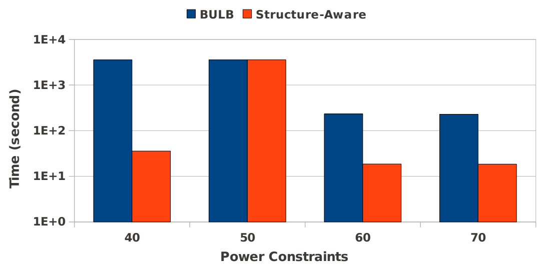

Fig. 5. Scheduling results with an area of 100 units

Figure 4 shows the RCS result using the BULB approach and our approach. Under the area constraint of 140 units, we check the RCS with different power constraints (from 60 to 120 with an increment of 10). We can find that when the power is taken into consideration during the RCS (i.e., the case with a power constraint 60), our approach can get the maximum speedup (more than 22 times). It is important to note that, for the design of *FDCT* with a 140-unit area constraint, the threshold of the power is 80 units. When the power limit is larger than 80 units in this case, the power constraint will not alter the outcome. Therefore, we can find that the last five instances in Figure 4 show the similar results. We also tried to do RCS for the FDCT design with a smaller area constraint (i.e., 100 units). As shown in Figure 5. We can find a similar trend in Figure 4. In this case with a power constraint of 50 units, none of the BULB approach and our method can figure out any optimal schedule within the time limit (i.e., 3600 seconds). Although it shows the same result in this case, in fact the results cannot be compared. From above figures, we can find that our approach is very promising in pruning the search space of designs with power and area constraints. Our approach can achieve up to 101 times improvement (for the case with 100-unit area and 40-unit power) compared to the BULB approach [3].

VI. Conclusion

This paper proposes a structure-aware approach to prune search space efficiently during resource-constrained HLS scheduling. Unlike existing B&B algorithms which are based on the comparison of upper and lower bounds between optimal schedule searched so far and incomplete searching schedules, our approach investigates structural scheduling information of the optimal schedule candidates during the search to prune other non-optimal schedules in a proactive manner. Our approach is orthogonal with the exsiting B&B approaches. The synergy of them can promote the overall RCS performance drastically by several orders of magnitude. Experimental results using various benchmarks with different resource constraints show that our method can achieve significantly better performance than existing state-of-the-art B&B techniques.

Acknowledgment

This work was partially supported by the grants from Natural Science Foundation of China (No. 61202103, 91118008, and 91118007), Open Project of Software/Hardware Co-design Engineering Research Center of MoE (No. 2013001), Shanghai Knowledge Service Platform Project (No. ZF1213), and Civil Aerospace Project 125.

References

[1] P. Coussy, D. Gajski, M. Meredith and A. Takach. An introduction to high-level synthesis. *Design & Test of Computers*, 26(4):8–17, 2009.
[2] M. Langevin and E. Cerny, “A Recursive Technique for Computing Lower-Bound Performance of Schedules”, in *Proc. of ICCD*, 1993, pp. 16–20.
[3] M. Narasimhan, J. Ramanujam. A fast approach to computing exact solutions to the resource-constrained scheduling problem. *ACM TODAES*, 6(4):490–500, 2001.
[4] S. Y. Ohm, F.J. Kurdahi, and N. Dutt. Comprehensive Lower Bound Estimation from Behavioral Descriptions. In *Proc. of ICCAD*, 182–186,1994
[5] P. Paulin and J. Knight, “Force-directed scheduling for behavioral synthesis of ASICs”, *IEEE TCAD*, vol. 8, no. 6, pp. 661–679, 1989.
[6] M. Rim and R. Jain. Lower-bound performance estimation for the high-level synthesis scheduling problem. *IEEE TCAD*, 13(4):451–458, 1994.
[7] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. *ACM JETCS*, 3(4), 2008.
[8] Z. Shen and C. Jong. Lower Bound estimation of hardware resources for scheduling in high-level synthesis. *Journal of Computer Science and Technology*, 17(6):718–730, 2002.
[9] J. A. Stankovic, M. Sprui, M. D. Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):15–25, 1995.
[10] G. Tiruvuri and M. Chung. Estimation of lower bounds in scheduling algorithms for high-level synthesis. *ACM TODAES*, 3(2):162–180, 1998.
[11] Media Benchmarks. http://express.ece.ucsb.edu/benchmark/.
[12] IBM ILOG CPLEX CP Optimizer V12.3. http://www-01.ibm.com /software/commerce/optimization/cplex-cp-optimizer/index.html.
[13] J. Hansen and M. Singh. A fast branch-and-bound approach to high-level synthesis of asynchronous systems. In *Proc. of ASYNC*, 107–116, 2010.
[14] C. Yu, Y. Wu and S. Wang. An in-place search algorithm for the resource constrained scheduling problem during high-level synthesis. *ACM TODAES*, 15(4), 2010.
[15] A. H. Timmer and J. A. G. Jess. Execution interval analysis under resource constraints. In *Proc. of ICCAD*, 454–459, 1993.
[16] H. Steve, B. Forrest. Automata-Based Symbolic Scheduling for Looping DFGs. *IEEE Transactions on Computers* , 50(3):250–267, 2001.