Tighter lower bounds for scheduling problems in high-level synthesis

M. Narasimhan and J. Ramanujam*
Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

August 1997

Abstract

This paper presents new results on lower bounds for the scheduling problem in high-level synthesis. While several techniques exist for lower bound estimation, comparisons among the techniques have been experimental with few guarantees on the quality of the bounds. In this paper, we present new bounds and a theoretical comparison of these with existing bounds. For the resource-constrained scheduling problem, we present a new algorithm which generalizes the bounding techniques of Langevin and Cerny [13] and Rim and Jain [18]. This algorithm is shown to produce bounds that are provably tighter than other existing techniques. For the time constrained scheduling problem, we show how to generate the tightest possible bounds that can be derived by ignoring the precedence constraints by solving a linear programming formulation. These bounds are therefore guaranteed to be tighter than the bounds generated by the techniques of Fernandez-Bussell [4] or Sharma-Jain [19]. As a result, we show that the linear relaxation of the ILP formulation of the time constrained scheduling problem produces tighter bounds than the two techniques mentioned above.

Keywords: Design automation, high level synthesis, resource constrained scheduling, time constrained scheduling, lower bounds, tighter bounds.

Corresponding author:

J. Ramanujam, Associate Professor
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803-5901

Email: jxr@ee.lsu.edu
Phone: (504) 388-5628
Fax: (504) 388-5200

*Corresponding author; J. Ramanujam is supported in part by an NSF Young Investigator Award CCR–9457768 and an NSF Research Initiation Award CCR–9210422.

1 Introduction

High-level synthesis (HLS) is the translation of a behavioral level specification into a register-transfer level description. A behavioral level description is typically converted to a data flow graph (DFG) that is used as an intermediate representation in the HLS system. A central task in HLS is scheduling, which is the mapping of operations of the data flow graph to appropriate control steps (c-steps), where a c-step is one cycle of the system clock. Often scheduling is the first task in an HLS system. The mapping of operations to c-steps must not violate precedence constraints between operations in a DFG. In addition to the precedence constraints in the DFG, a designer may face additional design criteria such as latency (time), area, clock cycle, etc.

The Resource-Constrained Scheduling (RCS) problem involves the minimization of the number of c-steps required to execute all the operations given that the number of instances of each resource available is fixed. The Time-Constrained Scheduling (TCS) problem involves minimizing the number of instances of each resource required when the number of time steps is given[1]. Unfortunately both these problems are NP-complete [20, 1]; so the best known algorithms for solving these problems have exponential time complexity. Because of the intractable nature of these problems, heuristics are often used. While heuristics very often produce solutions which are near optimal, if not optimal, these methods do not produce any performance guarantee, and very often, it is desirable to be able to bound the quality of the solutions produced. An easy way of doing so is to compute a lower bound on these quantities for all schedules. So, for example, if the lower bound on the latency of all schedules in the RCS problem equals the latency of the schedule produced by a heuristic, then clearly, the schedule returned by the heuristic has optimal latency. Verifying the optimality of schedules in this manner depends crucially on the quality (or tightness) of the the bounds obtained. Such a technique is useful for improving the performance of *exact scheduling techniques* such as branch-and-bound and Integer Linear Programming (ILP).

In this paper we present techniques to calculate tight lower bounds on the latency of schedules for the RCS problem and on the number of instances of each resource class required for the TCS problem.

There has already been a considerable amount of work done in the area of bounding techniques for the scheduling problem. Chaudhuri and Walker [3] and Hu et al. [8, 6, 7] have shown algorithms

[1] A third variant, TRCS, determines the existence of a feasible schedule subject to both time and resource constraints. We do not discuss TRCS in this paper.

for computing lower bounds for the Time-Constrained Scheduling problem. Rim and Jain [18] and Langevin and Cerny [13] have shown algorithms for computing lower bounds for the Resource-Constrained Scheduling problem. Several authors [9, 11, 10, 12, 16, 17, 19] have shown different algorithms that compute lower bounds for both these problems or for related problems. However, in most cases, it is difficult to guarantee that any one of these techniques will always produce a tighter bound than the others, and all comparisons between these algorithms so far have been experimental in nature. In this paper, in addition to deriving tighter bounds, we present a theoretical comparison of several techniques.

This paper is organized is follows. In Section 2, we discuss the problems considered in this paper along with the associated terminology and notation. Section 3 presents tighter lower bounds for the resource-constrained scheduling problem and shows that the bounds obtained by our technique are better than the bounds from other techniques. In Section 4, we derive an ILP formulation of the bounding problem for time-constrained scheduling and show that the solution to the linear relaxation of this ILP problem can be used to derive the solution to the the ILP problem (of bounding). Thus, the bounds derived using this are tighter than bounds derived using existing techniques. Section 5 concludes with a summary and discussion.

2 Preliminaries

Consider a DFG with nodes corresponding to a set of operations (tasks) T, and the edges corresponding to a set of dependencies E between the operations. It is assumed that we are provided with clock length $STEP$ and that an operation can start executing only at some time instant $c \cdot STEP$ for some $c \in \mathcal{N}^+ = \{1, 2, \ldots\}$. The time interval from $c \cdot STEP$ to $(c+1) \cdot STEP$, $[c \cdot STEP, (c+1) \cdot STEP)$, is called the c^{th} time step; an operation that begins executing at time instant $c \cdot STEP$ is said to be scheduled (to begin executing) at time step c. Each operation t_i takes $\mathsf{delay}(\mathsf{t_i})$ time steps to complete executing. Each dependency $\langle t_i, t_j \rangle \in E$ implies that operation $t_i \in T$ has to be completed before operation t_j can begin executing. So, if operation t_i is begins executing at time step k, then $\langle t_i, t_j \rangle \in E \Rightarrow$ operation t_j can begin executing only on or after time step $k + \mathsf{delay}(t_i)$. We assume that there are a number of resources classes (adders, multipliers etc.) and that each operation can execute only on one resource belonging to a particular class[2] We

[2]The problem of determining the resource class on which each operation is scheduled is called the *module selection problem;* we will assume that module selection is already completed.

assume that the number of such resource classes is R, and that the if any resource constraints are specified, then they are of the form that the number of simultaneously active (allocated to some task) resources of class r is no greater than M_r. We will let the function $\mathsf{Resource} : T \rightarrow \{1, 2, \ldots, R\}$ be a mapping of operations to resource classes.

Given the problem of scheduling the tasks T with dependencies E between these tasks, we model it as a directed acyclic graph $G = (T, E)$, where T represents the vertices of G and E the (directed) edges of G. Each edge in $\langle t_i, t_j \rangle \in E$ is has a weight equal to $\mathsf{delay}(t_i)$. A schedule of G is a function $\mathcal{S} : T \rightarrow \mathcal{N}^+$, where each operation t_i is scheduled to begin execution at time step $\mathcal{S}(t_i)$. Note that in order for the schedule to be valid, we require that the schedule satisfy all the dependence constraints, and that no resource and time constraints (if any) are violated. The latency of a schedules is defined to be the maximum completion time of all operations under that schedule; therefore, $\mathsf{Latency}(\mathcal{S}) = \max\limits_{t_i \in T} \{\mathcal{S}(t_i) + \mathsf{delay}(t_i)\}$.

We use $T_{pred}(t_i)$ (resp. $T_{succ}(t_i)$) to denote the set of direct and indirect predecessors (resp. successors) of task t_i. $G_{pred}(t_i)$ (resp. $G_{succ}(t_i)$) is the subgraph induced by $T_{pred}(t_i)$ (resp. $T_{succ}(t_i)$). Clearly, all elements in $T_{pred}(t_i)$ must have completed execution before t_i can be scheduled, and no operation in $T_{succ}(t_i)$ can begin execution before t_i completes executing.

Longest weighted directed paths in G are called *critical paths* of G. The length of a critical path of G is denoted by $\mathsf{CP}(G)$. A lower bound on the execution time of G is clearly $\mathsf{CP}(G) + 1$. For any operation (vertex) $t_i \in G$, we define

$$\mathsf{ASAP}(t_i) = \mathsf{CP}(G_{pred}(t_i)) + 1$$

and

$$\mathsf{ALAP}(t_i, UB) = UB - \mathsf{CP}(G_{succ}(t_i))$$

We require that the value of UB be at least the latency of the optimal schedule; thus UB is an upper bound on the latency of the optimal schedule. For the RCS problem, the value of UB is not specified. However, since the latency of the optimal schedule can not exceed that of any other valid schedule, we have simply to obtain some valid schedule (possibly by using a heuristic) to get an upper bound on the optimal latency.

When there is any doubt about which graph (or subgraph) we are referring to, we will explicitly indicate the graph (or subgraph) under consideration by the use of a subscript, e.g.,

$\mathsf{ASAP}_{G_{pred}(t_i)}(t_j)$.

$\mathsf{ASAP}(t_i)$ (As Soon As Possible) represents a lower bound on the time step that operation t_i can be scheduled at in any valid schedule. Similarly, $\mathsf{ASAP}(t_i, UB)$ (As Late As Possible) represents an upper bound on the time step at which operation t_i can be scheduled in any (valid) schedule that completes execution of all operations by step UB. The values of $\mathsf{ASAP}(t_i)$ and $\mathsf{ALAP}(t_i, UB)$ represent exact bounds only in the absence of constraints. Suppose we define

$$\mathsf{ASAP}_{ideal}(\phi, t_i) = \min_{t_i \in T, \mathcal{S} \in \phi} \mathcal{S}(t_i)$$

and

$$\mathsf{ALAP}_{ideal}(\phi, t_i) = \max_{t_i \in T, \mathcal{S} \in \phi} \mathcal{S}(t_i)$$

where ϕ represents the set of schedules we are interested in. For example, if ϕ represents the set of schedules in which all operations are scheduled to complete before UB while satisfying dependence constraints, then $\mathsf{ASAP}_{ideal}(\phi, t_i) = \mathsf{ASAP}(t_i)$ and $\mathsf{ALAP}(\phi, t_i) = \mathsf{ALAP}(t_i, UB)$. Now, if ϕ were the set of schedules satisfying all the dependence constraints and all the resource constraints specified, then the problem of determining $\mathsf{ASAP}_{ideal}(\phi, t_i)$ or $\mathsf{ALAP}_{ideal}(\phi, t_i)$ is NP-complete (as this would correspond to solving the RCS problem for $G_{pred}(t_i)$ or $G_{succ}(t_i)$ respectively). Therefore, we have lower bounds when resource constraints are added. However, these lower bounds can be tightened as we show in Section 3.

Since the scheduling problem is NP-complete [20, 1], we try to bound the optimal solution by discarding some of the constraints. Discarding constraints corresponds to increasing the size of the solution space. Minimizing over a larger solution space, gives a lower bound on the optimal solution in the original solution space. In the next two sections, we consider bounds that can be obtained by discarding precedence constraints. The advantage of discarding all the precedence constraints is that this causes the step (time) to which each task is scheduled to be relatively independent of the time to which other tasks are scheduled. Simply discarding all the precedence constraints causes too much of a relaxation of the problem. So we introduce some additional constraints which would certainly be satisfied in any solution to the original problem. The constraints that we introduce simply require that each task be scheduled at or after its ASAP time and at or before its ALAP time. In the next two sections, we prove several results and show how those results can be applied to solve this problem.

3 Bounds on Resource-Constrained Scheduling (RCS)

In this section, we will obtain bounds on the time taken to schedule all the tasks in T, given constraints on the number of resources of each type available. Our strategy for obtaining bounds will be to discard all the precedence constraints, and then calculate the optimal value of the latency of the schedule; this will be a lower bound on the optimal latency of schedules that preserve the precedence constraints.

Before we describe how to compute this bound, we will describe a technique to compute the optimal solution to simpler problem, and then show how this procedure can be applied to solve the problem at hand.

3.1 Scheduling Independent Operations with Release Times and Resource Constraints

In this section, we will consider the problem of scheduling independent tasks with release times. Let $T = \{t_1, t_2, \ldots, t_N\}$ be the set of operations to be scheduled. Each task $t_i \in T$ has a release time $\mathsf{Release}(t_i)$, and so the operation cannot begin executing before time step $\mathsf{Release}(t_i)$. We initially assume that all operations execute on the same type of resource, i.e., there is only one resource class, and that each resource of this class has unit delay. The number of instances of this resource available is M_1. Further, we assume that we are given another function $\theta : T \to \mathcal{N}^+$, and we seek a schedule $\mathcal{S} : T \to \mathcal{N}^+$ such that the value of

$$\mathsf{Demerit}(\mathcal{S}) = \max_{t_i \in T} (\mathcal{S}(t_i) + \theta(t_i))$$

is minimized.

Without loss of generality, we assume that the operations in T have been ordered in decreasing order of $\theta(t_i)$. So,

$$\theta(t_1) \geq \theta(t_2) \geq \theta(t_3) \geq \ldots \geq \theta(t_N)$$

Theorem 1 *The algorithm shown in Figure 1 will always produce a schedule $\mathcal{S}$ such that the value of* $\mathsf{Demerit}(\mathcal{S})$ *is minimum.*

PROOF. Let $\mathcal{S}(t_i)$ be the time step that operation t_i is assigned to by the algorithm in Figure 1.

```
Procedure SimpleBound(T, Release, θ, M_1)
NumberResAvailable[step] ← M_1          ∀step ∈ Z+
Demerit ← 0

for i ← 1 to N do
    step ← Release(t_i)
    while (NumberResAvailable[step] = 0) do
        step ← step + 1
    enddo
    S(t_i) ← step
    NumResAvailable[step] ← NumResAvailable[step] − 1
    Demerit ← max(Demerit, step + θ(t_i))
endif
enddo
return Demerit
enddo
```

Figure 1: Algorithm to minimize Demerit

Define FirstDiffer(A, B) where A and B represent schedules as follows.

$$\text{FirstDiffer}(A,B) = \begin{cases} 1 & \text{if } A(t_1) \neq B(t_1) \\ 1 + \left(\max\limits_{\forall 1 \le j \le i \; A(t_j) = B(t_j)} i \right) & \text{otherwise.} \end{cases}$$

Let $\mathcal{OPT}$ be the set of all schedules $\mathcal{S}$ for which Demerit$(\mathcal{S})$ is minimized. Pick $\mathcal{O} \in \mathcal{OPT}$ such that $\forall \mathcal{O}' \in \mathcal{OPT}$, we have

$$\text{FirstDiffer}(\mathcal{S}, \mathcal{O}) \geq \text{FirstDiffer}(\mathcal{S}, \mathcal{O}')$$

The outline of our proof is as follows. We will argue by contradiction, showing that if $\mathcal{O} \neq \mathcal{S}$ (i.e., FirstDiffer$(\mathcal{O}, \mathcal{S}) < N + 1$) then we can always construct another optimal schedule (in terms of minimizing Demerit) $\mathcal{O}'$ from $\mathcal{O}$ with FirstDiffer$(\mathcal{O}, \mathcal{S}) <$ FirstDiffer$(\mathcal{O}', \mathcal{S})$, thereby contradicting the way $\mathcal{O}$ was chosen.

Let $m =$ FirstDiffer$(\mathcal{S}, \mathcal{O})$. Let $Oper(\mathcal{O}, step)$ be the set of operations assigned to time $step$. So

$$Oper(\mathcal{O}, step) = \{t_j | \mathcal{O}(t_j) = step\}.$$

We must have $\mathcal{S}(t_m) < \mathcal{O}(t_m)$. There are two cases.

Case (a): The number of operations scheduled at the same step as t_m is less than the number of available resources, i.e., $|Oper(\mathcal{O}, \mathcal{S}(t_m))| < M_1$.

Since $\mathcal{S}(t_m) \geq \textsf{Release}(t_m)$, operation t_m can be rescheduled to step $\mathcal{S}(t_m)$. Modify schedule $\mathcal{O}$ by rescheduling operation t_m to step $\mathcal{S}(t_m)$ to obtain schedule $\mathcal{O}'$. Clearly $\textsf{Demerit}(\mathcal{O}') \leq \textsf{Demerit}(\mathcal{O})$. Therefore, $\mathcal{O}' \in \mathcal{OPT}$. Further, $\textsf{FirstDiffer}(\mathcal{S}, \mathcal{O}) < \textsf{FirstDiffer}(\mathcal{S}, \mathcal{O}')$. This violates our assumption and hence, this case is not possible .

Case (b): All units of the resource are allocated to operations that are scheduled to the same step as t_m, i.e., $|Oper(\mathcal{O}, \mathcal{S}(t_m))| = M_1$.

Then, there must be at least one $t_k \in \{t_{m+1}, t_{m+2}, \ldots, t_N\}$ for which $\mathcal{O}(t_k) = \mathcal{S}(t_m)$. Since $\theta(t_m) \geq \theta(t_m + 1) \geq \ldots \geq \theta(t_N)$, we must have $\theta(t_m) \geq \theta(t_k)$. Operation t_k can be scheduled to time step $\mathcal{O}(t_m)$ because

$$\textsf{Release}(t_k) \leq \mathcal{O}(t_k) = \mathcal{S}(t_m) \leq \mathcal{O}(t_m)$$

Further, operation t_m can be scheduled to time step $\mathcal{O}(t_k) = \mathcal{S}(t_m)$. Create a new schedule $\mathcal{O}'$ with

$$\mathcal{O}'(t_i) = \begin{cases} \mathcal{O}(t_i) & \text{if } i \neq m \text{ and } i \neq k \\ \mathcal{O}(t_m) & \text{if } i = k \\ \mathcal{O}(t_k) & \text{if } i = m \end{cases}$$

In other words, $\mathcal{O}'$ is the same as $\mathcal{O}$ with the exception that tasks m and k have their positions swapped. Note that $\mathcal{O}'$ is a correct schedule in that all operations are scheduled only after their $\textsf{Release}$ time and no resource constraints are violated.

$$\mathcal{O}'(t_k) + \theta(t_k) = \mathcal{O}(t_m) + \theta(t_k) \leq \mathcal{O}(t_m) + \theta(t_m)$$

$$\mathcal{O}'(t_m) + \theta(t_m) = \mathcal{O}(t_k) + \theta(t_m) \leq \mathcal{O}(t_m) + \theta(t_m)$$

Therefore, $\textsf{Demerit}(\mathcal{O}') \leq \textsf{Demerit}(\mathcal{O})$ and so $\mathcal{O}'$ is an optimal schedule with $\textsf{FirstDiffer}(\mathcal{S}, \mathcal{O}) < \textsf{FirstDiffer}(\mathcal{S}, \mathcal{O}')$, which is a contradiction.

Therefore the algorithm in Figure 1 always computes the optimal value of the function specified.

□

Lemma 2 *If* $\textsf{Release}_1(t_i) \geq \textsf{Release}_2(t_i)\ \forall t_i \in T$, *then*

$$\textsf{SimpleBound}(T, \textsf{Release}_1, \theta, M_1) \geq \textsf{SimpleBound}(T, \textsf{Release}_2, \theta, M_1)$$

PROOF. $\mathsf{SimpleBound}(T, \mathsf{Release}, \theta, M_1) =$

$$\mathsf{Demerit}(\mathcal{S}) = \min_{\mathcal{S}:\, \forall t_i, \mathcal{S}(t_i) \geq \mathsf{Release}(t_i)} \left(\max_{t_i} (\mathcal{S}(t_i) + \theta(t_i)) \right)$$

So if $\mathsf{Release}_1(t_i) \geq \mathsf{Release}_2(t_i)\ \forall t_i \in T$, then since

$$\{\mathcal{S} :\ \forall t_i, \mathcal{S}(t_i) \geq \mathsf{Release}_1(t_i)\} \subseteq \{\mathcal{S} :\ \forall t_i, \mathcal{S}(t_i) \geq \mathsf{Release}_1(t_i)\}$$

we must have $\mathsf{SimpleBound}(T, \mathsf{Release}_1, \theta, M_1) \geq \mathsf{SimpleBound}(T, \mathsf{Release}_2, \theta, M_1)$ □

Lemma 3 *If $\theta_1(t_i) \geq \theta_2(t_i)\ \forall t_i \in T$, then*

$$\mathsf{SimpleBound}(T, \mathsf{Release}, \theta_1, M_1) \geq \mathsf{SimpleBound}(T, \mathsf{Release}, \theta_2, M_1)$$

PROOF. For any schedule $\mathcal{S}$ and for all $t_i \in T$, we must have

$$\mathcal{S}(t_i) + \theta_1(t_i) \geq \mathcal{S}(t_i) + \theta_2(t_i)$$

Therefore, $\mathsf{SimpleBound}(T, \mathsf{Release}, \theta_1, M_1) \geq \mathsf{SimpleBound}(T, \mathsf{Release}, \theta_2, M_1)$ □

The bounding procedure SimpleBound has been defined only for sets of operations that are all mapped onto the same resource class. Now, suppose the operations in T are mapped to R different resource classes. In this case, we partition T into R subsets $\{T_1, T_2, \ldots, T_R\}$ such that $T = T_1 \cup T_2 \cup \ldots \cup T_R$ and each partition T_j consists of all tasks mapped to resource class j, i.e., $\mathsf{Resource}(t_i) = j\ \ \forall t_i \in T_j$. Given any function f, we will denote by $f|_{T_j}$ the restriction of f to the set T_j. We define the bound as

$$\mathsf{SimpleBound}(T, \mathsf{Release}, \theta) \quad = \quad \max_{1 \leq j \leq R} \mathsf{SimpleBound}(T_j, \mathsf{Release}|_{T_j}, \theta|_{T_j}, M_r)$$

3.2 Application to the original problem

Since we are considering bounds that can be obtained by relaxing precedence constraints, it is useful to (for this section alone) extend our definition of schedules. We will refer to all functions $\mathcal{S} : T \to \mathcal{N}^+$ that satisfy the resource constraints, i.e.,

$$\left(\sum_{t_i \in T, type(t_i) = r, \mathcal{S}(t_i) = s} 1 \right) \leq M_r \qquad \forall s \in \mathcal{N}^+$$

as schedules. We will refer to schedules that do not violate precedence constraints as valid schedules. We will also extend the definition of schedules to subsets of the set of tasks T. We will refer to $\mathcal{S}'$

as a valid schedule of $T' \subseteq T$ if it satisfies all precedences E' induced by T'. For any $T' \subseteq T$, we define

$$Opt(T') = \min_{\mathcal{S} \text{ a valid schedule for } T'} \left(\max_{t_i \in T'} \mathcal{S}(t_i) \right).$$

Let us assume that UB is an upper bound on the latency of the optimal valid schedule (i.e., the schedule does not violate any precedence constraints). Any valid schedule is an upper bound on the latency of an optimal valid schedule, and so this value can be easily obtained as the latency of a heuristic solution.

We will derive results for two functions $\mathcal{H}_1$ and $\mathcal{H}_2$. Suppose we are given functions $\mathcal{H}_1$ and $\mathcal{H}_2$ which satisfy the following properties.

- $UB - Opt(G_{succ}(t_i)) \leq \mathcal{H}_2(t_i) \leq UB$
- $0 \leq \mathcal{H}_1(t_i) \leq Opt(G_{pred}(t_i))$

Note that the ASAP and the ALAP functions as we defined before satisfy these conditions for $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. Also notice that $\mathcal{H}_1$ and $\mathcal{H}_2$ need not be schedules as they need not satisfy the resource constraints. The function $\mathcal{H}_1$ represents an upper bound on ASAP_{ideal}, while $\mathcal{H}_2$ is a lower bound on ALAP_{ideal}.

Let $\Lambda(\mathcal{S}) = \max_{t_i \in T}(\mathcal{S}(t_i) + UB - \mathcal{H}_2(t_i))$ and $\lambda(\mathcal{S}) = \max_{t_i \in T} \mathcal{S}(t_i)$.

Theorem 4 *For any valid schedule $\mathcal{S}$, $\lambda(\mathcal{S}) = \Lambda(\mathcal{S})$.*

Proof. It is clear that for any schedule $\mathcal{S}$ of T, $\lambda(\mathcal{S}) \leq \Lambda(\mathcal{S})$. So we need only show the reverse inequality.

For any $t_i \in T$

$$\begin{array}{rcccc} & UB - Opt(G_{succ}(t_i)) & \leq & \mathcal{H}_2(t_i) & \leq & UB \\ \Rightarrow & Opt(G_{succ}(t_i)) - UB & \geq & -\mathcal{H}_2(t_i) & \geq & -UB \\ \Rightarrow & Opt(G_{succ}(t_i)) & \geq & UB - \mathcal{H}_2(t_i) & \geq & 0 \end{array}$$

Therefore, if t_i has no successors $\mathcal{H}_2(t_i) = UB$. Therefore, for all $t_i \in T$ with no successors, $\mathcal{S}(t_i) = \mathcal{S}(t_i) + UB - \mathcal{H}_2(t_i)$ for all schedules $\mathcal{S}$ of T.

Now, Suppose t_i has at least one successor.

$$\begin{array}{rcccc} & Opt(G_{succ}(t_i)) & \geq & UB - \mathcal{H}_2(t_i) & \geq & 0 \\ \Rightarrow & \mathcal{S}(t_i) + Opt(G_{succ}(t_i)) & \geq & \mathcal{S}(t_i) + UB - \mathcal{H}_2(t_i) & \geq & \mathcal{S}(t_i) \end{array}$$

Procedure $\mathsf{LowerBound}(T,\, \mathcal{H}_1,\, \mathcal{H}_2,\, UB)$
for $i \leftarrow 1$ **to** $|T|$ **do**
  $\mathsf{Release}(t_i) \leftarrow \mathcal{H}_1(t_i)$
  $\theta(t_i) \leftarrow UB - \mathcal{H}_2(t_i)$
enddo
return $\mathsf{SimpleBound}(T,\, \mathsf{Release},\, \theta)$

Figure 2: Lower Bounding Routine

In no schedule $\mathcal{S}$ of T can $\mathcal{S}(t_i) - \mathcal{S}(t_j) < Opt(G_{succ}(t_i)) \quad \forall t_j \in G_{succ}(t_i)$. So there must exist some t_j with no successors such that $\mathcal{S}(t_j) \geq \mathcal{S}(t_i) + Opt(G_{succ}(t_i))$. Let $\mathsf{NoSucc}(T) = \{t_i \in T : t_i \text{ has no successor }\}$. Therefore,

$$\begin{aligned} \lambda(\mathcal{S}) &= \max_{t_i \in \mathsf{NoSucc(T)}} (\mathcal{S}(t_i)) \\ &= \max_{t_i \in \mathsf{NoSucc(T)}} (\mathcal{S}(t_i) + Opt(G_{succ}(t_i))) \\ &\geq \max_{t_i \in T - \mathsf{NoSucc(T)}} (\mathcal{S}(t_i) + Opt(G_{succ}(t_i))) \end{aligned}$$

Therefore, for any schedule $\mathcal{S}$, $\Lambda(\mathcal{S}) \leq \lambda(\mathcal{S})$.

Therefore $\lambda(\mathcal{S}) = \Lambda(\mathcal{S})$. □

As a result, it does not really matter whether we optimize for $\lambda(\mathcal{S})$ or for $\Lambda(\mathcal{S})$ when computing exact solutions. However, the choice of the optimizing function does make a difference when we consider a relaxation where all the precedences are discarded. It can be seen that the inequality $\Lambda(\mathcal{S}) \geq \lambda(\mathcal{S})$ holds irrespective of the validity (in terms of precedence constraints) of $\mathcal{S}$. However, the two need not be equal if any precedences are violated in $\mathcal{S}$ as the $UB - \mathcal{H}_2(t_i)$ term is effectively a penalty associated with precedence violations. Therefore, when we relax all precedence constraints, the optimal value of Λ is a tighter bound than the optimal value of λ. Solving for an optimal value of Λ can be computed by calling the Lower-Bound routine with $\mathcal{H}_1$ as the $\mathsf{Release}$ function and $UB - \mathcal{H}_2$ as the θ function. The complete algorithm for calculating a lower bound on the latency of the optimal valid schedule given the functions $\mathcal{H}_1$ and $\mathcal{H}_2$ is shown in Figure 2. Table 1 shows the results of solving for the optimal value of both λ and Λ for various benchmarks. It can be seen that solving for optimal Λ produces tighter bounds in most of the cases.

3.3 Obtaining $\mathcal{H}_1$ and $\mathcal{H}_2$

As stated in Lemmas 3.1 and 3.2, the tightest bounds on the latency of optimal schedules of T

Table 1: Lower bounds on latency

Benchmark	$*$	ALU	Optimal Latency	LB with λ	LB with Λ	LB with LP
EWF	1	1	28	27	28	23
EWF	2	1	28	27	28	23
EWF	1	2	21	20	21	19
EWF	2	2	18	17	18	18
FDCT	1	1	34	33	34	21
FDCT	2	2	18	17	18	13
FDCT	2	3	18	17	18	12
FDCT	3	3	14	12	13	10
ARF	1	1	34	32	34	29
ARF	1	2	34	32	34	29
ARF	2	1	18	16	18	16
ARF	2	2	18	16	18	16
HAL	1	1	13	12	13	12
HAL	2	1	8	6	7	8
HAL	3	2	6	6	6	6

$\mathcal{H}_1 \leftarrow \mathsf{ASAP}_G$

for $i \leftarrow 1$ **to** $|T|$ **do**
  $t_i \leftarrow i^{th}$ operation in topologically sorted order
  $\mathcal{H}_1(t_i) \leftarrow \mathsf{LowerBound}\left(G_{pred}(t_i), \mathcal{H}_1|_{G_{pred}(t_i)}, \mathsf{ALAP}|_{G_{pred}(t_i)}, UB\right)$
enddo

Figure 3: Code fragment to calculate $\mathcal{H}_1$ (and $\mathcal{H}_2$)

can be obtained when the tightest values of the $\mathcal{H}_1$ and $\mathcal{H}_2$ functions that satisfy the requirements are used. The values of $\mathcal{H}_1(t_i)$ and $\mathcal{H}_2(t_i)$ are actually lower and upper bounds on the value of $\mathsf{ASAP}_{ideal}(\phi, t_i)$ and $\mathsf{ALAP}_{ideal}(\phi, t_i)$ respectively, where ϕ is the set of all schedules that satisfy all the precedence and all the resource constraints and $\mathsf{Latency}(\mathcal{S}) \leq UB$.

Define $T(t_i, t_j) = T_{succ}(t_i) \cap T_{pred}(t_j)$ and a family of functions $\mathcal{H}_{2_j} : T_{pred}(t_j) \rightarrow \mathcal{N}^+$ where $\mathcal{H}_{2_j}(t_i) = \mathsf{CP}(T(t_i, t_j))$.

Although we do not show it, it can be shown that tighter results can be obtained by defining $\mathcal{H}_{2_j}$ as

$$\mathcal{H}_{2_j}(t_i) = \mathsf{LowerBound}(T(t_i, t_j), \mathsf{ASAP} - \mathsf{ASAP}(t_i), \mathsf{ALAP}(t_j, UB) - \mathsf{ALAP}(UB), UB)$$

Now, we use the fact that operation t_i cannot be scheduled until all the operations in $G_{pred}(t_i)$ have been completed. It can be shown that $\mathsf{ASAP}_{ideal}(\phi, t_i) \geq \mathsf{LowerBound}(T_{pred}(t_i), \mathsf{ASAP}, \mathcal{H}_{2_i}, UB) \geq ASAP(t_i)$. So we use the routine shown in Figure 3 to compute $\mathcal{H}_1$, which is a better bound for ASAP_{ideal} than is $ASAP$.

To compute the value of $\mathcal{H}_2$ we make use of the following Lemma.

Lemma 5 *A lower bound on the* ASAP *function for a graph G is also an upper bound on the* ALAP *function for the graph G^r obtained from G by reversing all the edge directions (dependencies) and vice-versa.*

PROOF. See [14] for a proof. □

Theorem 6 *Let $S \in \phi$ be a valid schedule for the DFG $G = (V, E)$. Let $G^r = (V, E^r)$ be another DFG, where $\langle x_i, x_j \rangle \in E^r \Leftrightarrow \langle x_j, x_i \rangle \in E$. Then, $S^r : V \rightarrow Z$ where $S^r(x_j) = Latency(S) - S(x_j) + 1$ is a valid schedule for G^r and $Latency(S) \geq Latency(S^r)$.*

PROOF: In order to prove the validity of S^r, we must prove the following

- All dependencies are satisfied;
- All resource constraints are satisfied.

Consider the dependency $\langle x_j, x_i \rangle \in E^r$. By construction, $\exists \langle x_i, x_j \rangle \in E$. Since S is a valid schedule, all dependencies must have been satisfied in S. Therefore, we have $S(x_j) > S(x_i)$. So $S^r(x_i) - S^r(x_j) = Latency(S) + 1 - S(x_i) - Latency(S) - 1 + S(x_j) = S(x_j) - S(x_i) > 0$. Therefore, the dependency $\langle x_j, x_i \rangle \in E^r$ is satisfied in S^r. To show that all resource constraints are satisfied in S^r, we notice that $S(x_j) = S(x_i) \Leftrightarrow S^r(x_j) = S^r(x_i)$. If we denote by $R(step, class, S)$ the set $\{x_j \in V | S(x_j) = step \bigcap type(x_j) = class\}$, then $R(step, class, S) = R(Latency(S) - step + 1, class, S^r)$. Therefore all resource constraints are satisfied in S^r.

Finally, $S^r(x) = Latency(S) + 1 - S(x) \leq Latency(S)$ because $S(x) \geq 1$.

Corollary 7 *If $S : V \rightarrow Z$ is an optimal schedule for $G = (V, E)$, then $S^r : V \rightarrow Z$ is an optimal schedule for $G^r = (V, E^r)$.*

A key point to notice is that the ASAP values of the operations of the DFG G become the ALAP values of the corresponding operations of the DFG G^r. This is because, the set of predecessors of an operation become the set of successors of the corresponding operation in the reversed DFG. So finding the ASAP values of G (resp. G^R) is the same as finding the ALAP values of G^R (resp. G).

Corollary 8 *If $S(x_j) \geq ASAP(x_j)$ $\forall x_j \in V(G)$, S a valid schedule, then $S^r(x_j) \leq Latency(S) - ASAP(x_j) + 1$.*

So we can use the same routine to compute the values of $\mathcal{H}_2$.

Lemma 9 *For any task graph $G = (T, E)$ and any $\mathcal{H}_1 \geq \mathsf{ASAP}$, the value of* $\mathsf{LowerBound}(T, \mathcal{H}_1, \mathcal{H}_2) \geq \mathsf{CP}(G)$.

Proof. This result can be seen immediately from the fact that if $\mathcal{H}_1 \geq \mathsf{ASAP}$, then there is at least one operation t_i whose Release Time ($\mathcal{H}_1$) is equal to the Critical Path Length of T. □

Lemma 10 *The code fragment in Figure 3 calculates values of $\mathcal{H}_1$ and $\mathcal{H}_2$ which satisfy the required properties, and which are tighter than the* ASAP *and the* ALAP *functions.*

Proof. From the code in Figure 3, it can be seen that $\mathcal{H}_1(t_i)$ is calculated only after $\mathcal{H}_1(t_j)$ has been calculated for all predecessors t_j of t_i. If t_i, has no predecessors, then the statement $\mathcal{H}_1(t_j) \geq \mathsf{ASAP}(t_j) \quad \forall t_j \in G_{pred}(t_i)$ holds vacuously. Therefore, the value calculated for $\mathcal{H}_1(t_i) \geq \mathsf{ASAP}(t_i)$ by Lemma 3.4. Now, suppose that for some t_i (with predecessors), $\mathcal{H}_1(t_j) \geq \mathsf{ASAP}(t_j) \quad \forall t_j \in G_{pred}(t_i)$ holds. Then again by Lemma 3.4, we have $\mathcal{H}_1(t_i) \geq \mathsf{ASAP}(t_i)$. Hence, by induction, $\mathcal{H}_1(t_i) \geq \mathsf{ASAP}(t_i) \quad \forall t_i \in T$.

The proof that $\mathcal{H}_2(t_i) \leq \mathsf{ALAP}(t_i) \quad \forall t_i \in T$ is similar to the one above. □

The bounds calculation is done in topological order so that any improvement in the value of either of these functions will be propagated to their successors or predecessors.

3.4 Discussion

It is easy to see that the bounds derived by Rim and Jain [18] are the same as the bounds obtained from $\mathsf{LowerBound}(T, \mathsf{ASAP}, UB - \mathsf{ALAP}, UB)$, and that the bounds computed by Langevin and Cerny [13] is the same as the bounds obtained from $\mathsf{LowerBound}(T, \mathcal{H}_1, UB - \mathsf{ALAP})$. Therefore, the bounds obtained from the procedure shown above are tighter than the bounds obtained by Langevin and Cerny [13] which in turn are tighter than those obtained by Rim and Jain [18].

The algorithm shown has $\mathrm{O}(N)$ time complexity, once $\mathcal{H}_1$ and $\mathcal{H}_2$ have been calculated; the functions $\mathcal{H}_1$ and $\mathcal{H}_2$ can be calculated in $\mathrm{O}(N^2)$ time, and so it is advantageous to use our algorithm both for initial bound estimation and as a part of a branch-and-bound scheme.

We also compared the bounds that we get from our technique and those in [18] and [13] with bounds from LP relaxation of an ILP formulation of the RCS problem; our results demonstrate that LP relaxation for RCS produces significantly weaker bounds for many of the cases.

4 Bounds on Time-Constrained Scheduling (TCS)

In this section, we consider the time-constrained scheduling problem. As before, we will consider the bounds that can be obtained by relaxing all the precedence constraints. We will show how the optimal solution can be calculated in the absence of resource constraints using an LP formulation. Then, we show that these bounds are no worse than bounds due to [4] and [19] and therefore that the LP relaxation of an ILP formulation of the TCS problem produces tighter bounds than bounds due to [4, 19].

4.1 Scheduling Independent Operations with Release Times and Deadlines (SIRD)

In this section we consider the following problem: Given a set of independent tasks T, with each task $t_i \in T$ having a release step $\mathsf{Release}(t_i)$ and a deadline $\mathsf{Deadline}(t_i)$, find the minimum number of resources that is required for executing all the tasks within the ranges specified. As before, we initially start by restricting all the operations to execute on the same resource class.

This problem can be modeled and solved using Integer Linear Programming. In this section, we will show that the polyhedra corresponding to this problem is (almost) integral, and that the ceiling of the optimal solution to the linear relaxation of this problem is the optimal solution to this problem; therefore, it is sufficient to solve the linear relaxation problem to find optimal integral solutions to the original problem.

4.2 ILP formulation for this (SIRD) problem

We use a linear programming formulation similar to the one used by Chaudhuri and Walker [3] and Gebotys and Elmasry [5]; this is a restriction of their formulation.

Each task t_i has to be scheduled at some step between $\mathsf{Release}(t_i)$ and $\mathsf{Deadline}(t_i)$, and we have to decide which step to schedule the task in. This is modeled by a set of decision variables $\{x_{i,n_1}, x_{i,n_1+1}, \ldots, x_{i,n_2}\}$, where $n_1 = \mathsf{Release}(t_i)$ and $n_2 = \mathsf{Deadline}(t_i)$. Since the task should only be executed on one of these steps, we have

$$\sum_{\mathsf{Release}(t_i) \leq j \leq \mathsf{Deadline}(t_i)} x_{i,j} \quad = \quad 1 \tag{1}$$

This is equivalent to the two constraints

$$\sum_{\mathsf{Release}(t_i) \leq j \leq \mathsf{Deadline}(t_i)} -x_{i,j} \quad \leq \quad -1 \tag{2}$$

and

$$\sum_{\mathsf{Release}(t_i) \leq j \leq \mathsf{Deadline}(t_i)} x_{i,j} \quad \leq \quad 1 \tag{3}$$

The number of resources required is the maximum of the number of resources used at any of the steps. If R is the number of resources required, then, we require that the number of resources used at each step j does not exceed R:

$$\forall \text{steps } j \quad \sum_{\{i \mid \mathsf{Release}(t_i) \leq j \leq \mathsf{Deadline}(t_i)\}} x_{i,j} \quad \leq \quad R. \tag{4}$$

Since the number of resources is a variable whose value has to be determined, we will express Inequality 4 as

$$\left(\sum_{\{i \mid \mathsf{Release}(t_i) \leq j \leq \mathsf{Deadline}(t_i)\}} x_{i,j} \right) - R \quad \leq \quad 0 \tag{5}$$

Of course, we require that all the decision variables be either 0 or 1; also, we require that the value of R be integral.

We can represent the conditions, i.e., Inequalities 2, 3 and 5, as a matrix inequality of the form $A'\vec{x} \leq \vec{b}$ where A' is the coefficient matrix and $\vec{x}$ is the column vector corresponding to the decision variables and R. The first $|T|$ rows of A' represent the coefficients of Inequalities 2, and the next $|T|$ rows represent the coefficients of Inequalities 3. We will let $N = 2 \cdot |T|$. The remaining *Latency* rows represent the coefficients of Inequalities 5 for the various steps. Let the number of variables (=the number of decision variables+1) be M. Therefore A' is a $(N + \mathit{Latency}) \times M$ matrix and $\vec{x}$ a $M \times 1$ matrix(vector). Without loss of generality, we will assume that the first $M - 1$ elements of vector $\vec{x}$ represent the decision variables, and the last element represents R. Then

$$A' = \left(\begin{array}{c:c} A_1 & \vec{0} \\ A_2 & -\vec{1} \end{array} \right) = \left(\begin{array}{c:c} A & B \end{array} \right)$$

where A_1 is an $N \times (M-1)$ matrix and A_2 is a $\mathit{Latency} \times (M-1)$ matrix and A is the $(N + \mathit{Latency}) \times (M-1)$ matrix obtained by deleting the last (i.e., Mth) column from A. Let $\vec{b_1}$ be the column vector of $|T|$ 1's. Let $\vec{b_2}$ the column vector of *Latency* 0's. We denote by $\vec{b}$ the column vector

$$\begin{pmatrix} -\vec{b_1} \\ \vec{b_1} \\ \vec{b_2} \end{pmatrix}$$

We denote by X the set of decision variables.

We will now prove that A is a Totally Unimodular (TU) matrix. To do so, we use Lemma 2.7 (Section III.1) in [15], which we state below.

Lemma 11 *The following statements are equivalent*

1. *A is totally unimodular.*
2. *For every $J \subseteq \{1, 2, \ldots, N + Latency\}$, there exists a partition J_1, J_2 of J such that*

$$\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| \leq 1 \qquad \forall i \in \{1, \ldots M\}$$

where a_{ij} represents the element in row j and column i.

Lemma 12 *The matrix A is totally unimodular.*

Proof. In this proof, we will assume that the first $N/2 = |T|$ rows of A are obtained from Inequality 2, with the elements of row i being the coefficients of Inequality 2 for task t_i. Similarly the elements of row $N/2 + i$ are the coefficients of Inequality 2 for task t_i.

Each column of A will have exactly 3 non zero entries. There will be a -1 in one of the rows $\{1, \ldots, N/2\}$, a 1 in one of the rows $\{N/2 + 1, \ldots, N\}$ and a 1 in one of the remaining rows. This is because each decision variable x_{ij} appears once in an inequality of the form of Inequality 2, once in an inequality of the form Inequality 3 and once in an inequality of the form Inequality 5, and its coefficients are as stated above.

Let J be some arbitrary set of rows. Now, let $J_0 = J \cap \{1, \ldots, N/2\}$, $J_1 = J \cap \{N/2+1, \ldots, N\}$ and $J_2 = J \cap \{N + 1, \ldots, N + Latency\}$. Notice that each column in the rows of J_1 and J_2 have at most one non-zero element (which must be a 1) in any column. For every $i \in J_0$, if $i + N/2 \in J_1$, then add i to J_1, else add it to J_2. Now, $J_1 \cup J_2 = J$.

Clearly $a_{ij} \in \{-1, 0, 1\} \quad \forall j \in J_2$. Suppose that for some $j \in J_2$, $a_{i_1 j} = -1$. Then $a_{i_1 j} = 0$ for all $j \in J_1$. If $a_{i_1 j} \neq -1 \quad \forall j \in J_2$, then $\left| \sum_{j \in J_2} a_{i_1 j} - \sum_{j \in J_1} a_{i_2 j} \right| \leq 1$. Therefore, we have,

$$\left| \sum_{j \in J_2} a_{ij} - \sum_{j \in J_1} a_{ij} \right| \leq 1 \qquad \forall i \in \{1, \ldots M\}.$$

□

Although we have shown that A is TU, A' need not be TU. Next, we show that the optimal solution to the SIRD problem can be obtained by taking the ceiling of the value returned by the LP relaxation of the ILP formulation.

4.3 Obtaining an optimal solution to the SIRD problem

The problem of finding the optimal (minimum) number of resources is equivalent to solving the following ILP model:

$$\text{(I)} \quad \text{Minimize} \quad R \quad \text{Subject to} \quad \left(A \;\vdots\; B \right) \vec{x} \leq \vec{b} \quad R, x \text{ integral } \forall x \in X \ .$$

We denote this problem as (I). The linear relaxation of this problem (LR) is then

$$\text{(LR)} \quad \text{Minimize} \quad R \quad \text{Subject to} \quad \left(A \;\vdots\; B \right) \vec{x} \leq \vec{b}$$

Lemma 13 *Let r be the optimal solution to (LR). Then $\lceil r \rceil$ is the optimal solution to (I).*

Proof. Since the solution to (I) must be integral, we know that $\lceil r \rceil$ is a lower bound on the solution to (I). Let b_3 be the vector of *Latency* $\lceil r \rceil$'s, and $\vec{b_r}$ be the vector $\begin{pmatrix} -\vec{b_1} \\ \vec{b_1} \\ \vec{b_3} \end{pmatrix}$. Then $\lceil r \rceil$ is the solution to (I) if and only if the space

$$A\vec{x} \leq \vec{b_r}$$

has at least one integral solution in it. Since the matrix A is Totally Unimodular, $\lceil r \rceil$ is the solution to (I) iff the space has any feasible (possibly non-integral) solution. It is clear that this space must have a feasible solution as the solution returned by (LR) is certainly a feasible solution for this space. Hence $\lceil r \rceil$ is the solution to (I). □

Hence the values returned by the LP solution to this problems must be no less than the values returned by those bounding techniques. Consider the case when the precedence constraints are added. Then the optimal solution must certainly be larger, because the space has reduced.

Let P denote matrix such that $P\vec{x} \leq \vec{q}$ represents the set of precedence constraints (see [2, 5]). We define the following set of bounds:

$$(\mathbf{FB}) \qquad LB_0 \quad = \quad \text{Bounds using [4, 16, 17, 19]}$$

Table 2: Comparisons of bounds for the TCS problem

Benchmark	#steps	F-B Lower Bounds		LP Lower Bounds		Upper Bounds	
		MULT	ALU	*MULT*	ALU	*MULT*	ALU
FDCT	9	6	3	8	3	8	3
FDCT	13	3	2	4	2	4	2
ARF	13	3	2	4	2	4	2
DCT	8	8	4	8	5	8	5
DCT	9	7	3	8	4	8	4

$$\begin{array}{rrcll} & LB_1 & = & \left\lceil \min_{A\vec{x}\le\vec{c}} R \right\rceil & \\ & LB_2 & = & \min_{A\vec{x}\le\vec{c},\vec{x}\text{ integral}} R & \\ (\mathbf{LPR}) & LB_3 & = & \left\lceil \min_{A\vec{x}\le\vec{c},P\vec{x}\le\vec{q}} R \right\rceil & \text{used in [5]} \\ (\mathbf{OPT}) & Solution & = & \min_{A\vec{x}\le\vec{c},P\vec{x}\le\vec{q},\vec{x}\text{ integral}} R & \end{array}$$

It is clear that $LB_1 \le LB_2 \le Solution$ and $LB_1 \le LB_3$. We have shown that $LB_1 = LB_2$. It can be shown that the $LB_0 \le LB_2$. Hence, we must have $LB_0 \le (LB_1 = LB_2) \le LB_3 \le Solution$. Hence we have shown that the ceiling of the solution to the linear relaxation of the ILP formulation is tighter than the bounds obtained using [4, 19].

4.4 Discussion

Chaudhuri and Walker [3] show that their technique generates tighter bounds than those produced by techniques in [4, 16, 17, 19]. They also show that the techniques in [4, 16, 17, 19] generate the same bounds; we will refer to these collectively as Fernandez-Bussell techniques. In this section we showed that LP relaxation is tighter than Fernandez-Bussell [4] and Sharma-Jain [19]; our approach was to cast the TCS bounding problem (SIRD) solved by [4, 19] as an LP instance and then show that the ceiling of the optimal solution to this LP instance is indeed optimal for the TCS bounding problem (SIRD). Both Walker and Chaudhuri [3] and Gebotys and Elmasry [5] have observed that LP relaxation found the exact bounds in all of their benchmark examples. Unlike the bounding techniques in [3, 4, 19] based on precedence relaxation, which solve the bounding problem per resource class, LP relaxation can also be used for lower bound estimates on area of resources. In addition, LP relaxation can generate useful information needed for branch node selection in a branch-and-bound technique, unlike other techniques. Thus, LP relaxation may be advantageous for both initial bound estimation as well as in a brach-and-bound technique. Experimental results are shown in Table 2. It is seen that in most of the cases, the bounds returned by LP relaxation are the same as those returned by the Fernandez-Bussell techniques. However, there are cases in which LP produces strictly tighter bounds.

In addition, given the tightness of the bounds produced by LP relaxation of the TCS problem—and the noticeable weakness of LP relaxation as a bounding technique for RCS—it may be better to use the TCS problem as the core problem in an exact design space exploration strategy using ILP scheduling in high-level synthesis.

5 Summary

This paper has presented several new results on lower bounds for the scheduling problem in high-level synthesis. For the resource-constrained scheduling problem, we presented a new way of characterizing lower-bounding, which results in a new bounding technique that is shown to be guaranteed to be no worse than the bounds using the Langevin-Cerny [13] approach, which is turn are proved to be no worse than Rim-Jain [18] bounds. For the time-constrained scheduling problem, we showed that by throwing away all precedence constraints and using ASAP and ALAP values, one could derive an LP formulation whose optimal solution is guaranteed to be integral. The solution to this LP problem is shown to be a lower bound for TCS and is guaranteed to be no worse than Fernandez-Bussell [4] or Sharma-Jain [19] bounds.

References

[1] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[2] Samit Chaudhuri and Robert A. Walker. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. *IEEE Transactions on Very Large Scale Integration*, 2(4):456–471, Dec. 1994.

[3] Samit Chaudhuri and Robert A. Walker. Computing bounds on functional units before scheduling. *IEEE Transactions on Very Large Scale Integration*, 4(2):273–279, June 1996.

[4] E. B. Fernańdez and B. Bussell. Bounds on the number of processors and time for multiprocessor optimal schedule. *IEEE Transactions on Computers*, C-22:745–751, Aug 1973.

[5] Catherine H. Gebotys and Mohamed I. Elmasry. Global optimization approach for architectural synthesis. *IEEE Trans. Computer-Aided Design*, 12(9):1266–1278, September 1993.

[6] Y. Hu and B.S. Carlson. Improved lower bounds for the scheduling optimization problem. In *Proceedings of the 1994 IEEE International Symposium on Circuits and Systems*, pages 295–298, London, England, June 1994.

[7] Y. Hu and B.S. Carlson. A unified algorithm for the estimation and scheduling of data flow graphs. *Journal of Circuits, Systems and Computers*, 6(3):287–318, 1996.

[8] Y. Hu, A. Ghouse, and B.S. Carlson. Lower bounds on the iteration time and the number of resources for functional pipelined data flow graphs. In *Proceedings of the IEEE International Conference on Computer Design*, pages 21–24, Cambridge, MA, Oct. 1993. IEEE Computer Press.

[9] C.T. Hwang, Y.C. Hsu, and Y.L. Lin. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. Computer-Aided Design*, 10(4):464–475, April 1991.

[10] R. Jain, M.J. Mlinar, and A. Parker. Area-time model for synthesis of non-pipelined designs. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 48–51, Santa Clara, November 1988. IEEE Computer Society Press.

[11] R. Jain, A. Parker, and N. Park. Predicting area-time tradeoffs for pipelined design. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 35–41, Miami Beach, June 1987. ACM and IEEE Computer Society.

[12] Rajiv Jain, Alice Parker, and N. Park. Predicting area-time tradeoffs for pipelined design. *IEEE Trans. Computer-Aided Design*, 11:955–965, August 1992.

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

[14] M. Narasimhan and J. Ramanujam. A fast approach to computing exact solutions to the resource-constrained scheduling problem, Jul. 1997. Submitted for publication.

[15] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley–Interscience, 1988.

[16] S. Y. Ohm, F. H. Kurdahi, and N. Dutt. Comprehensive lower bound estimation from behavioral descriptions. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, San Jose, Calif., 1994. IEEE Computer Society.

[17] J. M. Rabaey and M. Potkonjak. Estimating implementation bounds for real time DSP application specific circuits. *IEEE Trans. Computer-Aided Design*, 13:669–683, June 1994.

[18] Minjoong Rim and Rajiv Jain. Lower bound performance estimation for the high-level synthesis scheduling problem. *IEEE Trans. Computer-Aided Design*, 13(4):451–458, April 1994.

[19] A. Sharma and Rajiv Jain. Estimating architectural resources and performance for high-level synthesis applications. *IEEE Transactions on Very Large Scale Integration*, 1(2):175–190, June 1993.

[20] J.D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, June 1975.