High-Level Optimization in a Silicon Compiler

Krishna Palem[1],Donald.S.Fussell[2]
and Ashley.J.Welch[1]

TR-215 November 1982

[1]Department of Electrical Engineering
The University of Texas at Austin
Austin, Tx 78712

[2]Department of Computer Sciences
The University of Texas at Austin
Austin, Tx 78712

Abstract

Increasing system design costs are impeding the rapid and widespread application of VLSI technology. As a result, significant efforts have been directed in recent years towards the development of automatic design tools. A paradigm of silicon compilation is described in this paper. This paradigm differs from earlier methods in that the optimization principles are applied initially on a high-level abstraction of the final layout. The optimization problem is shown to be initially equivalent to that of determining a precedence-constrained schedule which is unfortunately known to be NP-complete in the general case. Linear time algorithms for determining optimal schedules for problem classes of interest are described and are shown to be correct. The final goal of this work is the production of the high-level optimization phase of a silicon compiler which translates a functional description of a system into an intermediate architecture for a chip in a faithful and efficient manner. The mask for final layout of the chip can then be generated from this intermediate architecture using known design automation techniques.

Key words: Graphs, Highly Parallel Architectures, Scheduling, Silicon Compilers, Time Complexity, VLSI Technology.

1 Introduction

Recent advances in the area of computer architecture and process technology have altered the system design paradigm significantly. As a result, the concept of a system centered around a single processing unit is being superseded by reconfigurable multiprocessors[1] and highly parallel VLSI architectures[2]. In particular, the technological developments will allow the cost effective realization of systems consisting of tens of thousands of computing elements[3,4]. However, rapidly escalating design costs are becoming a primary limitation on the applications of VLSI technology. As a result, much effort has been devoted in recent years to the development of automated design tools. Various paradigms for silicon compilation have been proposed, and systems for automatic synthesis of layouts for limited domains of functional specifications have been created[5,6].

The design of a complex VLSI circuit can be viewed as a hierarchical refinement process. An abstract and succint algorithm specification represents the highest level of this hierarchy. This description is gradually transformed into a physical layout to be embedded in silicon (Figure 1). The initial phase of this process transforms this algorthmic specification into some sort of a functional level description of the design. Correctness of the resulting description can be verified at this level through simulation. This design is subsequently refined to yield an intermediate gate and switch level description of the architecture. Both these phases are currently performed manually with the aid of interactive graphics tools. Existing systems have the ability to automatically generate layout descriptions given the intermediate architecture. Design rule violations in the final layout can also be checked automatically.

The efficiency of a VLSI circuit from performance (speed) and cost viewpoints is dependent on the layout geometry to a great extent. As a consequence, minimizing the area of the final layout is central to the process of designing optimal VLSI architectures. If this goal is to be achieved, the designer must strive to generate intermediate architectures amenable to further geometric optimization. Due to the fine granularity of the specification of the circuit at the gate level, and the resultant large number of elements involved,the task of manually generating intermediate architectures is very time consuming.

This problem can be ameliorated by attacking the optimization problem earlier, when the circuit descriptions are at a more abstract level and hence fewer elements are involved. In such a case, layout optimization should only be required within high level elements , the inter-element optimization having already been done. We thus break a large and difficult problem into a number of smaller, easier problems, with a consequent performance gain for the overall system (Figure 2).

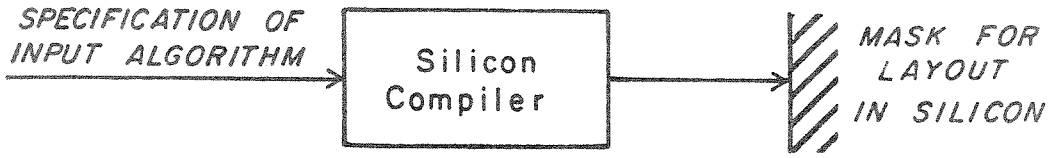


Fig. 1. The process of silicon compilation

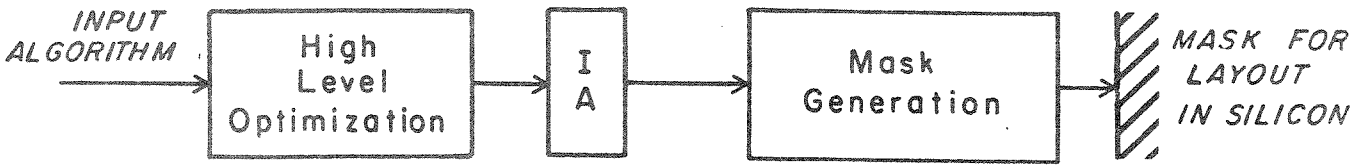


Fig. 2. The high level optimization stage

Based on this approach, a new paradigm of silicon compilation is proposed below. This proposal differs from previous efforts in that the optimization process of the implementation is not confined to the final layout stage, but is also performed early in the compilation process on a more abstract representation of the problem. The high level optimization phase is composed of data flow analysis and data path extraction and optimization. The data flow analysis problem involves determining an optimal schedule for completing the algorithm specified as an input to the compiler. The schedule proceeds by assigning resources with the goal of completing the various steps of the computation as specified by the input algorithm in a faithful manner.The objective of the optimization step is to determine the minimum number of resources required by the schedule. The schedule and resource estimates of the data flow analysis phase serve as an input to the data path extraction and optimization phase. The primary goal of this phase involves binding the data paths in an optimum fashion. It has to be noted that optimality requires that data paths be bound in such a fashion that efficient(in an area sense) layout techniques are known for the resulting topologies.

The input, as specified to the silicon compiler, is initially transformed into an equivalent program graph representation. This model is described in the next section. In addition, algorithms are described for determining tight upper- and lower-bounds on the resource requirements and corresponding schedules for classes of program graphs of interest. Also, these algorithms are shown to be correct and efficient. The high-level optimization phase is formularized next. These results will be instrumental in bridging the gap between the high-level input specifications and the conventional design automation phase of silicon compilation by providing automatic tools whose complexity and correctness issues are well understood.

2 The High Level Optimization Phase

2.1 Input Specification and Representation

While the input specification of the algorithm to be compiled into silicon should ideally be procedural, a more appropriate internal representation is required for translation into hardware. This representation should effectively model the data flow paths as well as the transformations which are performed on the data during the execution of the algorithm. These aspects are captured effectively by a graph representation of the input algorithm since the data flow and transformation semantics are explicitly represented by the edges and vertices of the corresponding graph representation.

A program graph (P-graph)model of the input algorithm is described below.

Also, the data flow analysis and data path extraction and optimization (schedule determination) phases are defined in the context of this model. Algorithms for determining optimal schedules and upper- and lower-bounds on the corresponding resource requirements for the class of program graphs of interest are described subsequently. Finally, these algorithms are proved to be correct and their time complexity is shown to be proportional to **n**, the number of tasks to be scheduled in the worst case.

Formally, a program graph $G = <V,E>$ is a partial order where V and E represent the vertex and edge sets respectively. The vertex set V is partitioned into n equivalence classes $\gamma_1 \ldots \gamma_n$. The $v_i \epsilon\ \gamma_1$ are termed value transporters and hence γ_1 represents the value transporter set TP. The remaining vertices $v_j \epsilon\ (V-\gamma_1)$ constitute the value transformer set TF. Note that the $e_i \epsilon\ E$ specify the data flow requirements. The example illustrated in Figure 3 aids in understanding these definitions. The algorithm indicated in Figure 3a computes the gradient of a 4x4 grey level image. A 3x3 (Figure 3b) window is used for this purpose and the corresponding program graph is illustrated in Figure 3c. It can easily be seen that G is a forest of in-trees with $TP=\gamma_1$. Also, the equivalence relations defining γ_2, γ_3 and γ_4 correspond to addition, subtraction and absolute value determination respectively and collectively define the TF set.

The $v_i \epsilon$ TP represent the I/O requirements of G. Thus, these vertices represent value sources and sinks responsible for transporting data values into and out of the program graph. The execution environment consists of these value transporters (members of TP) pumping values into the value transformers (members of TF). Subsequent to completion of the computation represented by G, the results are in turn pumped out of the graph. Note that memory external to the module encapsulating the architecture is implicit to this characterization. For all $v_i \epsilon$ TP, $d^+(v_i) = 0$, iff, $d^-(v_i) \neq 0$ and vice-versa where $d^+(v_i)$ and $d^-(v_i)$ represent the in- and out-degrees of vertex v_i respectively. Also, $d^+(v_i)$ and $d^-(v_i) \neq 0$ for all $v_i \epsilon$ TF.

2.2 Intermediate Architecture Generation

The two major phases constituting high-level optimization, namely:

1. data flow analysis and

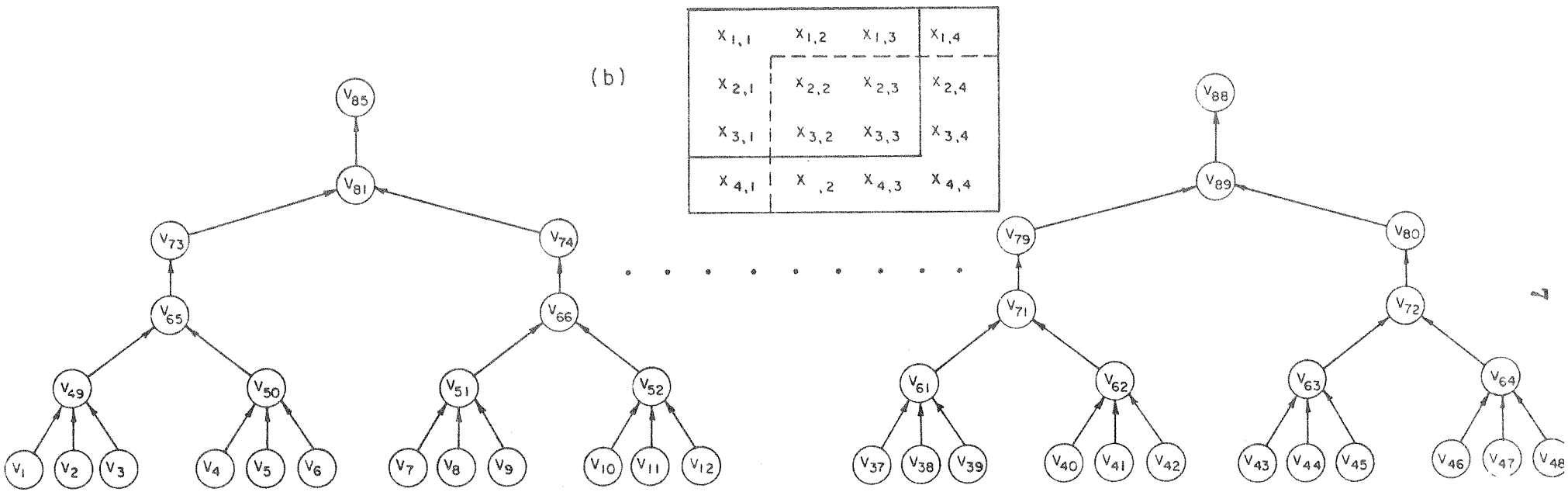
2. data path extraction

are described in this section.

The computation represented by G is initially viewed in a logical framework in which execution of the algorithm is equivalent to completion of the tasks represented by the corresponding vertices and edges. We may consider each of these tasks to be a primitive action. Task completion is accomplished by **assigning** resources which

```
For each window position do
    begin
        Add elements of the leftmost column in
        the window;
        Add elements of the rightmost column in
        the window;
        Obtain the absolute difference of the above
        two sums (A1);
        Add the elements of the top row in the window;
        Add the elements of the bottom row in the window;
        Obtain the absolute difference of these
        two sums (A2);
        Compute the sum of A1 and A2 as the gradient
        value at that point;
    end;
```

Fig. 3(a). Procedural description of the gradient operation.



(c)

γ_2 (addition) $= \{v_{49} - v_{64} \cup v_{81} - v_{84}\}$

γ_3 (subtraction) $= \{v_{65} - v_{72}\}$

γ_4 (Absolute value) $= \{v_{81} - v_{84}\}$ $TP = \{\gamma_2 \cup \gamma_3 \cup \gamma_4\}$

$\gamma_1 = TF = \{v_1 - v_{48} \cup v_{85} - v_{88}\}$

Fig. 3(b). The 4X4 image with the 3X3 window centered on (i,j), $\{i,j = 2,2$ (solid line) and $3,3$ (dashed line)$\}$.
(c). Corresponding P-graph.

are capable of completing these tasks. These resources are drawn from a resource set defined below. For example, a vertex representing addition would require that a resource of type 'adder' be assigned to it when its input data is available. More precisely, the execution of the algorithm specified by G can be viewed as a schedule S which consists of a sequence of assignment functions. Each function in this sequence assigns resources and therefore is responsible for the completion of some subset of all the tasks specified by G. It is important to note that data paths and I/O requirements are being treated as resources in a uniform fashion.

The Resource Set R is partitioned into $\mathbf{n+1}$ equivalence classes $\Pi'_1 \ldots \Pi'_{n+1}$. Each $\Pi'_i \subseteq R$ is in a one-to-one correspondence with the equivalence class $\gamma_i \subseteq V$ for $1 \leq i \leq n$. The equivalence relation defining Π'_{n+1} is of particular interest since its members are resources of type 'data path' and therefore correspond to the edge set E of G. Thus, a task specified by a member of γ_i $1 \leq i \leq n$ is completed by some $r_i \in R$ if and only if $r_i \in \Pi'_i$.

Note that the schedule determination process requires that an appropriate resource estimate be made. This estimate must clearly be sufficient to complete all the tasks as specified by the schedule. However, these steps do not completely specify the topology of the intermediate architecture since the data paths connecting the various resources are not bound as yet. This task is accomplished by the data path extraction and optimization phase as shown in Figure 2. The schedule S and the corresponding resource estimates serve as inputs to this stage. The schedule manifests itself as the control function or controller in the intermediate architecture or IA (Figure 4). In order to ensure optimality, the data paths must be bound in a fashion amenable to subsequent area efficient layout.

It should be noted that the following points of practical significance are implicitly accounted for by this approach. First, it is necessary to be able to specify time or cost bounds on the system being designed. Note that both these alternatives are conveniently handled by the schedule determination phase. While time bounds are directly reflected in the length of the schedule S, cost bounds can easily be incorporated into the corresponding resource estimation procedures by associating appropriate cost labels with the $r_i \in R$. Also, the partition defined on R reflects the real-world standard cell based design environment. The various components of the architecture are drawn from a predefined library of standard cells in such an environment. Note that different types of cells are allowed in the library where each cell type is capable realizing a specalized function or task. Thus the essential characteristics of a realistic design environment are reflected in this approach.

3 Data Flow Analysis Results

Recall that the process of deriving a schedule involves determining a sequence of assignment functions subject to the precedence and compatibility constraints imposed by G. In formally stating this problem, it is convenient to deal with G', an instance of a precedence graph (PR-graph) derived by homeomorphically transforming G. This transformation is illustrated in Figure 5 where each $e_i \epsilon$ E is replaced by a pair of edges in series. Note that the vertices of G' represent tasks and that the transformation is unique. Thus, G'=<T,E'> is the PR-graph corresponding to G where T is the task set and E' represents the precedence relationship between these tasks. Accordingly, the vertex set T of G' is partitioned into n+1 equivalence classes $\Pi_1,\Pi_2...\Pi_{n+1}$ where the Π_i are essentially the same as $\gamma_i \epsilon$ V for $1 \leq i \leq n$. The vertices introduced through the transformation belong to Π_{n+1} and represent the data flow paths.

A schedule S can now be defined as a sequence of assignment functions represented by the m-tuple $<\sigma_1,\sigma_2...\sigma_m>$ such that $\sigma_i: R_i \rightarrow T_i$ is a bijection where $R_i \subseteq R$ and $T_i \subseteq T$. In order to ensure that the precedence constraints are preserved, it is required that $\sigma_i\{r_j | r_j \epsilon R_i\} = \{t_k | t_k \epsilon T_i\}$ iff every $t_j \epsilon$ T which has a path to t_k is a member of some T_l for $l<i$. The compatibility constraints can be incorporated into the definition of σ_i by requiring that $\sigma_i(r_j)=t_k$ iff $t_k \epsilon \Pi_l \Rightarrow r_j \epsilon \Pi_l'$ for all l. Finally, note that $\bigcup_{i=1}^{m} \sigma_i(r_j)=T$ and $\sigma_i(r_k) \neq \sigma_j(r_k)$ for $i \neq j$ and all k.

Goyal [7] has shown that the problem of finding a schedule for the case where each task requires unit execution time (UET) is NP-complete even when the precedence constraints are in the form of an arbitrary forest. It is assumed that each distinct resource class corresponding to the Π_i' definition has exactly one element in it. However, several problems of interest exemplified by matrix multiplication and a large class of operations drawn from the area of image processing including algorithms defined on pyramid architectures [8,9],point transforms, neighborhood operations[10-12] and others can be represented by a special class of PR-graphs; the class of **well-behaved** (WB) trees. These are rooted directed trees such that all the tasks t_i at the same level of the tree belong to the same Π_j for some **j**. Also, all the t_i at the same level of G' have the same in(out)-degree $d^-(t_i)$ ($d^+(t_i)$) and G' is an in(out)-tree. The level of a vertex is its distance from the root. For example, the PR-graph G' corresponding to the gradient operation (Figure 5) belongs to this class. In the sequel, our analysis will be restricted to PR-graphs which are WB-trees and we shall show that polynomial time optimization can be done for this class.

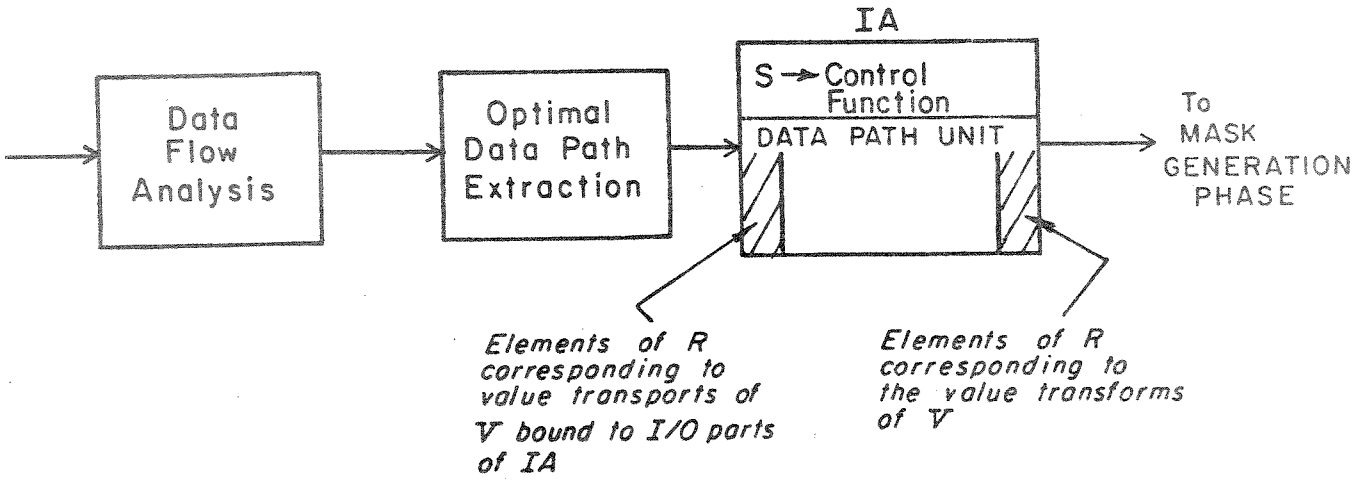


Fig. 4. IA generation through high level optimization

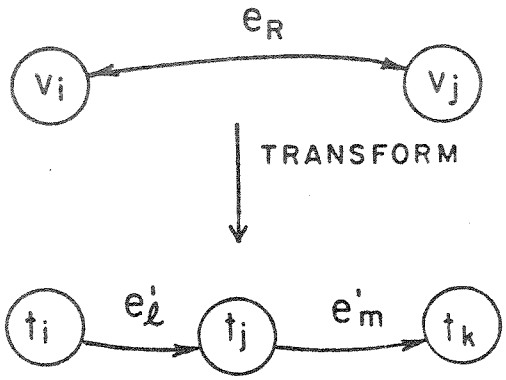


Fig. 5. Transforming a P-graph into a PR-graph

3.1 Upper Bound Determination

The scheduling problem considered in this subsection involves the determination of an upper bound on the number of resources required and a schedule for a given PR-graph G' which is a well behaved tree. A linear-time algorithm is described which estimates this upper bound given a description of G' and the value of **m**, the length of the schedule. Recall that UET systems are being considered. The algorithm and the accompanying schedule are shown to be correct.

Given G', consider the following labeling procedure which is defined recursively:

Base: Label the root '0'

Recursion: Label the children of each $t_1 \epsilon$ T uniquely

using the elements of the set defined by

$$L=l(t_1).d^-(t_1)+k \quad 0 \leq k < d^-(t_1)$$

where $l(t_i)$ is the label of task t_i. The result of applying this labeling procedure to the WB-tree of Figure 5 is illustrated in Figure 6. Note that the labels are indicated adjacent to the vertex and are drawn from **N**, the set of natural numbers. If G' is a forest, the labeling procedure is applied to an equivalent tree which can be constructed by introducing a virtual root as the parent of all the true roots (Figure 6).

The modification to the assignment function described below is central to the determination of the upper bounds and the schedule. This modification associates resources with levels of G' in that once a resource is assigned to a task at a certain level **i**, it can never be re-assigned to a task at level **j** on a later step unless i=j. This relationship partitions R associated with G' of height **h** into an equal number of equivalence classes Θ_i $0 \leq i \leq h-1$. So long as G' belongs to the class of WB-trees, all the tasks at a given level belong to the same Π_j implying that $\Theta_i \subseteq \Pi_j$ for some **j**. The assignment function σ'_i is got from σ_i by adding the additional constraint that $\sigma_i(r_j)=t_k$ iff level $(t_k)=m \Rightarrow r_j \epsilon \Theta_m$. Thus, we have a **roll-over** schedule RS of length **m**, defined by the m-tuple $<\sigma'_1,\sigma'_2 \ldots \sigma'_m>$ such that given some 'R', the number of tasks that are assigned resources by σ'_i is maximized for all **i** and the order of assignment is monotone increasing on the labels.

The example of Figure 7 illustrates this algorithm. The corresponding assignment sequence is indicated in the window (i,j) where **i** spans the sequence of assignment functions and **j** represents the levels of G'. Note that the assignment function σ'_i defines the tasks (indicated by their task labels t_i and distinct from those derived from the labeling scheme) being assigned resources at each level **j**. It should be noted that the choice of the cardinality of each of the Θ_i is not arbitrary for reasons to be discussed later. Let the set of all tasks at level **j** which are enabled on step **i** be formally denoted by $M_{i,j}$. Thus, each entry in a table such as that in Figure 7 cor-

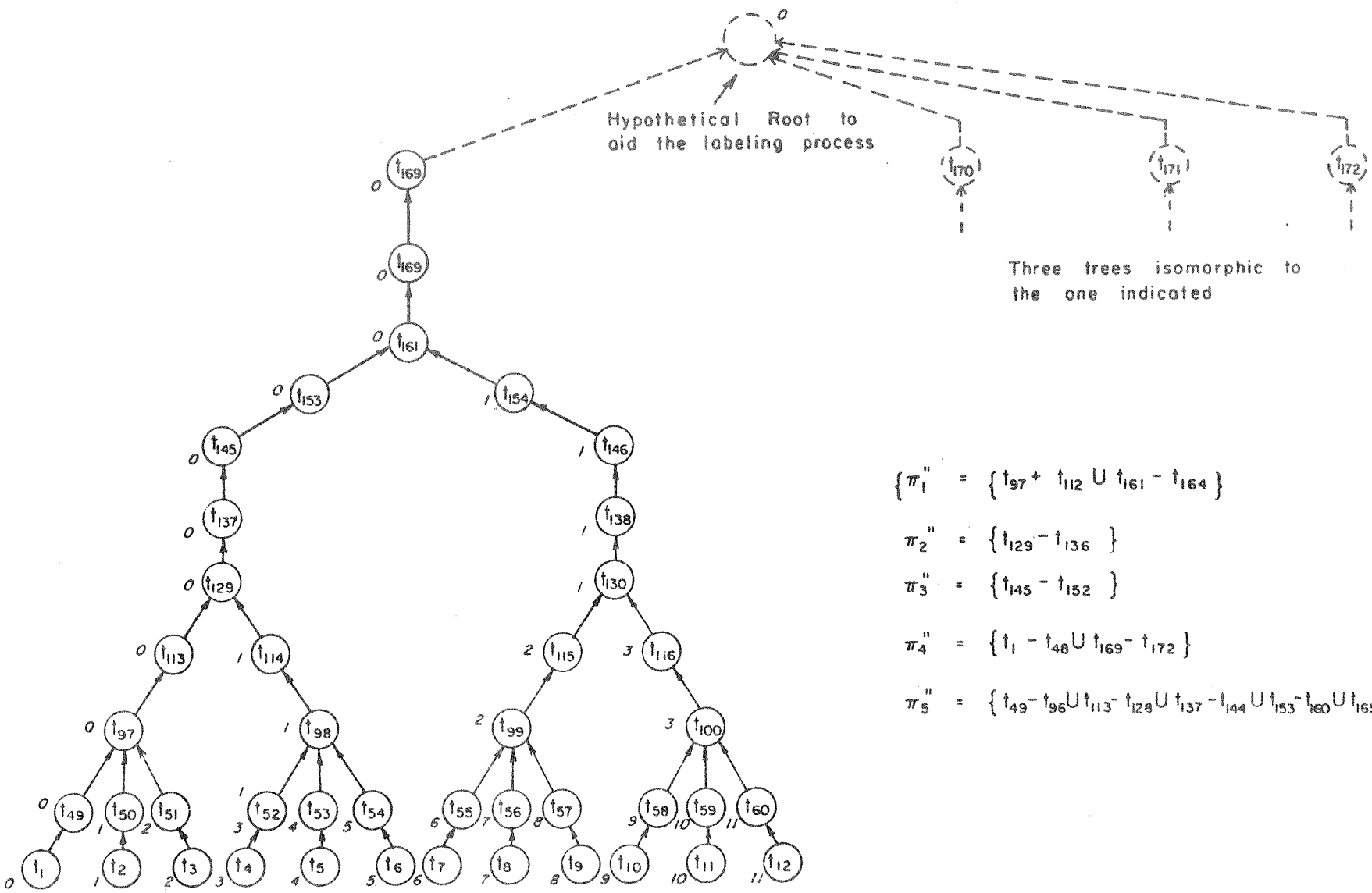


Fig. 6. The PR-graph G' corresponding to the P-graph G' of Fig. 3. Only one of the four trees is indicated for the sake of brevity.

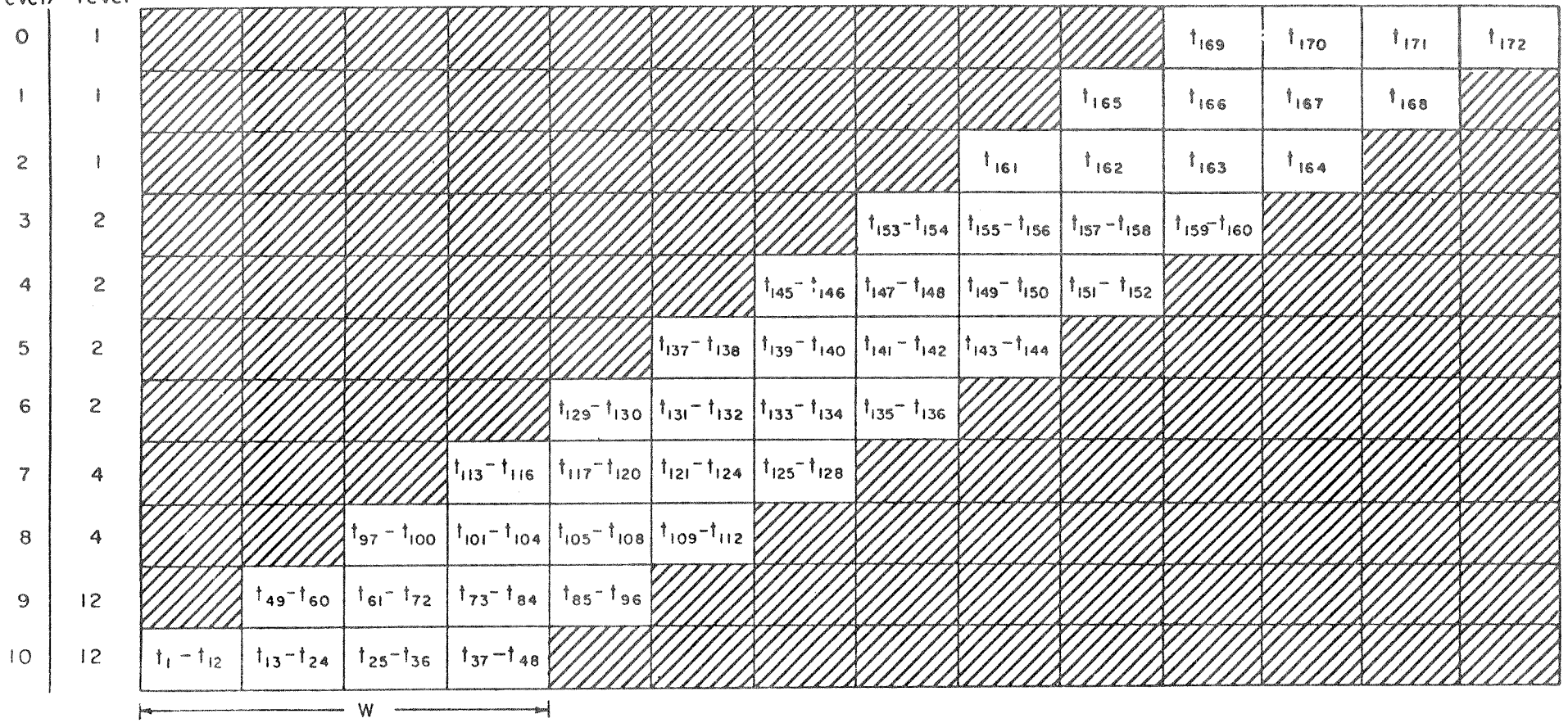


Fig. 7. The roll-over scheduling procedure for h=11 and m=13

responds to some subset of $M_{i,j}$ since in general not all the enabled tasks are immediately assigned resources. Also, $|M_{i,j}|$ represents the cardinality of this set.

Referring back to Figure 7, observe that the shaded areas indicate periods of inactivity. Let p_j denote the minimum value of **i** when tasks are enabled at level j. Also, q_j denotes the corresponding maximum value of **j**. Then, the **active period** of level **j** is represented by the value of **i** such that $p_j \leq i \leq q_j$. It then follows from the above discussion that p_j and q_j trail (lead) p_{j+1} and q_{j+1} by one unit respectively when G' is an in(out)-tree. Also, $q_j - p_j + 1$ represents the maximum length of the active period at level **j** for a given G' and a RS.

It is easy to see that using RS, the minimum cardinality of Θ_j needs to be determined independently for each level of G'. If RS is of length **m**, note that all tasks at level **j** have to be completed by $q_j = m-j$. Also, the smallest value of **i** given by p_j when a task can be initiated, is h-j where G' is an in-tree of height **h**. Therefore, the maximum length of the active period at level **j** for an RS of length **m** given a G' of height **h** is a constant $q_j - p_j + 1 = m-h+1$. for all **j**. Observe that while the analysis is indicated only for the in-tree case, the result can easily be generalized to include out-trees as well. Hence, it can be concluded that $|\Theta_j|_{min} = n_j = \lceil x_j / W \rceil$ where $W = m-h+1$ and x_j equals the number of tasks at level **j** of G'. Algorithm 1 can be used to estimate $|\Theta_j|_{min}$ for all **j** given the description of G' in the form of a vector of x_js, its height and a specification of **m**:

```
ALGORITHM 1:
begin
   W=m-h+1
   for i=0 to h-1 in 1 do
      begin
         n_j = ⌈x_j/W⌉
      endfor
   end
```

Note that the $|\Theta_j s|$ used in the example of Figure 7 were determined by applying this algorithm.

It remains to be shown that the cardinality of R as estimated by Algorithm 1 is sufficient to ensure correctness when roll-over scheduling is used. Recall that G' is a well behaved in-tree and d_j^- represents the in-degree of vertices at level 'j ' since all these vertices have the same in-degree by definition.

Then, it can be stated that

Lemma 1: For any given $n_{j+1} = k_{j+1}\, d_j^- + m_{j+1}$ such that k_j and m_j are integers and

$k_j \geq 0$, $0 < m_j < d_j^- - 1$, the number of tasks enabled at any level j $0 \leq j < h-1$ of a WB-intree G' using RS, $|M_{i,j}|$, is such that $k_{j+1} \leq |M_{i,j}| \leq k_{j+1}+1$ for $p_j \leq i < q_j$. Furthermore, the upper bound also holds for $i = q_j$.

Proof: The proof proceeds by induction on j, the number of levels in G'.

Base Case: The result is established at level h-2 preceded only by the leaf level h-1. Expressing $n_{h-1} = k_{h-1} \cdot d_{h-2}^- + m_{h-1}$ and noting that on step $i < q_{h-2}$ using RS, the n_{h-1} resources span at least k_{h-1} subtrees rooted at level h-2, the lower bound on $|M_{i,h-2}|$ is easily established for $2 \leq i < q_{h-2}$.

The situation depicted in Figure 8 aids in establishing the upper bound that is $|M_{i,h-2}| \leq k_{h-1}+1$ for $i \leq q_{h-2}$. Observe that $m'_{h-1} + m''_{h-1} = m_{h-1}$. In order to maximize the number of tasks enabled at level h-2 on some step $i \leq q_{h-2}$, m''_{h-1} needs to be maximized. Since $m_{h-1} < d_{h-2}^-$, it can be inferred that the m''_{h-1} resources cannot enable a task at level h-2. However, the m'_{h-1} resources can enable a task since the $d_{h-2}^- - m'_{h-1}$ resources might be enabled on the previous step i-1. Therefore, $|M_{i,h-2}| \leq k_{h-1}+1$ for $i \leq q_{h-2}$.

Induction Hypothesis: Let the lemma be true for all levels $> r$, $0 \leq r \leq h-1$.

Induction : It needs to be shown that the lemma holds for level r.

Since $n_{r+1} = \lceil \overline{x}_{r+1} / \overline{W} \rceil$ from Algorithm 1 it can be inferred that $n_{r+1} \leq k_{r+2} + 1$.

Case a: ($n_{r+1} < k_{r+2}$): The proof for this case follows exactly along the lines indicated in the base step with h-1 and h-2 replaced by r+1 and r respectively.

Case b: ($n_{r+1} = k_{r+2}+1$) Expressing n_{r+1} as $k_{r+1}\, d_r^- + m_{r+1}$ and invoking the induction hypothesis, we have $k_{r+2}+1 \geq |M_{i,r+1}|$ for $p_{r+1} \leq i \leq q_{r+1}$. Also, $|M_{i,r+1}| \geq k_{r+2}$ for $p_{r+1} \leq i < q_{r+1}$. If $|M_{i,r+1}|$ is $k_{r+2}+1$ for $p_{r+1} \leq i \leq q_{r+1}$, then the result follows from the arguments used to establish the result in the base step. However, to prove the lemma without loss of generality, the case where $|M_{i,r+1}| = k_{r+2}$ has to be considered for some i such that $p_{r+1} \leq i \leq q_{r+1}$. Clearly, this does not affect the upper bound that is $|M_{i,r}| \leq k_{r+1}+1$ for $p_r \leq i \leq q_r$. In order to verify that the assertion regarding the lower bound holds, it is sufficient to note that since $m_{r+1} > 0$, $k_{r+2} \geq k_{r+1} \cdot d_r^-$ completing the proof.

A direct extension of this lemma is the case where $m_{r+1} = 0$ and $0 \leq r+1 \leq h-1$ yielding $n_{r+1} = k_{r+1} \cdot d_r^-$ for some $k_{r+1} > 0$. This extension is formally stated below :

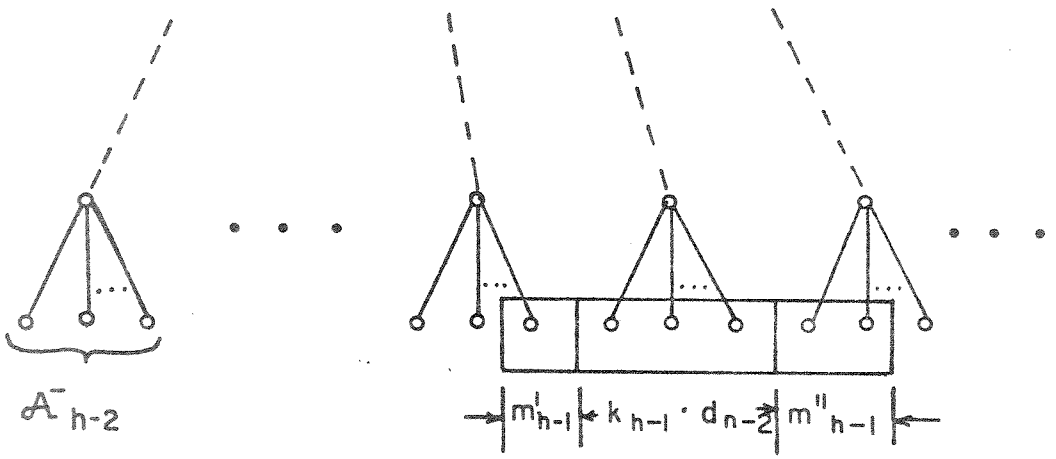


Fig. 8. Illustrating the upper bound on $|M_{i,h-2}|$.

Corollary 1: If $m_{r+1}=0$ for some level 'r' $0\leq r\leq h-1$ implying that $n_{r+1}=k_{r+1}.d_r^-$, then $k_{r+1}-1\leq|M_{i,r}|\leq k_{r+1}$ for $p_r\leq i<q_r$. Furthermore, the upper bound also holds for $i=q_r$.

Proof: Immediate from Lemma 1.

Consider the mismatch between the number of resources available n_j and $|M_{i,j}|$ for $p_j\leq i\leq q_j$ and some **j** represented by $|M_{i,j}|-n_j=\delta(i,j)$. It then follows that a perfect match will yield a $\delta(i,j)$ of 0. Also, $\delta(i,j)>0$ signifies task overflow implying that an insufficient number of resources were available. However, this overflow can be compensated for by a $\delta(k,j)<0$ of equal magnitude for some $q_j\geq k>i$ since a negative $\delta(k,j)$ signifies an excess of resources. Thus, Δ_j corresponding to level **j** can be recursively defined as follows:

Base: $SS\delta(p_j,j) = \delta(p_j,j)$

Recursion: If SS $\delta(i-1,j)\geq 0$, then
SS $\delta(i,j)$=SS $\delta(i-1,j)+\delta(i,j)$
else $\delta(i,j)$

Restriction: $q_j\geq i\geq p_j$ and $0\leq j\leq h-1$.

$$\Delta_j = SS\ \delta(i,j).$$
$$p_j\leq i\leq q_j$$

Based on the above discussion, it can be inferred that

Lemma 2: A necessary and sufficient condition for n_j resources to complete the tasks at level **j** whose active period is bounded by p_j and q_j is $\Delta_j\geq 0$.

The results indicated above are sufficient to state that:

Theorem 1: The n_j estimates determined by algorithm 1 provide an upper bound on $|\Theta_j|$ $0\leq j\leq h-1$ for a PR-graph G' using a roll-over schedule of length **m**.

Proof: The number of tasks enabled during any step **i** at level **j** that is $|M_{i,j}|$ is bound above by some integer l_j and the corresponding lower bound equals l_j-1 for $p_j\leq i<q_j$ from Lemma 1 and Corollary 1 for all levels of G' provided RS is used. Also, the upper bound equals l_j for $i=q_j$.

Case a: $n_j = l_j$
It follows immediately that $\Delta_j \leq 0$.

Case b: $n_j < l_j$: This case is proved by contradiction. Assume $\Delta_j > 0$. Then, $W.n_j + \Delta_j = x_j$ for some $\Delta_j > 0$ implying that $n_j.W < x_j$. But since $n_j = \lceil x_j/W \rceil$, $n_j.W \geq x_j$. Since this leads to a contradiction the theorem is proved.

3.2 Lower Bound Determination

It was pointed out earlier that the maximum length of an active period at any level j of G' is a constant W. If the constraint imposed by Θ_i is relaxed,then any $r_i \epsilon$ R can be assigned to a task at any level so long as the compatibility constraint is satisfied. This leads to a more general definition of a roll-over schedule. Under this new definition, elements of the resource set can be assigned to compatible tasks independent of the level in G' to which the task belongs.

If a certain type of task appears at exactly one level of G', say j and is of type l, then the minimum cardinality of Π'_l represented by $|\Pi'_l|$ is given by $n'_j = \lceil x_j/W \rceil$. Note that a specification of G' and m is implicit at this point. This estimate can be extended to include the case where the type l resource occurs at two levels of G' say j and j-k for some $k>0$ and G' is an in-tree. Then, the overall active period associated with elements of Π'_l is bounded above by $q_{j-k} - p_j + 1 = W + k$ (Figure 9). Thus, the corresponding lower-bound on $|\Pi'_l|$ will be $\max\{\lceil (x_j/W) \rceil, \lceil (x_j + x_{j-k})/W+k \rceil\} = \max(n'_j, n'_{j-k})$ and can likewise be inductively extended to the case where tasks of type l occur at Φ levels $l_1\ l_2 \ldots l_\Phi$.

The algorithm outlined below can be used to determine these lower bounds on all the equivalence class Π'_k in R given an instance of the PR-graph G' and m.

Algorithm 2:

1. level l is initially h-1. //Let the tasks at level l belong to class Π_k//

2. Compute Temp = n^*_1 //Estimate new max.//

3. Replace current lower bound of Π^*_k by max[current lower bound, Temp]

4. Repeat steps 2 and 3 for levels h-2 to 0 in that order.

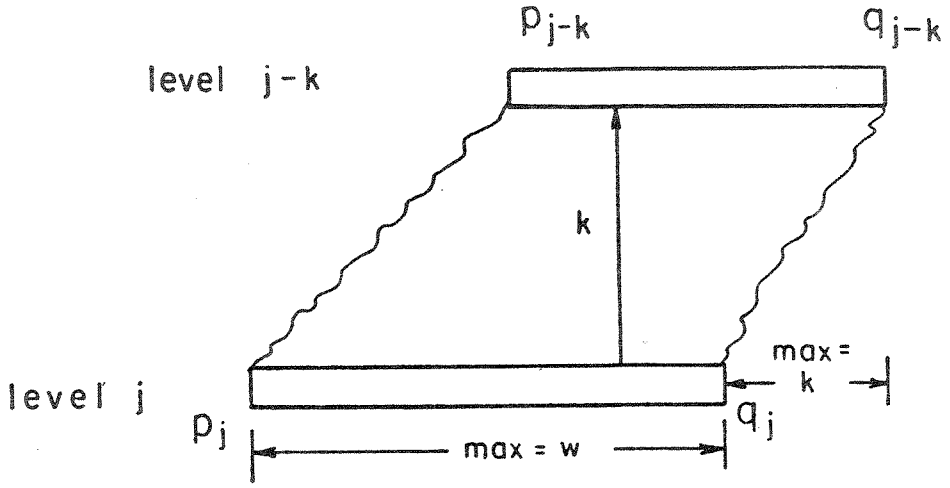


Fig. 9. Active periods for the resource re-use case where G' is an in-tree.

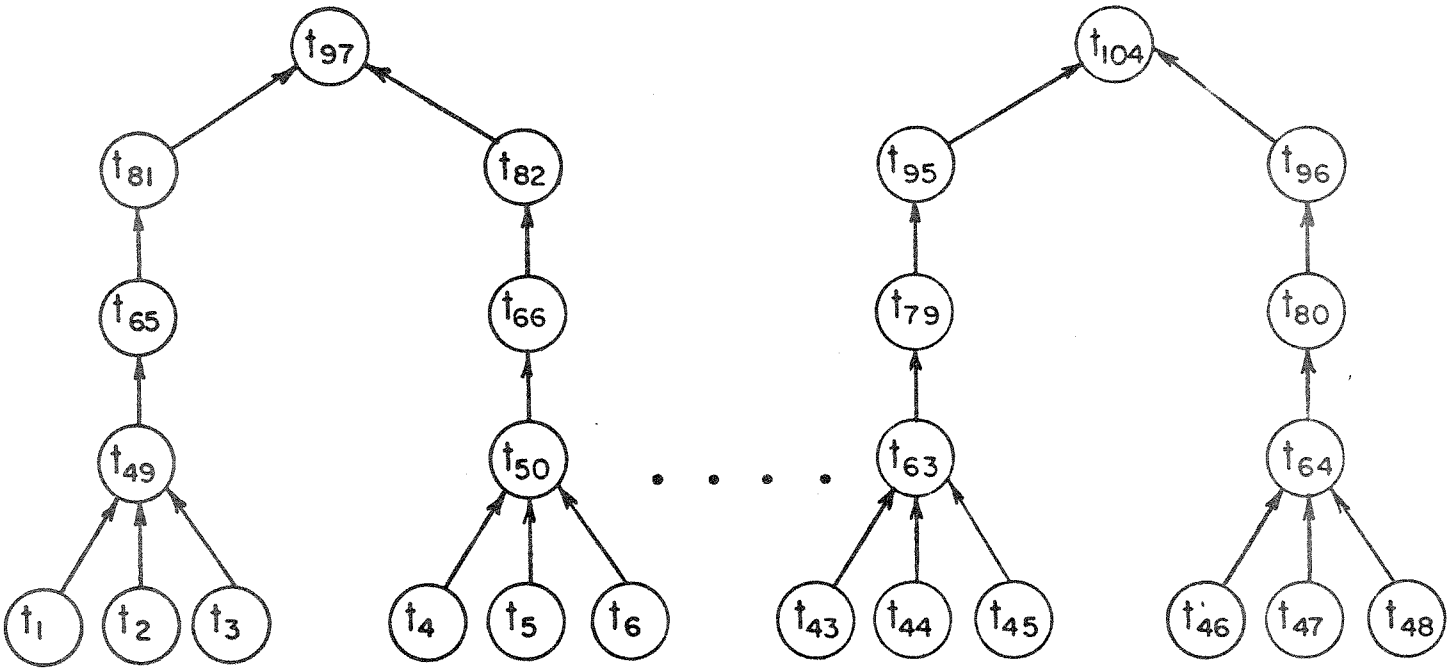
It follows from the arguments stated earlier in this subsection that Algorithm 2 computes the lower bound for each $\Pi'_k \subseteq R$. Note that the distinction between the in- and out-tree cases was not made since algorithm 2 is applicable to the entire class of WB-trees. Observe that Algorithm 2 iterates (step 4) along the direction of data flow for in-trees and against it for out-trees. This observation is central to developing arguments for establishing the correctness of Algorithm 2 for the out-tree case.

3.3 Comments on the Tightness of the Bounds and Complexity of the Algorithms

The example illustrated in Figures 10 a and 10 b will be used to establish the tightness of the lower bound on the resource requirements. As indicated in Figure 10b, the resource estimates as determined by Algorithm 2 are sufficient to determine a correct schedule. Also, note that the length of the schedule was chosen to be ten steps that is m=10. The meanings of the entries in the table of Figure 10b and all the related details have already been explained in the context of Figure 7. Finally, in order to establish the tightness of the upper bound on R as determined by Algorithm 1, it is sufficient to consider any G' all of whose levels are distinct, that is, $\Pi_i = \Theta_i$ for $0 \leq i \leq h-1$ where G' is of height **h**.

In order to estimate the time complexity of Algorithms 1 and 2 note that both of them iterate on the number of levels in G'. This value is a maximum when the tasks in question define a linear ordering on the set T and is proportional to **n**, the number of tasks. Thus, the time complexity of the iterative loop is O(n). In the case of Algorithm 1, it is immediately obvious that each action included in the body of the iteration requires constant time. While this is not immediately obvious in the case of Algorithm 2, it is indeed the case. This is achieved by maintaining the additional information regarding the level at which tasks $t_i \in \Pi_k$ first appeared and the corresponding current lower bound estimates of $|\Pi'_k|$. Thus, the time complexity of Algorithms 1 and 2 is O(n).

Note that the complexity of determining a roll-over schedule is determined by the labeling procedure defined earlier. This procedure starts at the root and visits each of the tasks exactly once. Thus, it requires exactly (cn) steps for some constant $c>0$ given **n** tasks and **c** steps are required to label each task. Equivalently, the time complexity of determining a roll-over schedule given $G'=<T,E'>$ and $|T|=n$ is O(n). Thus, the cumulative time-complexity of the procedure for determining bounds on R and **roll-over schedules** for PR-graphs which are WB-trees with **n** vertices (tasks) is O(n).



$$\pi_1'' = \{ t_1 - t_{48} \cup t_{81} - t_{96} \}$$

$$\pi_2'' = \{ t_{49} - t_{64} \}$$

$$\pi_3'' = \{ t_{65} - t_{80} \cup t_{97} - t_{104} \}$$

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \\ 16 \\ 16 \\ 48 \end{bmatrix}$$

Fig. 10(a). Establishing the tightness of the lower bound.

				0	0	0	$t_{97} - t_{99}$	$t_{100} - t_{102}$	$t_{103} - t_{106}$
			0	0	0	$t_{81} - t_{88}$	$t_{89} - t_{83}$	$t_{94} - t_{96}$	
		$t_{65} - t_{66}$	$t_{67} - t_{69}$	$t_{70} - t_{72}$	$t_{73} - t_{74}$	$t_{75} - t_{77}$	$t_{78} - t_{80}$		
	$t_{49} - t_{50}$	$t_{51} - t_{53}$	$t_{54} - t_{56}$	$t_{57} - t_{58}$	$t_{59} - t_{61}$	$t_{62} - t_{64}$			
$t_1 - t_8$	$t_9 - t_{16}$	$t_{17} - t_{24}$	$t_{25} - t_{32}$	$t_{33} - t_{40}$	$t_{41} - t_{48}$				

$$\begin{bmatrix} |\pi_1'| \\ |\pi_2'| \\ |\pi_3'| \end{bmatrix}_{\min} = \begin{bmatrix} 8 \\ 3 \\ 3 \end{bmatrix}$$

Fig. 10(b). m=10

4 Conclusions and Recommendations

The essential components of a silicon compiler particularly relevant in the design of custom VLSI have been described. The need for a high-level optimization phase was emphasized. This problem was shown to be composed of a data flow analysis phase and a data path extraction and optimization phase. These steps are together instrumental in generating an intermediate description of the target architecture whose correctness is easily verified. Efficient algorithms for determining schedules and appropriate resource estimates have been described and shown to be correct.

The scheduling procedures accept time deadlines as an input parameter. Thus, the time constraints imposed on the architecture are handled naturally by the high-level optimization phase. Mismatch between the I/O bandwidth and the computing elements encapsulated in a VLSI chip poses serious restrictions on the designs. However, this problem is circumvented by the data flow analysis phase since I/O bandwidth is modeled as a resource thus enabling the estimation algorithms to ensure an efficient match. Also, cost paramaters can be introduced by associating appropriate cost labels with the various elements of the resource set.

Issues related to the data path extraction and optimization phase are currently being investigated. An important part of this work involves identifying transformations for binding data paths in an optimal fashion. It is important to determine methods for synthesizing layouts which involve replication of a small set of building blocks in a regular fashion. Related questions regarding optimal distribution of control are also of relevance.

5 References

1. G. J. Lipovski and A. Tripathi, A Reconfigurable Varistructured Array Processor, International Conference on Parallel Processing, August 1977, pp. 165-174.

2. Computer, Vol. 15, No. 1, January 1982.

3. C. A. Mead and L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1980.

4. H. T. Kung, Why Systolic Architectures? in [3].

5. D.Johannsen, Bristle Blocks: A silicon compiler, Proc. 16th Design Automation Conference, 1979 pp310-313.

6. J.M.Siskind, J.R.Southard, and D.W.Crouch, Generating Custom High Performance VLSI Designs from Succint Algorthmic Descriptions, Proc. Conference on Advanced Research in VLSI, 1982, pp28-40.

7. D.K.Goyal, Scheduling Processor Bound Systems, TR-CS-76-035, Computer Sciences Department, Washington State Universtiy, Pullman, Wash., November 1975.

8. L.Uhr, Converging Pyramids of Arrays, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, 1981, pp 31-34.

9. C.R.Dyer, A VLSI Pyramid Machine for Heirarchical Parallel Image Processing, Proc. of the Pattern Recognition and Image Processing Conference, 1981, pp381-386.

10. G. R. Nudd, Image Understanding Architectures, Proc. of the Nationa Computer Conference, 1980, pp. 377-390.

11. H. T. Kung and S. W. Song, A Systolic 2-D Convolution Chip, Proc.

IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, 1981, pp. 159-160.

12. S. L. Tamimoto, Systolic Cellular Logic: Inexpensive Parallel Image Processors, Proc. Pattern Recognition and Image Processing Conference, August 1981, pp. 306-309.