6-1-1987

A Decomposition Approach for Balancing Large-Scale Acyclic Data Flow Graphs

P. R. Chang
Purdue University

C. S. G. Lee
Purdue University

A Decomposition Approach for Balancing Large-Scale Acyclic Data Flow Graphs

P. R. Chang
C. S. G. Lee

TR-EE 87-23
June 1987

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

A Decomposition Approach for Balancing Large-Scale Acyclic Data Flow Graphs

P. R. Chang and C. S. G. Lee

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE 87-23

June 1987

Abstract

In designing VLSI architectures for a complex computational task, the functional decomposition of the task into a set of computational modules can be represented as a directed task graph, and the inclusion of input data modifies the task graph to an acyclic data flow graph (ADFG). Due to different paths of traveling and computation time of each computational module, operands may arrive at multi-input modules at different arrival times, causing a longer pipelined time. Delay buffers may be inserted along various paths to balance the ADFG to achieve maximum pipelining. This paper presents an efficient decomposition technique which provides a more systematic approach in solving the optimal buffer assignment problem of an ADFG with a large number of computational nodes. The buffer assignment problem is formulated as an integer linear optimization problem which can be solved in pseudo-polynomial time. However, if the size of an ADFG increases, then integer linear constraint equations may grow exponentially, making the optimization problem more intractable. The decomposition approach utilizes the critical path concept to decompose a directed ADFG into a set of connected subgraphs, and the integer linear optimization technique can be used to solve the buffer assignment problem in each subgraph. In other words, a large-scale integer linear optimization problem is divided into a number of smaller-scale subproblems, each of which can be easily solved in pseudo-polynomial time. Examples are given to illustrate the proposed decomposition technique.

This work was supported in part by the National Science Foundation Engineering Research Center Grant CDR-8500022.

1. Introduction

With the advent of VLSI technology, the rapid decrease in computational costs, reduced power consumption and physical size, and increase in computational power suggest that an interconnection of VLSI processors, which are configured and arranged based on a functional decomposition of the computational task to exploit the great potential of pipelining and multiprocessing, provides a novel and cost-effective solution for many computational problems in pattern recognition [6], signal processing [12], and robotics [14]-[15]. This type of computational structure has been referred to as a *systolic* array or system [10]. One of the main advantages of using a systolic array is that each input data item can be used a number of times once it is accessed, and thus, a high computation throughput can be achieved with only a modest bandwidth. Other advantages include modular expandability, and simple and regular data and control flow.

In general, a computational task of interest is partitioned or decomposed into a set of smaller computational modules, and the interconnection of these computational modules can be represented as a directed task graph. The inclusion of input data modifies the task graph to an acyclic data flow graph (ADFG). The nodes of an ADFG correspond to the computational modules, each of which can be realized by a linear pipelined functional unit for increasing the system throughput [11]. The operands or data move along the edges, each of which connects a pair of nodes. Due to different computational time of the modules, data flow (both inputs and results from one module to another) in an ADFG may occur at different speeds in different directions. Thus, operands may arrive at multi-input modules at different arrival times, causing an unnecessary longer pipelined time in the ADFG. A conventional approach is to insert delay buffers (FIFO queues) at various paths to buffer the inputs or the output results from one module to another to achieve a balanced (or synchronous) ADFG. This is exemplified in Figure 1(a) which is a graph and consists of nodes A, B, and C whose numbers of computing stages are assumed to be, respectively, 3, 5, and 6. From the figure, there are two paths from node A to node C. For path 2, it takes 5 computing stages before an operand arrives at node C, while path 1 requires no computing stages. Node C can not start computation until all of its operands are available. As a result, the second set of data values can not be fed into the pipeline in 5 computing stages because data will only exist in path 1. So the minimum latency of the pipeline is greater than 5 computing stages and the maximum throughput is less than 1/5. To eliminate this undesirable behavior so that successive data of an array may pipeline through the ADFG with maximum throughput, a delay buffer D which is equivalent to 5 computing stages can be inserted in path 1 so that the "length" (or the cost) of path 1 and path 2 will be balanced (Figure 1(b)). Thus, the latency of the ADFG will be decreased to one computing stage, and maximum pipelining can be achieved. Once the balanced ADFG has been established, a systolization procedure can be used to transform the balanced ADFG into a systolic array [16].

The problem of balancing a directed ADFG by inserting appropriate buffers along appropriate paths to achieve maximum pipelining has been solved previously by the cut-set theorem [11]-[12], the local correctness criterion [12], and the graph-theoretic approach [4]-[5]. Furthermore, Hwang and Xu [9] showed that the balanced ADFG can be realized in a two-level pipeline network which is reconfigurable and provides the flexibility in various vector processing applications. The delay matching may be handled by programmable buffers, so that proper non-compute delays can be inserted in each data flow path. An example is the design of the LINC chip [8], which is an 8-by-8 crossbar up to 32 units of programmable delays in each data flow path.

This paper presents an efficient decomposition technique which provides a more systematic approach in solving the optimal buffer assignment problem of an ADFG with a large number of computational nodes. Since it is of vital importance to minimize the number of buffers used in a systolic system to minimize the design cost, the optimal buffer assignment problem is formulated as an integer linear optimization problem, which can be easily solved in computers in pseudo-polynomial time [18]. However, if the number of computational nodes in an ADFG is quite large, then integer linear constraint equations may grow exponentially, making the optimization problem more difficult than it should be. The construction of integer linear constraint equations in a large-scale ADFG reveals the existence of many redundant integer linear constraint equations; so, the optimization problem may become intractable. The redundant integer linear constraint equations come from the path overlapping between two paths of two different multi-input nodes. They can be removed easily by recognizing the overlapping path (or common path) traversed by different paths. In an effort to reduce the difficulty of optimizing a large number of integer linear constraint equations, an efficient and systematic decomposition technique is proposed to recognize all the decomposable subgraphs in an ADFG and generate their associated integer linear constraint equations. The decomposition approach utilizes the critical path concept to decompose a directed ADFG into a set of connected subgraphs, and the integer linear optimization technique can be used to solve the buffer assignment problem in each subgraph. In other words, a large-scale integer linear optimization problem is divided into a number of smaller-scale subproblems, each of which can be easily solved in pseudo-polynomial time. Examples are given to illustrate the decomposition approach; and, finally, the proposed decomposition technique is used to balance an interconnection of CORDIC (COordinate Rotation DIgital Computer [1], [20]) processors for computing the robot inverse kinematic position solution [15].

2. Formulation For Balancing Acyclic Data Flow Graphs

In formulating the optimal buffer assignment problem, we shall assume that the number of computing stages of any computational module of an ADFG is finite and that the execution time of any stage is a constant, called a basic time unit or stage latency. An ADFG is maximum pipelined if the minimum number of time units needed for obtaining two successive outputs from the pipeline is equal to one basic time unit. Before giving a formal formulation of the balancing problem, we concentrate our interests on single input single output (SISO) ADFG's and introduce some necessary definitions for formulation:

Definition 1: A weighted ADFG $GW = (V, E, W)$ corresponding to an ADFG $G = (V, E)$ is a weighted direct graph where W is a weight function from E to a set of non-negative real number. $V = (v_1, v_2, \cdots, v_n)$ is a finite set of computational nodes (or modules), and $E = (e_1, e_2, \cdots, e_n)$ is a finite set of edges. An edge connecting node v_i to node v_j is denoted by $e(i, j)$.

A logical way to convert an ADFG to a corresponding weighted ADFG is to assign weights to each output edge of a computational node such that the weight assigned to each edge equals to the number of the computing stages of the computational node. For example, the weight $w(e(i, j))$ assigned to the edge $e(i, j)$ equals to the number of computing stages of node v_i.

Definition 2: The cost (or weight) of any kth path $\phi_k(v_p, v_q)$ from node v_p to node v_q can be defined as the sum of the weights of all edges along the path. That is,

$$w(\phi_k) = \sum_{e(i,j) \in \phi_k(v_p, v_q)} w(e(i, j)).$$

Thus, the cost of a path from node v_p to node v_q equals to the number of computing stages needed for an operand to travel along the corresponding path from node v_p to node v_q.

Definition 3: A weighted ADFG GW with an input node u_0 is said to be balanced if the cost for any two different paths from the input node u_0 to an arbitrary multi-input node u_k is equal.

This definition indicates that a balanced ADFG achieves maximum pipelining. Unfortunately, most ADFG's derived from given tasks are usually unbalanced. To balance an ADFG, appropriate delay buffers must be inserted along appropriate paths from the input node u_0 to any particular multi-input node of interest. Thus, any different paths from the input node u_0 to a multi-input node will have equal costs. The appropriate buffering graph in which delay buffers are inserted to balance an unbalanced ADFG can be defined as:

Definition 4: Let $GW = (V\ , E\ , W)$ be a weighted ADFG and $GB = (V\ , E\ , WB)$ be a corresponding weighted graph, where the weight WB corresponds to the buffering introduced on E. Then, GB is called a buffering graph of GW. Furthermore, an ADFG $GW' = (V\ , E\ , W')$ can be composed from GW and GB such that $w'(e(i\ ,j)) = w(e(i\ ,j)) + wb(e(i\ ,j))$; for all $e(i\ ,j) \in E$, where $wb(e(i\ ,j))$ is the weight of the buffers from node v_i to node v_j. If GW' is a balanced ADFG, then GB is a balanced buffering graph for GW.

It can be shown that a buffering graph GB for a corresponding GW always exists, though it may not be unique. In order to minimize the cost for implementing an ADFG in a VLSI device, it is desirable to obtain a balanced buffering graph with a minimum number of delay buffers.

Since the cost for any two different paths from the input node u_0 to an arbitrary multi-input node u_k must be equal for a balanced ADFG, buffer delays can be applied to balance the cost for all paths from the input node u_0 to a multi-input node u_k. Assume $U = \{u_0, u_1, u_2, \cdots, u_n\}$ is a finite set of all multi-input nodes and the input and end nodes in GW and there are m_k paths from the input node u_0 to a multi-input node u_k, that is, $\phi_l(u_0, u_k)$† , $1 \le l \le m_k$ and $1 \le k \le n$. The *critical path* $\phi_{l_k^*}(u_k)$ of a multi-input node u_k in GW is the path from the input node u_0 to the node u_k, $1 \le k \le n$, having the "heaviest" path weight defined as

$$w^c(u_k) \triangleq w^c(\phi_{l_k^*}(u_k)) \triangleq \max_{1 \le l \le m_k} \sum_{e(i\ ,j) \in \phi_l(u_k)} w(e(i\ ,j)) \tag{1}$$

No other path from the input node u_0 to the node u_k can have a path weight greater than the critical path weight $w^c(u_k)$. Thus, the cost of the critical path from the input node u_0 to the end node u_n constitutes the initial delay time of the pipeline. In order to balance an ADFG, buffers $B(e(i\ ,j))$ are introduced to insert into appropriate paths $\phi_l(u_k)$, from the input node u_0 to a multi-input node u_k, $1 \le k \le n$, to achieve all paths entering the node u_k to have the same cost. That is,

$$\sum_{e(i\ ,j) \in \phi_l(u_k)} w(e(i\ ,j)) + \sum_{B(e(i\ ,j)) \in \phi_l(u_k)} |B(e(i\ ,j))| \tag{2}$$

$$= \begin{bmatrix} \text{The critical path} \\ \text{cost of } u_k \\ \text{in } GW \end{bmatrix} + \begin{bmatrix} \text{Buffer stages added} \\ \text{to the critical path} \\ \text{of } u_k \text{ in } GB \end{bmatrix}$$

† We use the notation $\phi_l(u_i, u_k)$ to indicate an lth path from node u_i to node u_k. If node u_i is the input node u_0, then $\phi_l(u_0, u_k) \equiv \phi_l(u_k)$.

$$= w^c(u_k) + \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)} |B(e(i,j))|$$

where $|B(e(i,j))|$ is the weight or the number of computing stages in the buffer $B(e(i,j))$, $1 \le l \le m_k$ and $1 \le k \le n$. The first term in Eq. (2) is a constant and can be easily computed. The problem of finding all critical paths of u_k, $1 \le k \le n$, is known to be solvable by applying Bellman's equation with time complexity of $O(|N|^2)$ [13], where N is the number of computational nodes in the GW.

Since it is desirable to minimize the initial delay time of the pipeline so that it equals to $w^c(u_n)$, no buffers $B(e(i,j))$ should be assigned to the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n. We can state this fact in a lemma.

Lemma 1. The critical path $\phi_{l_n^*}(u_n)$ of the end node u_n is independent of the buffer stage variables.

Taking this into consideration and rewriting Eq. (2), we have

$$\sum_{B(e(i,j)) \in \phi_l(u_k)/\phi_{l_n^*}(u_n)} |B(e(i,j))| - \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)/\phi_{l_n^*}(u_n)} |B(e(i,j))| \quad (3)$$

$$= [\, w^c(u_k) - \sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) \,] = b(e,n)$$

where $b(e,n)$ is a computed integer constant, $|B(e(i,j))|$ are undetermined buffer stages, $1 \le l \le m_k$, $1 \le k \le n$, and the notation $\phi_l(u_k)/\phi_{l_n^*}(u_n)$ denotes set subtraction and is defined as $\phi_l(u_k)/\phi_{l_n^*}(u_n) = \phi_l(u_k) - (\phi_l(u_k) \cap \phi_{l_n^*}(u_n))$.

Equation (3) is a set of linear simultaneous equations and can be expressed in a matrix-vector form as $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a matrix introduced from the paths, $\mathbf{x}$ and $\mathbf{b}$ are unknown buffer stage vector and constant vector, respectively. The solution $\mathbf{x}$ is usually not unique, however, we can impose some restrictions on the problem to become an integer linear optimization problem. That is, we would like to minimize the total number of buffer stages in a balanced buffering graph GB,

Minimize the total number of buffer stages in GB

$$= \text{Min} \sum_{B(e(i,j)) \in GB/\phi_{l_n^*}(u_n)} |B(e(i,j))| \quad (4)$$

Subject to the equality constraints of

$$\sum_{B(e(i,j)) \in \phi_l(u_k)/\phi_{l_n^*}(u_n)} | B(e(i,j)) | - \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)/\phi_{l_n^*}(u_n)} | B(e(i,j)) | = b(e\ ,n) \tag{5}$$

and

$$| B(e(i,j)) | \geq 0, \text{ integer} \tag{6}$$

where $1 \leq l \leq m_k$ and $1 \leq k \leq n$. The above integer linear programming problem can be solved in pseudo-polynomial time [18].

In the above buffer assignment problem, the number of buffer stages are obtained from the solution of the integer linear programming problem and the buffers are placed on the edges in the buffering graph *GB* corresponding to the *GW* except the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n. In order to reduce the total number of buffer stage *variables* in the optimal buffer assignment problem, a useful equivalent transformation on a balanced buffering graph is introduced. A transformation of a balanced buffering graph *GB* with respect to a weighted ADFG *GW* by adjusting the position and amount of its buffering is said to be an equivalent transformation if the new transformed buffering graph GB' is also a balanced buffering graph (since a balanced buffering graph is not unique) with respect to the weighted ADFG *GW*. In general, the equivalent transformation has the following three properties:

(a) A buffer stage can be moved along a chain which is defined as a directed path in a buffering graph *GB* such that the incoming and outgoing edge for all nodes along the path is equal to one, except the starting and ending nodes of the chain.

(b) Two or more buffers on the same chain can be combined together to form a new buffer which has the same number of computing stages as the sum of these buffers.

(c) Combination of properties (a) and (b).

Based on the equivalent transformation of a balanced buffering graph *GB*, we can move the buffers along the chains of *GB* to multi-output nodes (or multi-input nodes). The new balanced buffering graph GB' has the same properties as the *GB*, with the buffers attached to the multi-output nodes (or multi-input nodes). We say that the new balanced buffering graph GB' is *normalized*. As an example, in Figure 2(a), paths $A-B-C-D$, $E-F-D$, and $E-G-D$ are chains. By combining the buffers along the chains and moving the resultant buffers to the output edges of the multi-output nodes A and E, we arrive at a new balanced buffering graph GB' as shown in Figure 2(b).

With the equivalent transformation on a balanced buffering graph, the optimal buffer assignment problem can be reformulated for the normalized balanced buffering graph instead of the balanced buffering graph. This, in effect, greatly reduces the total number of buffer stage variables because these variables are attached to multi-output (or multi-input) nodes. While constructing the integer linear programming formulation for the normalized balanced buffering graph for a weighted ADFG GW, it can be shown that many redundant integer linear constraint equations (in Eq. (5)) exist, making the optimization problem more difficult than it should be. The redundant integer linear constraint equations come from the path overlapping between two paths of two different multi-input nodes. They can be removed easily by recognizing the overlapping path (or common path) traversed by the different paths. A path decomposition technique is utilized to remove redundant integer linear constraint equations. Let $\phi_l(u_k)$ denote an l th path from the input node u_0 to a multi-input node u_k which passes through some other multi-input nodes. Among these multi-input nodes, a multi-input node u^* which is *nearest* to the node u_k is selected to decompose the path $\phi_l(u_k)$ into two sub-paths, that is, $\phi_l(u_k) = \phi_l(u^*) + \phi_l(u^*, u_k)$. Thus, the integer linear constraint equations of the path $\phi_l(u_k)$ with respect to the node u_k can be written as:

$$\sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u_k)} |B(e(i,j))| \qquad (7)$$

$$= \sum_{e(i,j) \in \phi_l(u^*)} w(e(i,j)) + \sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j))$$

$$+ \sum_{B(e(i,j)) \in \phi_l(u^*)} |B(e(i,j))| + \sum_{B(e(i,j)) \in \phi_l(u^*, u_k)} |B(e(i,j))|$$

where $1 \leq l \leq m_k$. Using Eq. (2) for the path $\phi_l(u^*)$ to the node u^*, Eq. (7) becomes

$$\sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u_k)} |B(e(i,j))|$$

$$= w^c(u^*) + \sum_{B(e(i,j)) \in \phi_{l^*}(u^*)} |B(e(i,j))| + \qquad (8)$$

$$\sum_{B(e(i,j)) \in \phi_l(u^*, u_k)} |B(e(i,j))| + \sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j))$$

Using the result of Lemma 1 and Eq. (3), Eq. (8) becomes

$$\sum_{B(e(i,j)) \in \phi_l(u^*, u_k)/\phi_{l^*_n}(u_n)} |B(e(i,j))| + \sum_{B(e(i,j)) \in \phi_{l^*}(u^*)/\phi_{l^*_n}(u_n)} |B(e(i,j))|$$

$$- \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)/\phi_{l_n^*}(u_n)} | B(e(i,j)) | \qquad (9)$$

$$= [\, w^c(u_k) - w^c(u^*) - \sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j)) \,]$$

With the above procedure for reducing redundant equations, the integer linear constraint equations for the normalized balanced buffering graph with respect to a weighted ADFG *GW* can be constructed according to the Procedure ILEG (Integer Linear Equation Generator) listed below.

Procedure ILEG (*GW* , *ILCE*(*GB*$'$)). This procedure generates integer linear constraint equations *ILCE*(*GB*$'$) for a normalized balanced buffering graph *GB*$'$ with respect to a given weighted ADFG *GW* with labeled nodes.

Input: A weighted ADFG *GW* with labeled nodes.

Output: A set of integer linear constraint equations, *ILCE*(*GB*$'$), for a normalized balanced buffering graph *GB*$'$ with respect to the given weighted ADFG *GW* with labeled nodes.

Step 1. [*Determine all critical paths*] Find all the critical paths $\phi_{l_k^*}(u_k)$ and the cost of each critical path $w^c(u_k)$ with respect to a multi-input node u_k, $1 \le k \le n$, by applying the Bellman's equation [13].

Step 2. [*Assign buffer stage variables*] Assign buffer stage variables to the output (or input) edges which are attached to multi-output (or multi-input) nodes, except for the output (or input) edges belonging to the critical path of the end node u_n. Output edges will be preferred if output and input edges are on the same chain. It is worth pointing out that a node may be both multi-input and multi-output node.

Step 3. [*Generate integer linear constraint equations*] For any path $\phi_l(u_k)$ with respect to a multi-input node u_k , $1 \le l \le m_k$, $1 \le k \le n$, if $\phi_l(u_k)$ does not pass through any other multi-input nodes, then we have

$$\sum_{B(e(i,j)) \in \phi_l(u_k)/\phi_{l_n^*}(u_n)} | B(e(i,j)) | - \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)/\phi_{l_n^*}(u_n)} | B(e(i,j)) | \qquad (10a)$$

$$= [\, w^c(u_k) - \sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) \,]$$

Otherwise, a multi-input node $u^* \in \phi_l(u_k)$ nearest to the node u_k is selected for path decomposition

$$\sum_{B(e(i,j)) \in \phi_l(u^*, u_k)/\phi_{l_n^*}(u_n)} |B(e(i,j))| + \sum_{B(e(i,j)) \in \phi_{l^*}(u^*)/\phi_{l_n^*}(u_n)} |B(e(i,j))|$$

$$- \sum_{B(e(i,j)) \in \phi_{l_k^*}(u_k)/\phi_{l_n^*}(u_n)} |B(e(i,j))| \qquad (10b)$$

$$= [\, w^c(u_k) - w^c(u^*) - \sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j)) \,] \ .$$

Note that the paths and their costs between two multi-input nodes may be found with time complexity $O(n^3)$ by using the path-finding algorithm [2].

Step 4. [*Output integer linear constraint equations*] Output the integer linear constraint equations from Eq. (10a) or Eq. (10b) and return.

END ILEG

Let us illustrate the above Procedure ILEG by an example. Figure 3(a) shows a weighted ADFG GW. We would like to obtain an optimal normalized balanced buffering graph GB' corresponding to the GW.

Step 1. Nodes G, J, K, and M are multi-input nodes. Then the critical path for
(a) Node G: $\phi_{l_G^*}(G) =$ Path $A-C-G$, $w^c(G) = 25$.
(b) Node J: $\phi_{l_J^*}(J) =$ Path $A-C-G-J$, $w^c(J) = 31$.
(c) Node K: $\phi_{l_k^*}(K) =$ Path $A-C-G-K$, $w^c(K) = 31$.
(d) Node M: $\phi_{l_M^*}(M) =$ Path $A-C-G-J-M$, $w^c(M) = 41$.

Step 2. Applying the buffer assignment rules, we obtain the normalized buffering graph as shown in Figure 3(b).

Step 3. The integer linear constraint equations are generated according to Eq. (10a) or Eq. (10b):
(a) The integer linear constraint equations generated with respect to the paths associated with node G:
(i) Path $A-B-E-G$: $|B_1| + |B_4| = (25-5-6-2) = 12$.
(ii) Path $A-B-G$: $|B_1| + |B_5| = (25-5-6) = 14$.
(b) The integer linear constraint equations generated with respect to the paths associated with node J:
(i) Path $A-B-F-J$: $|B_1| + |B_3| = (31-5-6-12) = 8$.
(c) The integer linear constraint equations generated with respect to the paths associated with node K:

(i) Path $A-D-H-K$: $|B_2| + |B_6| - |B_8| = (31-5-7-6) = 13$.

(ii) Path $A-D-I-K$: $|B_2| + |B_7| - |B_8| = (31-5-7-10) = 9$.

The above integer linear constraint equations have been generated according to Eq. (10a). The following case will show the integer linear constraint equations generated by using Eq. (10b).

(d) In generating the integer linear constraint equations with respect to the paths associated with node M, we select $u^* \equiv K$ as the multi-input node nearest to the node M and in the path $\phi_l(M)$ from the input node u_0 passing through the buffer $|B_7|$ to the node M. The path $\phi_l(M)$ can be decomposed into two sub-paths, that is, $\phi_l(M) = \phi_l(u^*) + \phi_l(u^*, M)$. According to Eq. (10b), we have $|B_8| + |B_9| = (41-31-8) = 2$.

Step 4. Applying the integer linear programming to minimize the total number of buffer stages, we have:

$$\text{Minimize } \sum_{i=1}^{9} |B_i|$$

subject to the constraints of the integer linear equations generated in *Step 3*. The optimization gives $|B_1| = 8$, $|B_2| = 9$, $|B_3| = 0$, $|B_4| = 4$, $|B_5| = 6$, $|B_6| = 4$, $|B_7| = 0$, $|B_8| = 0$, $|B_9| = 2$, and the total number of buffer stages is 33.

3. Formulation for Decomposition Approach

The previous section indicates that the optimal buffer assignment problem can be solved by formulating it as an integer linear optimization problem. If the task graph is simple, then the buffer assignment problem can be easily solved as illustrated in the above example. However, if the number of computational nodes in an ADFG is quite large, then integer linear constraint equations may grow tremendously, making the optimization problem more intractable. Thus, a systematic approach in reducing the computational difficulty in a large-scale integer linear optimization for the buffer assignment problem must be devised. A decomposition approach, which utilizes the critical path concept to decompose the task graph into a set of connected subgraphs from which the integer linear optimization technique can be used to solve the buffer assignment problem in each subgraph, will be addressed in this section.

Lemma 2. If a multi-input node $u_k \in \phi_{l_n^*}(u_n)$ and its critical path is $\phi_{l_k^*}(u_k)$, then $\phi_{l_k^*}(u_k) \subseteq \phi_{l_n^*}(u_n)$, that is, $\phi_{l_k^*}(u_k)$ is the path from the input node u_0 to the node u_k along the critical path $\phi_{l_n^*}(u_n)$, where u_n is the end node.

Proof: We shall prove Lemma 2 by contradiction. Assume that Lemma 2 is not true, then there exists a critical path $\overline{\phi}_{l_k^*}(u_k)$ for node u_k which is not a path segment of the critical path $\phi_{l_n^*}(u_n)$ for node u_n and its cost $w(\overline{\phi}_{l_k^*}(u_k))$ is greater than the cost of any other path from the input node u_0 to the node u_k, that is, we have $w(\overline{\phi}_{l_k^*}(u_k)) > w(\phi_{l_n^*}(u_0, u_k))$, where $\phi_{l_n^*}(u_0, u_k)$ is the path from the input node u_0 to the node u_k along the critical path $\phi_{l_n^*}(u_n)$. Let $\phi_{l_n^*}(u_k, u_n)$ be the remainder of the critical path of $\phi_{l_n^*}(u_n)$, that is, $\phi_{l_n^*}(u_n) = \phi_{l_n^*}(u_0, u_k) + \phi_{l_n^*}(u_k, u_n)$. A new path $\overline{\phi}_{l_k^*}(u_n)$ can be constructed by connecting the two subpaths, that is, $\overline{\phi}_{l_k^*}(u_n) = \overline{\phi}_{l_k^*}(u_k) + \phi_{l_n^*}(u_k, u_n)$. However, the cost of $\overline{\phi}_{l_k^*}(u_n)$ is greater than the cost of $\phi_{l_n^*}(u_n)$, that is, $w((\overline{\phi}_{l_k^*}(u_n)) = w((\overline{\phi}_{l_k^*}(u_k)) + w((\phi_{l_n^*}(u_k, u_n)) > w(\phi_{l_n^*}(u_0, u_k)) + w(\phi_{l_n^*}(u_k, u_n)) = w(\phi_{l_n^*}(u_n)) = w^c(u_n)$. This conclusion contradicts the definition of the critical path, thus, $\overline{\phi}_{l_k^*}(u_k) \equiv \phi_{l_k^*}(u_k) \subseteq \phi_{l_n^*}(u_n)$.

$\square$

Definition 5: Let $\overline{GW} = (\overline{V}, \overline{E}, \overline{W})$ be an undirected graph with $\overline{N} = |\overline{V}|$ and $\overline{M} = |\overline{E}|$. A connected component π_m of $\overline{GW}$ is a maximal connected subgraph, which is a connected subgraph that is not contained in any larger connected subgraphs.

Definition 6: A directed block $\vec{\pi}_m$ of a directed graph GW^* is a directed subgraph and its corresponding undirected subgraph π_m (i.e. $\pi_m = Undirect^{\dagger}(\vec{\pi}_m)$) is a connected component of the corresponding undirected graph $\overline{GW}$ ($\overline{GW} = Undirect(GW^*)$).

The problem of finding all the connected components of an undirected graph $\overline{GW}$ may be solved with the time complexity of $O(\overline{N} + \overline{M})$ by using the depth-first search algorithm SEARCH ($\overline{GW}$, π_m) in [3], where $\overline{GW}$ is an input undirected graph and π_m, $1 \leq m \leq m_{cc}$, are output connected components, where m_{cc} is the number of the directed blocks in the corresponding directed graph GW^* of $\overline{GW}$. The problem of finding the directed blocks $\vec{\pi}_m$ of a given directed graph GW^* may be solved by a modified depth-first search algorithm which is described in the Procedure DBS1 (Directed Blocks Searcher1) listed below:

Procedure DBS1 (GW^*, $\vec{\pi}_m$). This procedure finds all the directed blocks of a given directed graph GW^*.

Input: A directed graph GW^*.

Output: The directed blocks of GW^*, $\vec{\pi}_m$, $1 \leq m \leq m_{cc}$, where m_{cc} is the number of the directed blocks in GW^*.

†The notation $Undirect$ ($\vec{\pi}_m$) means taking the directed arrow of $\vec{\pi}_m$ out.

Step 1. [*Obtain the undirected graph of* GW^*] Let $\overline{GW} = Undirect(GW^*)$. That is, remove the directed arrow of GW^*.

Step 2. [*Determine undirected connected components of* $\overline{GW}$] Find all the undirected connected components, π_m, $1 \leq m \leq m_{cc}$, of $\overline{GW}$ by the depth-first search algorithm SEARCH $(\overline{GW}, \pi_m)$.

Step 3. [*Determine directed blocks*] Obtain all the directed blocks $\vec{\pi}_m$, $1 \leq m \leq m_{cc}$, by assigning the directed arrow back to π_m, $1 \leq m \leq m_{cc}$, according to the input directed graph GW^*.

Step 4. [*Output the directed blocks*] Output all the directed blocks $\vec{\pi}_m$, $1 \leq m \leq m_{cc}$.

END DBS1

The connected components π_m from the algorithm SEARCH $(\overline{GW}, \pi_m)$ and the directed blocks $\vec{\pi}_m$, $1 \leq m \leq m_{cc}$, from Procedure DBS1 will be used in our decomposition approach in obtaining a set of connected subgraphs from which the integer linear optimization technique can be applied to each subgraph to solve the buffer assignment problem. Our decomposition approach utilizes the critical path of the end node u_n, i.e., $\phi_{l_n^*}(u_n)$, as a cut set to partition an ADFG GW into several subgraphs. The procedure of graph partition and the determination of decomposed subgraphs (or directed blocks) is called graph decomposition [19]. The idea of the graph decomposition approach is first to take the critical path of the given directed graph out. This creates several edge disjoint subgraphs with some of the edges not connecting a pair of nodes because the nodes in the critical path are removed. In order to remedy this, nodes that are in the critical path $\phi_{l_n^*}(u_n)$ and are attached to two or more edges (incoming or outgoing) are called the decomposed nodes and denoted by $\tilde{u}_k$ (the kth decomposed node); each of these decomposed nodes $\tilde{u}_k$ will be "splitted" into several independent pseudo-nodes $\tilde{u}_k^i$, $1 \leq i \leq d_k$, which are labeled according to the attached edges from left to right, and the last pseudo-node $\tilde{u}_k^{d_k}$ is always assigned to the kth decomposed node in the critical path $\phi_{l_n^*}(u_n)$, where d_k is the number of independent pseudo-nodes for the kth decomposed node. Thus, a new directed graph GW^* containing splitted directed subgraphs of the ADFG GW can be obtained by removing the critical path $\phi_{l_n^*}(u_n)$ and "splitting" the decomposed nodes. That is, $GW^* = (GW/\phi_{l_n^*}(u_n)) \cup \{\text{labeled pseudo-nodes } \tilde{u}_k^i,\ 1 \leq i \leq (d_k - 1),\ 1 \leq k \leq k_{DN}\}$, where k_{DN} is the number of the decomposed nodes in GW. The determination of the directed blocks $\vec{\pi}_m$ of an ADFG GW when the critical path $\phi_{l_n^*}(u_n)$ is taken out is very similar to the Procedure DBS1 for finding the directed blocks $\vec{\pi}_m'$ of GW^*. The directed blocks $\vec{\pi}_m$ and $\vec{\pi}_m'$ are always equivalent except for the existence of the pseudo-nodes, $\tilde{u}_k^i$. The procedure for determining the directed blocks $\vec{\pi}_m$ of an ADFG

GW when the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n is taken out can be described in the following Procedure DBS2 (Directed Blocks Searcher2):

Procedure DBS2 ($GW, \vec{\pi}_m$). This procedure finds all the directed blocks of GW when its critical path $\phi_{l_n^*}(u_n)$ is taken out.

Input: A weighted graph GW and its critical path $\phi_{l_n^*}(u_n)$ of the end node u_n.

Output: The directed blocks, $\vec{\pi}_m$, $1 \le m \le m_{cc}$, of the ADFG GW when the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n is taken out.

Step 1. [*Remove the critical path in GW and label the decomposed nodes*]

(i) Obtain all the subgraphs from the ADFG GW by removing the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n and splitting the decomposed nodes $\tilde{u}_k$, $1 \le k \le k_{DN}$.

(ii) Label the independent pseudo-nodes of the decomposed node $\tilde{u}_k$, that is $\tilde{u}_k^i$, $1 \le i \le d_k$, and $\tilde{u}_k = \tilde{u}_k^1 \bigoplus \tilde{u}_k^2 \ldots \bigoplus \tilde{u}_k^{d_k}$, where $\bigoplus$ is the direct sum of the pseudo-nodes coming from the same decomposed node.

Step 2. [*Construct* GW^*] Construct a new directed graph GW^* which is the splitted directed subgraphs with labeled pseudo-nodes in *Step* 1

$$GW^* = (GW/\phi_{l_n^*}(u_n)) \cup (\bigcup_{k=1}^{k_{DN}} \bigcup_{i=1}^{(d_k-1)} \{\tilde{u}_k^i\}).$$

Step 3. [*Find the directed blocks of* GW^*] Apply the Procedure DBS1 to find the directed blocks $\vec{\pi}_m'$ of GW^*, that is, DSB1 ($GW^*, \vec{\pi}_m'$).

Step 4. [*Identify and merge pseudo-nodes in each directed block*] Determine the labeled pseudo-nodes which come from the same decomposed node and are in the same directed block $\vec{\pi}_m'$. These labeled pseudo-nodes will be merged into a big labeled pseudo-node by the direct sum operator $\bigoplus$.

Step 5. [*Determine and output the directed blocks* $\vec{\pi}_m$] Obtain $\vec{\pi}_m$ from $\vec{\pi}_m'$ by applying the pseudo-nodes merging procedure in *Step* 4 and output $\vec{\pi}_m$, $1 \le m \le m_{cc}$.

END DBS2

Using the Procedure DBS2 ($GW, \vec{\pi}_m$), we can obtain all the directed blocks of GW, $\vec{\pi}_m$, $1 \le m \le m_{cc}$. Furthermore, new subgraphs can be constructed from $\vec{\pi}_m$ and defined as $\vec{\pi}_m^+ = \vec{\pi}_m \cup_{DN} \phi_{l_n^*}(u_n)$, for $1 \le m \le m_{cc}$, where the operator $\cup_{DN}$ means performing the set union of $\vec{\pi}_m$ and $\phi_{l_n^*}(u_n)$ (except the pseudo-nodes) and the direct sum on the pseudo-nodes coming from the same decomposed nodes in $\vec{\pi}_m$ and

$\phi_{l_n^*}(u_n)$, simultaneously. These new subgraphs are called pseudo-connected components of the ADFG *GW* and will be used to decompose the buffer assignment problem into several small subproblems.

It has been shown in section 2 that the buffer stage variables in *GB* are determined from solving the associated integer linear constraint equations which are obtained from the Procedure ILEG. Let $\vec{\pi B}_m^+$ be a normalized balanced buffering graph for $\vec{\pi}_m^+$ and *ILCE* $(\vec{\pi B}_m^+)$ be the associated integer linear constraint equations which are obtained from the Procedure ILEG. Since an ADFG *GW* may have a large number of nodes, determining the buffer stage variables in *GB* from its large number of integer linear constraint equations may not be desirable. Since $GB = \bigcup_{DN\, m=1}^{m_{cc}} \vec{\pi B}_m^+$, we would like to use this fact to see whether solving the buffer stage variables in each $\vec{\pi B}_m^+$, $1 \le m \le m_{cc}$, separately and independently is equivalent to solving the buffer stage variables in *GB*. If this is true, then we have divided a large-scale integer linear optimization problem into m_{cc} smaller-scale subproblems, each of which can be easily solved. This is stated in Theorem 1.

Theorem 1. Let *GB* and $\vec{\pi B}_m^+$, $1 \le m \le m_{cc}$, be, respectively, the normalized balanced buffering graphs of *GW* and its pseudo-connected components $\vec{\pi}_m^+$, $1 \le m \le m_{cc}$. The buffer stage variables in *GB* can be determined from their associated integer linear constraint equations, *ILCE* $(\vec{\pi B}_m^+)$, $1 \le m \le m_{cc}$, separately and independently. Furthermore, the buffer stage variables determined from the integer linear constraint equations, *ILCE* $(\vec{\pi B}_{m1}^+)$, have no relations to the buffer stage variables determined from the equations, *ILCE* $(\vec{\pi B}_{m2}^+)$, where $m1 \ne m2$.

Proof: In order to prove the above theorem, we follow the procedure for constructing the associated integer linear constraint equations for *GB* and show how they can be replaced by *ILCE* $(\vec{\pi B}_m^+)$, $1 \le m \le m_{cc}$. For convenience, we assume there is a multi-input node u_k in both *GB* (or the corresponding *GW*) and $\vec{\pi B}_m^+$ (or the corresponding $\vec{\pi}_m^+$). $\vec{\pi B}_m^+$ is the mth pseudo-connected component of *GB*. Assume that the associated paths from the input node u_0 to the node u_k in *GB* (or *GW*) are $\phi_l(u_k)$, $1 \le l \le m_k$. Two cases are possible: (1) some of these paths pass through $\vec{\pi B}_m^+$ *only*, and (2) some of them pass through some other pseudo-connected components of *GB*. In case (1), because the paths in *GB* are also the paths in $\vec{\pi B}_m^+$, we will obtain the same resulting associated integer linear equations for the paths in *GB* and the paths in $\vec{\pi B}_m^+$. In case (2), the paths from the input node u_0 to the node u_k may pass through some other pseudo-connected components, but they must intersect the critical path $\phi_{l_n^*}(u_n)$ of the end node u_n at some nodes, and finally end at the node u_k in $\vec{\pi B}_m^+$. It has been shown previously that a multi-input node u^*, which is on the critical path $\phi_{l_n^*}(u_n)$ and nearest to the node u_k, can be selected to decompose the path into two subpaths, that is, $\phi_l(u_k) = \phi_l(u^*) + \phi_l(u^*, u_k)$, where $\phi_l(u^*)$ is the path from the input node u_0 to the node u^* and passes through some other pseudo-connected components, and the entire traversal of the path $\phi_l(u^*, u_k)$ is in the $\overrightarrow{\pi B}_m^+$. Thus, the associated integer linear equation for the path $\phi_l(u_k)$ in GB can be rewritten as follows:

$$\sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u_k)} |B(e(i,j))| \tag{11}$$
$$= \sum_{e(i,j) \in \phi_l(u^*)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u^*)} |B(e(i,j)|$$
$$+ \sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u^*, u_k)} |B(e(i,j))|$$

Using Lemma 1 and Eq. (2), the first two terms on the right hand side of Eq. (11) can be written as

$$\sum_{e(i,j) \in \phi_l(u^*)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u^*)} |B(e(i,j))| \tag{12}$$
$$= w^c(u^*) + \sum_{B(e(i,j)) \in \phi_{l^*}(u^*)} |B(e(i,j))|$$

Using the result of Lemma 2, the critical path to the node u^*, $\phi_{l^*}(u^*)$, is the path from the input node u_0 to the node u^* along the critical path $\phi_{l_n^*}(u_n)$, that is, $\phi_{l^*}(u^*) = \phi_{l_n^*}(u_0, u^*)$, which is independent of the buffer stage variables. Then Eq. (12) becomes

$$\sum_{e(i,j) \in \phi_l(u^*)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u^*)} |B(e(i,j))| = w^c(u^*) = \text{a constant} \tag{13}$$

Substituting Eq. (13) into Eq. (11), we have,

$$\sum_{e(i,j) \in \phi_l(u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u_k)} |B(e(i,j))| = w^c(u^*) + \tag{14}$$
$$\sum_{e(i,j) \in \phi_l(u^*, u_k)} w(e(i,j)) + \sum_{B(e(i,j)) \in \phi_l(u^*, u_k)} |B(e(i,j))|$$

Equation (14) indicates two things: First, the associated integer linear equations with respect to the node $u_k \in \overrightarrow{\pi B}_m^+$ depend on the buffer stage variables in $\overrightarrow{\pi B}_m^+$ and are independent of the buffer stage variables in the other pseudo-connected components because $\phi_l(u^*, u_k) \in \overrightarrow{\pi B}_m^+$. Second, Eq. (14) can be generated and replaced by a path in $\overrightarrow{\pi B}_m^+$, that is, the path travels from the input node u_0 to the node u^*

along the critical path $\phi_{l_n^*}(u_n)$, then from node u^* to node u_k along the path $\phi_l(u^*, u_k)$ in $\overrightarrow{\pi B}_m^+$. So, for any multi-input node u_k belonging to GB and $\overrightarrow{\pi B}_m^+$, it has been shown that the associated integer linear equation system for node u_k in GB can be replaced by the associated integer linear equation system for node u_k in $\overrightarrow{\pi B}_m^+$. In other words, the associated integer linear equation system for GB, i.e., $ILCE\,(GB)$, can be replaced by the associated integer linear equation systems for $\overrightarrow{\pi B}_m^+$, i.e., $ILCE\,(\overrightarrow{\pi B}_m^+)$, $1 \leq m \leq m_{cc}$.

$\square$

Using the results from Theorem 1 and based on the fact that $GB = \bigcup_{DN\, m=1}^{m_{cc}} \overrightarrow{\pi B}_m^+$, $\sum_{B(e(i,j)) \in GB} |\, B(e(i,j))\, |$ becomes $\sum_{m=1}^{m_{cc}} \sum_{B(e(i,j)) \in \overrightarrow{\pi B}_m^+} |\, B(e(i,j))\, |$, and the integer linear optimization problem in Eqs. (4) and (5) can be rewritten as follows:

$$\text{Min} \sum_{m=1}^{m_{cc}} \sum_{B(e(i,j)) \in \overrightarrow{\pi B}_m^+} |\, B(e(i,j))\, | \tag{15}$$

subject to the associated integer linear equation systems $ILCE\,(\overrightarrow{\pi B}_m^+)$, $1 \leq m \leq m_{cc}$. Because the buffer stage variables in different pseudo-connected components of GB are independent, Eq. (15) can be decomposed into the following subproblems:

For each $m = 1, 2, \cdots, m_{cc}$:

$$\text{Min} \sum_{B(e(i,j)) \in \overrightarrow{\pi B}_m^+} |\, B(e(i,j))\, | \tag{16}$$

subject to the associated integer linear equation system $ILCE\,(\overrightarrow{\pi B}_m^+)$.

This graph decomposition approach provides us with a technique to divide a large-scale integer linear optimization problem into a number of smaller-scale subproblems (m_{cc} subproblems), each of which can be easily solved in pseudo-polynomial time.

Let us apply the above decomposition approach to solve the same buffer assignment problem in section 2.

Step 1. (a) Decompose the ADFG GW in Figure 3(a) into subgraphs by removing the critical path $\phi_{l_M^*}(M)$ of the end node M.

(b) Label the pseudo-nodes of the decomposed nodes A, G, J, M, that is, $\{A_1, A_2, A_3\}$, $\{G_1, G_2, G_3, G_4\}$, $\{J_1, J_2\}$, and $\{M_1, M_2\}$.

(c) Construct $GW^* = (GW / \phi_{l_M^*}(M)) \cup \{A_1, A_2, G_1, G_2, G_3, J_1, M_1\}$. Note that pseudo-nodes A_3, G_4, J_2, and M_2 are attached to the critical path $\phi_{l_M^*}(M)$. GW^* , $\phi_{l_M^*}(M)$, and the labeled pseudo-nodes are shown in Figure 4. (4(a), 4(b), and 4(c))

Step 2. This step is the same as the Procedure DBS2 $(GW^*, \vec{\pi}_m)$.

(a) Apply the directed block search Procedure DBS1 $(GW^*, \vec{\pi}'_m)$ to find the $\vec{\pi}'_m$, $1 \leq m \leq 2$, in GW^*. These directed blocks are shown in Figure 4 (4(a) and 4(b)).

(b) Merge the labeled pseudo-nodes that come from the same decomposed node and are in the same directed block $\vec{\pi}'_m$ into a big labeled pseudo-node by the direct sum operator. For example, G_1 and G_2 are the labeled pseudo-nodes coming from the decomposed node G in $\vec{\pi}'_1$, and will be merged into $G_{1,2} = G_1 \oplus G_2$.

(c) Obtain $\vec{\pi}_m$ from $\vec{\pi}'_m$, $m = 1, 2$, by applying the pseudo-nodes merging procedure. The directed block $\vec{\pi}_1$ is shown in Figure 4(d).

Step 3. Let $\vec{\pi}^+_m = \vec{\pi}_m \cup_{DN} \phi_{l^*_M}(M)$, $1 \leq m \leq 2$, which are the pseudo-connected components of GW. $\vec{\pi}^+_1$ and $\vec{\pi}^+_2$ are shown in Figure 4 (4(e) and 4(f) respectively).

Step 4. The corresponding normalized balanced buffering graphs GB and $\vec{\pi B}^+_m$ can be easily obtained by the buffer assignment rules and have the same graph structure as GW and $\vec{\pi}^+_m$, respectively. The buffer stage variables $B^1_1, B^1_2, B^1_3, B^1_4$ in $\vec{\pi B}^+_1$ as shown in Figure 4(g) and $B^2_1, B^2_2, B^2_3, B^2_4, B^2_5$ in $\vec{\pi B}^+_2$ as shown in Figure 4(h) correspond to the buffer stage variables B_1, B_3, B_4, B_5 and B_2, B_6, B_7, B_8, B_9 in GB as shown in Figure 3(b), respectively.

Step 5. Generate the associated integer linear equations system for $\vec{\pi B}^+_1$ and $\vec{\pi B}^+_2$ by the Procedure ILEG, that is, $ILCE(\vec{\pi B}^+_1)$ and $ILCE(\vec{\pi B}^+_2)$ as follows:

$$ILCE(\vec{\pi B}^+_1): \begin{array}{l} |B^1_1| + |B^1_2| = 8 \\ |B^1_1| + |B^1_3| = 12 \\ |B^1_1| + |B^1_4| = 14 \end{array}, \quad ILCE(\vec{\pi B}^+_2): \begin{array}{l} |B^2_1| + |B^2_2| - |B^2_4| = 13 \\ |B^2_1| + |B^2_3| - |B^2_4| = 9 \\ |B^2_4| + |B^2_5| = 2 \end{array}$$

It has stated in *Step* 4 that $B^1_1 \equiv B_1$, $B^1_2 \equiv B_3$, $B^1_3 \equiv B_4$, $B^1_4 \equiv B_5$ and $B^2_1 \equiv B_2$, $B^2_2 \equiv B_6$, $B^2_3 \equiv B_7$, $B^2_4 \equiv B_8$, $B^2_5 \equiv B_9$. Thus, $ILCE(\vec{\pi B}^+_1)$ and $ILCE(\vec{\pi B}^+_2)$ become:

$$ILCE(\vec{\pi B}^+_1): \begin{array}{l} |B_1| + |B_3| = 8 \\ |B_1| + |B_4| = 12 \\ |B_1| + |B_5| = 14 \end{array}, \quad ILCE(\vec{\pi B}^+_2): \begin{array}{l} |B_2| + |B_6| - |B_8| = 13 \\ |B_2| + |B_7| - |B_8| = 9 \\ |B_8| + |B_9| = 2 \end{array}$$

Step 6. The integer linear programming problem for GB can be solved by two separated subproblems:

(1) Min $\sum_{B_i \in \vec{\pi B}_1^+} |B_i| \Longrightarrow$ Min $[\,|B_1| + |B_3| + |B_4| + |B_5|\,]$
subject to *ILCE* $(\vec{\pi B}_1^+)$ (found in *Step* 5).

(2) Min $\sum_{B_i \in \vec{\pi B}_2^+} |B_i| \Longrightarrow$ Min $[\,|B_2| + |B_6| + |B_7| + |B_8| + |B_9|\,]$
subject to *ILCE* $(\vec{\pi B}_2^+)$ (found in *Step* 5).

The optimization of subproblem (1) yields $|B_1| = 8$, $|B_3| = 0$, $|B_4| = 4$, $|B_5| = 6$, and the optimization of subproblem (2) gives $|B_2| = 9$, $|B_6| = 4$, $|B_7| = 0$, $|B_8| = 0$, $|B_9| = 2$. The results and solution are the same as given in the example in section 2, but the optimization is much faster and simplier.

4. Application to CORDIC Pipeline for Robot Inverse Kinematic Solution

Let us apply the above decomposition approach to solving the buffer assignment problem of a larger problem — balancing a CORDIC-based pipeline architecture for computing the robot inverse kinematic position solution [15]. The task of computing the joint solution of a PUMA-like robot manipulator is shown in Figure 5. The nodes in Figure 5 represent CORDIC processors. The objective is to balance this task graph to achieve maximum pipelining [15]. By using the Procedure DBS2 $(GW, \vec{\pi}_m)$, where GW is the directed task graph shown in Figure 5, 16 directed blocks, $\vec{\pi}_m$, $1 \le m \le 16$, in GW are obtained. From these directed blocks, we can obtain the 16 pseudo-connected components, $\vec{\pi}_m^+$, $1 \le m \le 16$. The corresponding normalized balanced buffering graph GB for GW and the 16 pseudo-connected components in GB, $\vec{\pi B}_m^+$, $1 \le m \le 16$, can be created. The associated integer linear equation systems for $\vec{\pi B}_m^+$, $1 \le m \le 16$, are obtained as follows:

(a) *ILCE* $(\vec{\pi B}_1^+)$: $|B_4| = 3$.

(b) *ILCE* $(\vec{\pi B}_2^+)$: $|B_{16}| = 1$.

(c) *ILCE* $(\vec{\pi B}_3^+)$: $|B_5| = 3$.

(d) *ILCE* $(\vec{\pi B}_4^+)$: $|B_1| - |B_{17}| = 3$, $|B_2| - |B_{17}| = 3$,
$|B_{17}| + |B_{19}| = 10$, $|B_{17}| + |B_{20}| = 9$, $|B_{17}| + |B_{21}| = 8$.

(e) *ILCE* $(\vec{\pi B}_5^+)$: $|B_3| = 2$.

(f) *ILCE* $(\vec{\pi B}_6^+)$: $|B_{22}| = 15$.

(g) *ILCE* $(\vec{\pi B}_7^+)$: $|B_6| + |B_{10}| = 5$, $|B_6| + |B_{11}| + |B_{12}| = 8$,
$|B_6| + |B_{11}| + |B_{13}| + |B_{15}| = 10$,
$|B_6| + |B_{11}| + |B_{13}| + |B_{14}| - |B_{30}| - |B_{33}| = 10$,
$|B_{23}| - |B_{30}| = 5$, $|B_{30}| + |B_{33}| + |B_{34}| = 0$.

(h) $ILCE(\overrightarrow{\pi B}_8^+)$: $|B_{24}| = 4.$

(i) $ILCE(\overrightarrow{\pi B}_9^+)$: $|B_{25}| = 1.$

(j) $ILCE(\overrightarrow{\pi B}_{10}^+)$: $|B_{26}| = 5.$

(k) $ILCE(\overrightarrow{\pi B}_{11}^+)$: $|B_7| - |B_{36}| = 12$, $|B_8| - |B_{18}| = 3$, $|B_9| - |B_{18}| = 3,$
$|B_{18}| + |B_{27}| - |B_{36}| = 8$, $|B_{18}| + |B_{28}| - |B_{41}| = 10,$
$|B_{36}| + |B_{39}| = 3$, $|B_{36}| + |B_{40}| - |B_{41}| = 1,$
$|B_{41}| + |B_{44}| = 1$, $|B_{41}| + |B_{45}| = 2.$

(l) $ILCE(\overrightarrow{\pi B}_{12}^+)$: $|B_{29}| + |B_{31}| - |B_{35}| = 4$, $|B_{29}| + |B_{32}| = 10,$
$|B_{35}| + |B_{46}| = 5.$

(m) $ILCE(\overrightarrow{\pi B}_{13}^+)$: $|B_{37}| = 1.$

(n) $ILCE(\overrightarrow{\pi B}_{14}^+)$: $|B_{38}| = 2.$

(o) $ILCE(\overrightarrow{\pi B}_{15}^+)$: $|B_{42}| = 4.$

(p) $ILCE(\overrightarrow{\pi B}_{16}^+)$: $|B_{43}| = 2.$

Each of the above integer linear equation systems, $ILCE(\overrightarrow{\pi B}_m^+)$, $1 \leq m \leq 16$, corresponds to an integer linear optimization subproblem. For example, the integer linear optimization subproblem for $ILCE(\overrightarrow{\pi B}_7^+)$ is to:

$$\text{Min} \sum_{B_i \in \overrightarrow{\pi B}_7^+} |B_i|$$

subject to the associated integer linear equation system $ILCE(\overrightarrow{\pi B}_7^+)$. That is,

$$\text{Min} \ (|B_6| + |B_{10}| + |B_{11}| + B_{12}| + |B_{13}| + |B_{14}| + |B_{15}| + |B_{23}|$$
$$+ |B_{30}| + |B_{33}| + |B_{34}|)$$

subject to the associated integer linear equation system $ILCE(\overrightarrow{\pi B}_7^+)$. The optimization of this subproblem gives $|B_6| = 5$, $|B_{10}| = 0$, $|B_{11}| = 3$, $|B_{12}| = 0$, $|B_{13}| = 2$, $|B_{14}| = |B_{15}| = 0$, $|B_{23}| = 5$, $|B_{30}| = 0$, $|B_{33}| = |B_{34}| = 0$, and $\sum_{B_i \in \overrightarrow{\pi B}_7^+} |B_i| = 15$ delay buffers.

Similarly, the integer linear optimization subproblems for $ILCE(\overrightarrow{\pi B}_4^+)$, $ILCE(\overrightarrow{\pi B}_{11}^+)$, and $ILCE(\overrightarrow{\pi B}_{12}^+)$ give the optimization solutions $\{|B_1| = |B_2| = 3$, $|B_{17}| = 0$, $|B_{19}| = 10$, $|B_{20}| = 9$, $|B_{21}| = 8\}$, $\{|B_7| = 12$, $|B_8| = |B_9| = 3$, $|B_{18}| = 0$, $|B_{27}| = 8$, $|B_{28}| = 10$, $|B_{36}| = 0$, $|B_{39}| = 3$, $|B_{40}| = 1$, $|B_{41}| = 0$, $|B_{44}| = 1$, $|B_{45}| = 2\}$, and $\{|B_{29}| = 4$, $|B_{31}| = 0$, $|B_{32}| = 6$, $|B_{35}| = 0$, $|B_{46}| = 5\}$, respectively. The numbers of delay buffers in these subproblems are, respectively, $\sum_{B_i \in \overrightarrow{\pi B}_4^+} |B_i| = 33$, $\sum_{B_i \in \overrightarrow{\pi B}_{11}^+} |B_i| = 43$, and

$\sum_{B_i \in \overrightarrow{\pi B}_{12}^+} |B_i| = 15.$

The optimization solution for all the integer linear optimization subproblems yields a total of 159 buffer stages which agree with the solution given in [15]. This example shows that the graph decomposition approach can solve a large-scale buffer assignment problem by decomposing it into a set of smaller subproblems, each of which can be solved separately as an integer linear optimization problem. The initial delay time of the CORDIC pipeline is the cost of the critical path from the input node to the end node and is equal to 18 basic time units. For a commercial CMOS CORDIC processor [7], a reasonable stage latency is 40 μs. Thus, the initial delay time of the pipeline is equal to 720 μs. The balanced ADFG can be realized with 25 CORDIC processors and 159 buffer stages. The resultant maximum pipelined CORDIC architecture has a pipelined time of one basic time unit or 40 μs. The number of buffer stages can be further reduced if a special buffering device, tapped-delay-line buffer, is used [15].

5. Conclusion

An efficient graph decomposition technique which provides a systematic approach in solving the optimal buffer assignment problem of a large-scale ADFG has been presented and discussed. The optimal buffer assignment problem is formulated as an integer linear programming problem. The construction of integer linear constraint equations in a large-scale ADFG reveals the existence of many redundant integer linear constraint equations, making the optimization more intractable. The redundant integer linear constraint equations come from the path overlapping between two paths of two different multi-input nodes. The proposed graph decomposition approach utilizes the critical path concept to decompose an ADFG into a set of connected subgraphs from which the integer linear optimization technique can be used to solve the buffer assignment problem in each subgraph. Thus, a large-scale integer linear optimization problem is divided into a number of smaller-scale subproblems which can be easily solved in computers in pseudo-polynomial time. The proposed graph decomposition technique is illustrated by two examples and its efficiency and advantages can be seen in the example for balancing a CORDIC pipeline for computing the robot inverse kinematic position solution.

6. References

[1] H. M. Ahmed, J. M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer*, vol. 15, no. 1, pp. 65-82, Jan. 1982.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, pp. 195-199.

[3] S. Baase, *Computer Algorithms: Introduction to Design and Analysis,* Addison-Wesley, 1978, pp. 145-148.

[4] J. D. Brock and L. B. Montz, "Translation and Optimization of Data Flow Programs," *Proc. of 1979 Int'l. Conf. on Parallel Processing,* pp. 46-54, Aug. 1979.

[5] J. B. Dennis and R. G. Gao, "Maximum Pipelining of Array Operations on Static Data Flow Machine," *Proc. of 1983 Int'l. Conf. on Parallel Processing,* pp. 331-334, Aug. 1983.

[6] K. S. Fu (editor), *VLSI for Pattern Recognition and Image Processing,* Springer-Verlag, 1984.

[7] G. L. Haviland and A. A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. Comput.*, vol. C-29, no. 2, pp. 8- , Feb. 1980.

[8] F. H. Hsu, H. T. Kung, T. Nishizawa, and A Sussman, "LINC: The Link and Interconnection Chip," Department of Computer Science, Carnegie-Mellon University, 1984.

[9] K. Hwang and Z. Xu, "Multipipeline Networking for Fast Evaluation of Vector Compound Functions," *Proc. of 1986 Int'l Conf. on Parallel Processing,* pp. 495-502, August 1986.

[10] H. T. Kung, "Why Systolic Architectures?" *IEEE Computer,* pp. 37-46, Jan. 1982.

[11] H. T. Kung and M. Lam, "Wafer-Scale Integration and Two-level Pipelined Implementation of Systolic Arrays," *J. of Parallel and Distributed Computing,* vol. 1, no. 1, Sept. 1984, pp. 32-63.

[12] S. Y. Kung, H. J. Whitehouse, and T. Kailath, (editors), *VLSI and Modern Signal Processing,* Prentice-Hall, 1985.

[13] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[14] C. S. G. Lee and P. R. Chang, "Efficient Parallel Algorithm for Inverse Dynamics Computation," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-16, no. 4, pp. 532-542, July/August 1986.

[15] C. S. G. Lee and P. R. Chang, "A Maximum Pipelined CORDIC Architecture for Robot Inverse Kinematic Position Computation," Technical Report TR-EE 86-5, School of Electrical Engineering, Purdue University, January 1986.
Also to appear in *IEEE J. of Robotics and Automation.*

[16] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *J. VLSI and Computer Systems,* vol. 1, 1983, pp. 41-68.

[17] C. Mead and L. Conway, *Introduction to VLSI systems,* Addison-Wesley, 1980.

[18] C. H. Papadimitrious and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[19] J. A. Starzyk and A. Konczykowska, "Flowgraph Analysis of Large Electronic Networks," *IEEE Trans. on Circuits and Systems,* vol. CAS-33, pp. 302-315, March, 1986.

[20] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers,* vol. EC-8, no. 3, pp. 330-334, Sept. 1959.

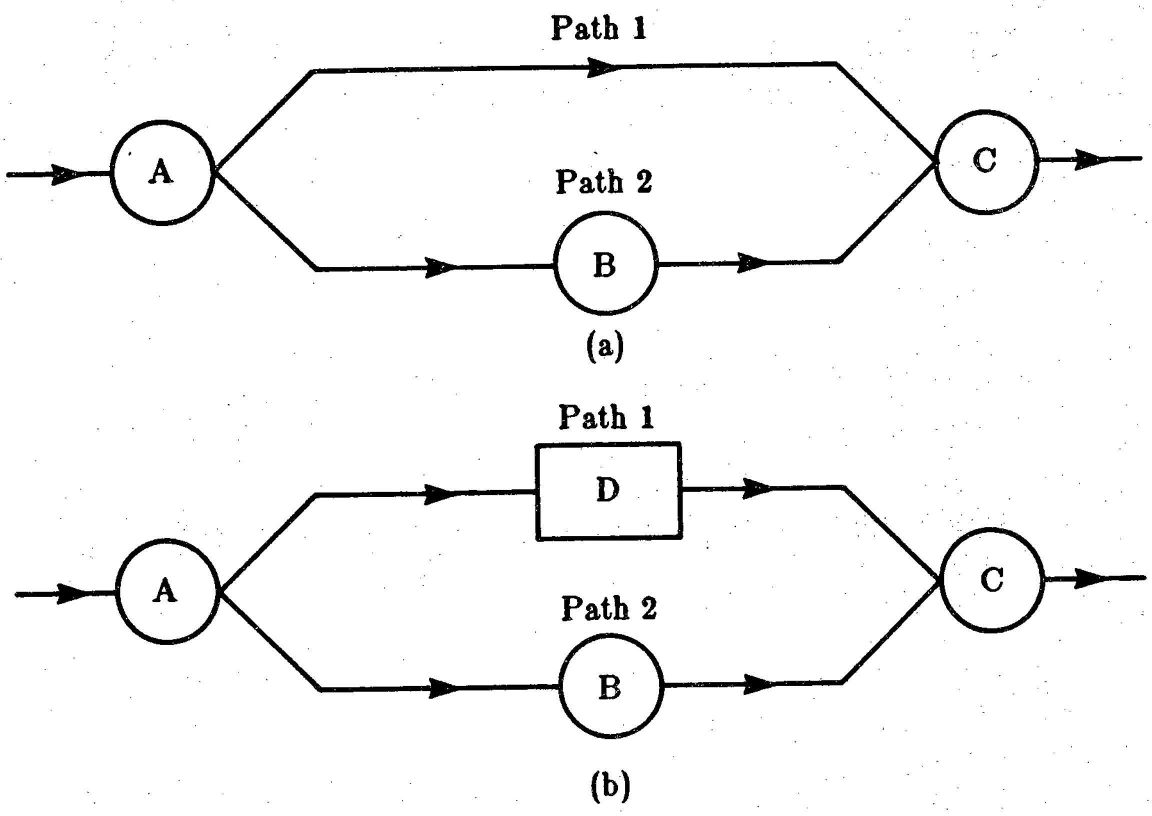

Figure 1 An Example for Inserting Delay Buffer Stages

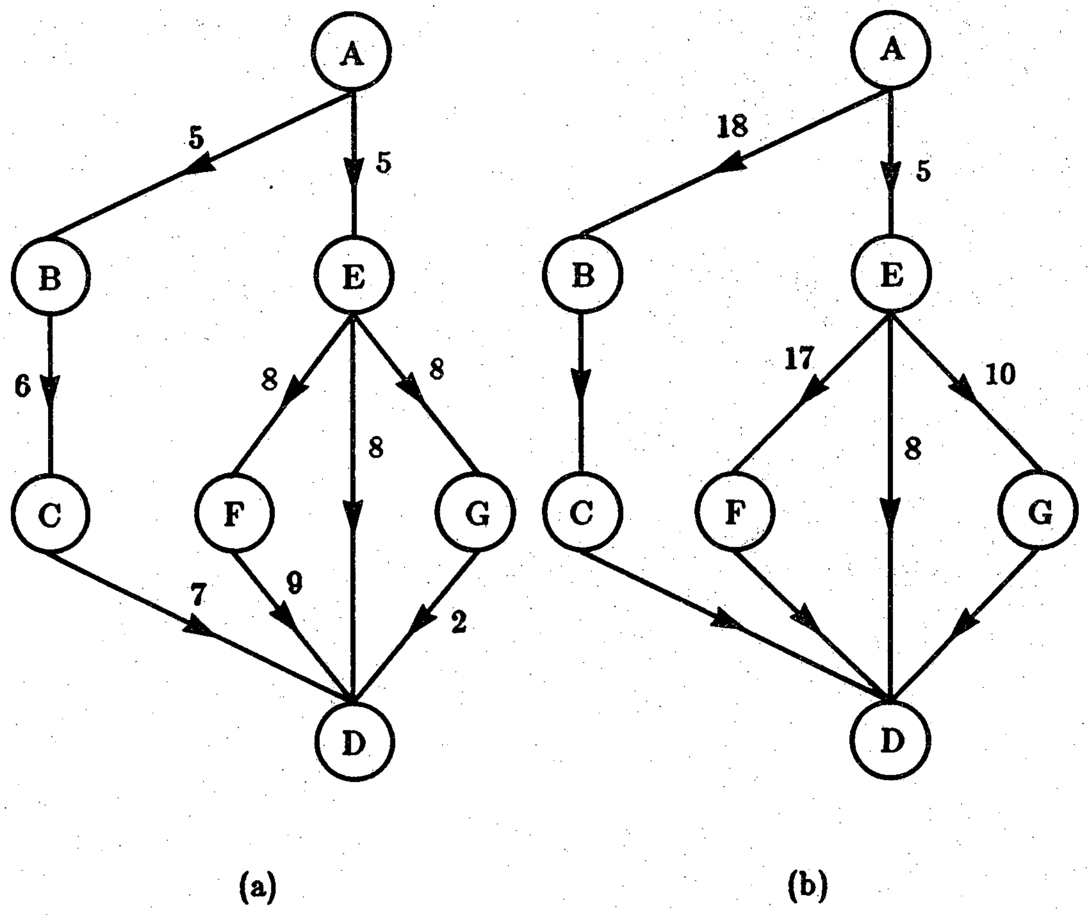

Figure 2 Normalization of a Balanced Buffering Graph

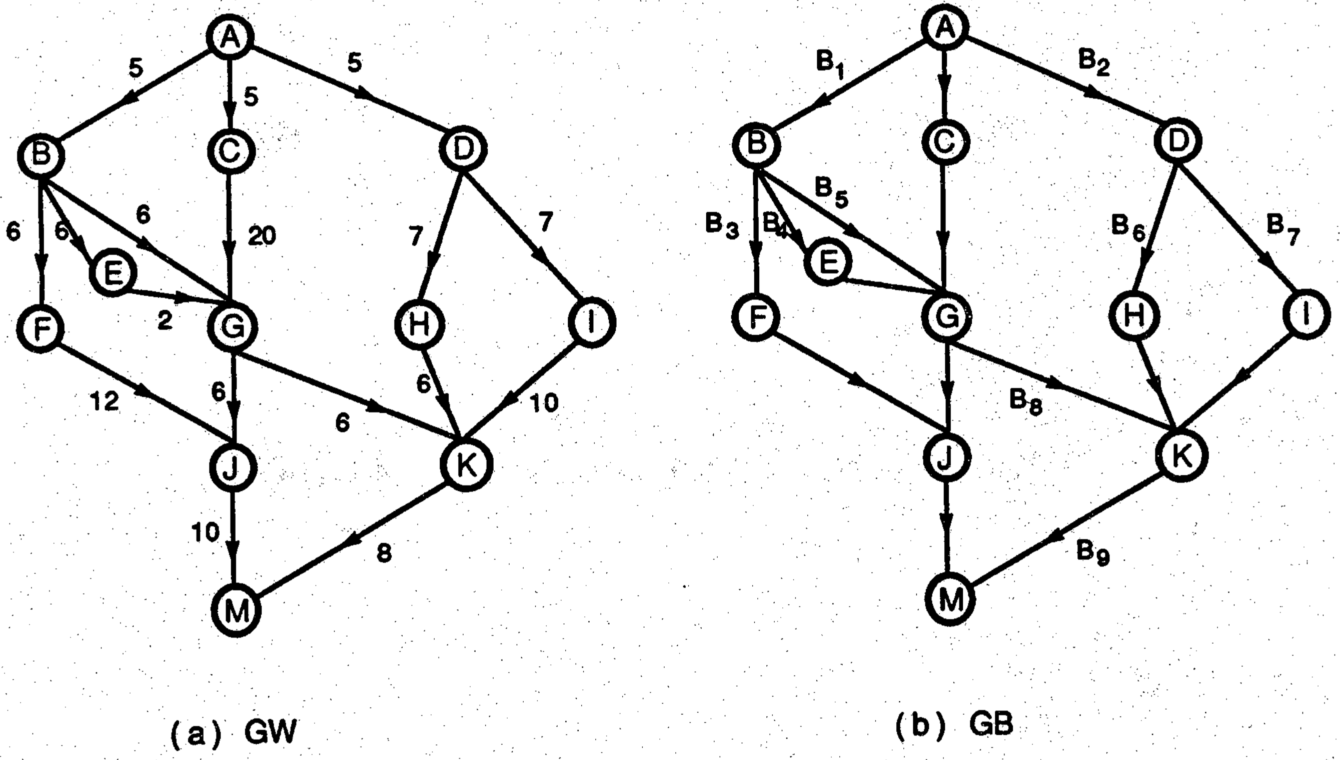

Figure 3 An Example for Buffer Assignment Problem

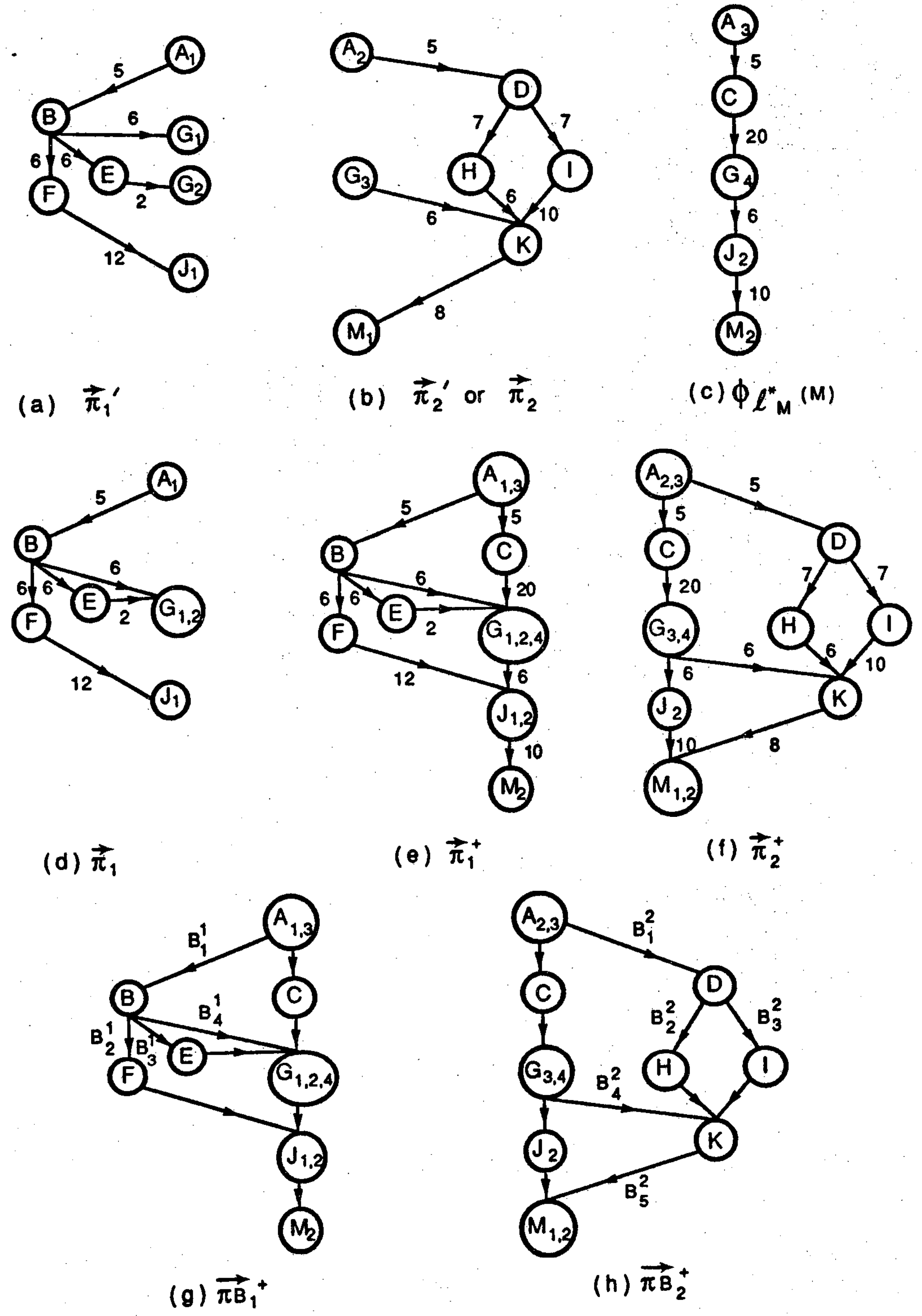
(a) $\vec{\pi}_1'$
(b) $\vec{\pi}_2'$ or $\vec{\pi}_2$
(c) $\phi_{\ell^*_M}(M)$
(d) $\vec{\pi}_1$
(e) $\vec{\pi}_1^+$
(f) $\vec{\pi}_2^+$
(g) $\overrightarrow{\pi B_1}^+$
(h) $\overrightarrow{\pi B_2}^+$

Figure 4 Graph Decomposition of the Example in Figure 3

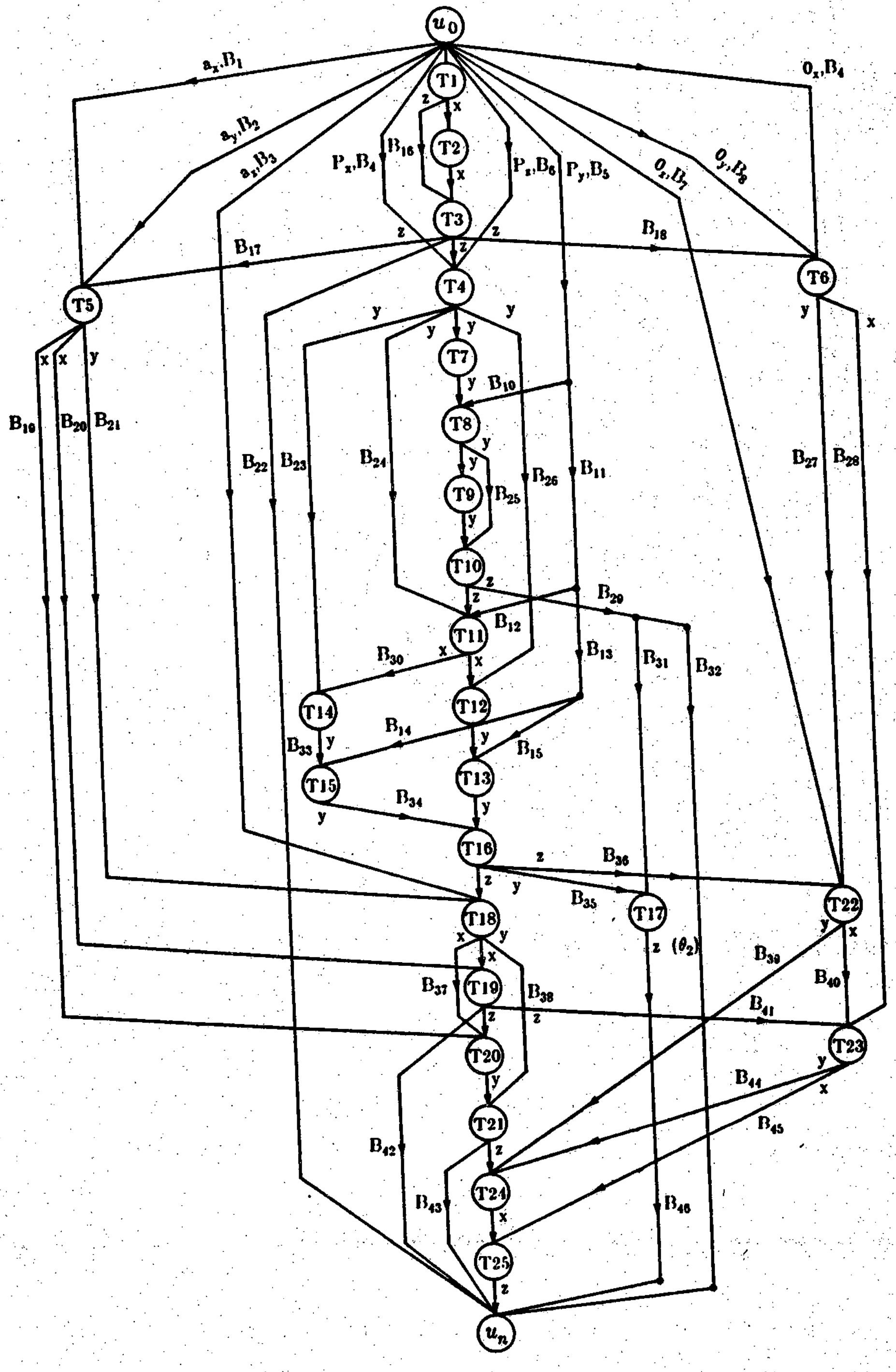

Figure 5 An ADFG for a PUMA-Type Robot Inverse Kinematic Position Solution