A Technique to Derive Time Costs of Data Flow Programs

Bin Qin, Reda A. Ammar and Howard A. Sholl

U-155, Computer Science & Engineering Department
The University of Connecticut
Storrs, CT 06268

Abstract

We present a technique in this paper to analyze the time cost behavior of data flow programs that are loop-free or contain non-nested loops that meet certain constraints. It is assumed the underlying computer system is a dynamic data flow computer which has sufficient processors to support the execution of data flow programs. The token model is modified in this paper to describe both the execution and time cost behavior. The technique is similar to some flow analysis techniques and graph reduction techniques. In analyzing the time cost of a data flow program, the technique first determines the number of tokens that pass through each edge in the program. All nodes that are neither input nodes nor output nodes are then deleted. Every time a node or an edge is deleted, the time cost associated with it is properly distributed within the program. Since the resultant program only contains edges that connect input nodes and output nodes, the time cost of the program can be easily obtained.

Two examples are given in the paper to show the use of the technique.

1. Introduction

The data flow approach provides an alternative to achieve parallel processing. It has been recognized that the data flow approach provides high degree of parallelism, freedom from side effects, locality of effect and single assignment rule [1]. On the other hand, it has been found the data flow approach causes excessive pipeline and tends to waste memory space [1, 2]. Research work has been done to evaluate the performance of the data flow approach [3, 4]. Most of the work concentrates on the high level performance, i.e., the system performance.

In this paper, we consider the performance of the data flow approach at a rather low level. That is, we consider the performance behavior of data flow programs, where the time cost is defined as the performance measure. Given a user data flow program, our goal is to predict its time cost without actually executing it.

It is assumed in our work that the underlying system is a dynamic data flow computer [1] which has sufficient processors to support the execution of data flow programs. It is also assumed the user’s data flow program is loop-free or contains non-nested loops that meet certain constraints (refer to Section 4). A deterministic model based on the token model [5] is used to describe both the execution and time cost behavior. In the model, each node is associated with a time cost that is equal to the time required to perform the specified operations; each edge is associated with a flow that indicates the number of tokens passing through the edge, and a time cost that is equal to the time needed to pass a token from one end of the edge to the other. A technique based on the model is developed to analyze the time cost behavior of user data flow programs. This technique is similar to some flow analysis techniques and graph reduction techniques [6, 7, 8, 9]. Given a user data flow program, the technique first determines the number of tokens that pass through each edge in the program. All nodes that are neither input nodes nor output nodes are then deleted from the program. Once a node and the edges associated with the node are deleted, their time costs are properly distributed within the program. In this way, the time cost information is retained. After the removal of all the nodes that are neither input nodes nor output nodes, the time cost of the user data flow program can be easily derived.

This paper is organized in the following way. Execution and time cost modelings are first described in Sections 2 and 3. The technique to derive the time cost is then discussed in Section 4. An example is shown in Section 5 to illustrate the time cost analysis process. Conclusions are finally given in Section 6.

2. Execution Modeling

The model used in this paper is based on the token model [5]. It is assumed a data flow program contains 8 different types of nodes as illustrated in Figure 1 (the weights of edges in Figure 1 will be explained in Section 3). An input node provides input to a program, an output node holds some result generated by a program, an operation node performs a given operation, an and node merges parallel data paths into a single
one, an or node serves as a junction point, a fork node splits a data path into several parallel paths, and a select or a distribute node checks some conditions.

Tokens are used in the model as the carriers of data. It is assumed an input node will provide as many tokens as needed for the execution of a program. The execution of a program starts when tokens start to move out of input nodes. When a token arrives at an output node, it is immediately absorbed by the output node. When an operation node receives a token from its incoming edge, it performs the specified operations and then puts a token on its outgoing edge. When an and node receives one token from each of its incoming edges, it puts a token on its outgoing edge. When an or node receives a token from one of its incoming edges, it moves the token to its outgoing edge. When a fork node receives a token from its incoming edge, it puts one token on each of its outgoing edges. When a select node receives a token (control token) from the incoming edge marked with e, depending on the value of the control token (F or T), it moves a token from one of its incoming edges marked with F and T to its outgoing edge. When a distribute node receives a control token from the incoming edge marked with e, (refer to Figure 1.f), depending on the value of the control token, the token coming from the incoming edge marked with e is moved to one of the outgoing edges marked with T and F. When no node in the program can be activated and no token is on its way to an output node, the execution of a data flow program is terminated.

3. Time Cost Modeling

In a data flow graph, each node except the input node and output node is associated with a time cost that is equal to the time to perform the operations specified by the node; and each edge is also associated with a time cost that is equal to the time to transfer a token from the start node of the edge to the end node of the edge. In this paper, we assume the time cost of a node is the same every time it is activated; and the time cost of an edge is the same every time a token passes through the edge. Although these assumptions are reasonable, one should keep in mind that in general, a node may have different time cost every time it is activated (for example, for a node that performs multiplication, 0*0 and 987*329 may have different time costs); and an edge may have different time cost every time a token passes through the edge (for example, the time of transferring a character string depends on the length of the string).

It is assumed in this paper that no redundant tokens are provided to any node. Under such an assumption, flows in a data flow program must have the following properties.

For the operation node in Figure 1.c,

\[ e_i = e_o \]

For the and node in Figure 1.d,

\[ e_{11} = \cdots = e_{1n} = e_o \]
For the or node in Figure 1.e,
\[ \sum_{j=1}^{n} e_{ij} = e_{o} \]
For the fork node in Figure 1.f,
\[ e_{1} = e_{o} = \cdots = e_{on} \]
For the select node in Figure 1.g,
\[ e_{s} + e_{f} = e_{s} = e_{o} \]
For the distribute node in Figure 1.h,
\[ e_{l} = e_{s} = e_{f} = e_{o} \]

When a node with time cost \( C \) receives all the required tokens, it starts to perform the specified operations; and after \( C \) amount of time, tokens leave the node through the outgoing edges of the node. Once a token is put on an edge which has a time cost \( C \), it takes \( C \) amount of time to transfer the token from the start node of the edge to the end node of the edge. If \( T_{0} \) is the time at which tokens start to leave input nodes and \( T_{1} \) is the time at which the last token in the program is absorbed by an output node (i.e., the time at which no token is left in the program), \( T_{1} - T_{0} \) is then defined as the time cost of the data flow program.

The execution and time cost definitions given in this paper imply the underlying computer system is dynamic [1] and has enough processors to support the execution of a data flow program.

4. Time Cost Analysis Technique

In this section, we propose a technique to analyze the time cost of data flow programs. It is assumed the data flow program under investigation doesn't contain nested loops; each loop in the data flow program is controlled by a number of distribute nodes as illustrated in Figure 2; the body of the loop communicates with the outside world only through these distribute nodes; and within the loop body, there is no any distribute or select node. Since an input node or an output node doesn't connect to the body of any loop, any flow associated with an input node or an output node is equal to either zero or one. Since no select or distribute node is used in the body of a loop, no or node is used in the body of any loop.

In the time cost analysis technique, it is assumed the time cost of each node and each edge is known; and sufficient information is provided so that the flow of each edge can be determined. The goal of the analysis is to derive the accurate time cost from a data flow program. However, as already been found [9], knowing only the flow of each edge is not sufficient to derive the accurate time cost from a general parallel program (for example, the accurate time cost can not be derived if there are two parallel paths, both belong to a loop and at least one of them contains some loops; or there are two parallel paths, both belong to a loop and both contain some decision nodes [9]). This is why the constraints described in the previous paragraph are used in the technique. Eliminating these constraints requires to provide extra flow information to the technique or make assumptions about flow distributions within the technique as what is done in the TCAS system [9].

A loop is much more difficult to analyze in a data flow program than in a traditional von Neumann style program. In a von Neumann style program, computations in the \((i+1)\)'th iteration of a loop can not start until all the computations in the \(i\)'th iteration are finished. Therefore, if the \(i\)'th iteration takes \( C_{i} \) time, the time cost for \( n \) iterations is simply \( \sum_{i=1}^{n} C_{i} \).

Since in a data flow program, a computation in the \((i+1)\)'th iteration of a loop may already be finished even before another computation in the \(i\)'th iteration starts, the loop analysis technique for von Neumann style programs can not be used. Instead of determining the time cost of a loop, an attempt is made in the proposed technique to determine the time at which each computation in the \(i\)'th iteration of a loop starts. Once the time at which each computation in the last iteration starts is determined, the time at which tokens generated by the last iteration leave the loop can be easily determined.

To derive the time cost from a data flow program, the technique deletes all the nodes that are neither input nodes nor output nodes from the program. Every time a node is deleted, the time cost of the node; and the time cost and flow of each edge associated with the node are properly distributed within the program. After all the nodes that are neither input nodes nor output nodes are deleted, the time cost of the data flow program can be easily obtained. In deriving the time cost, the technique takes five steps. These five steps are described in detail in the remaining part of this section.
Step One: Perform a flow balance operation to determine the flow of each edge in the program. The properties of flows have been described in Section 3. A flow balance operation first uses these properties to set up a set of linear equations; and then solves these equations for flows. It is assumed the user will provide sufficient information to help the flow balance operation solve these equations.

Step Two: Remove all the select nodes from the program. If a select node has the form as given in Figure 1.g, the select node and the edges associated with it are replaced by a structure as given in Figure 3.a. In Figure 3.a, the time costs of the or and and nodes are set to zero; and the time cost of the operation node is set to the time cost of the select node. It can be seen the time cost of the select node in Figure 1.g and the time cost of the structure in Figure 3.a are the same (note, it has been assumed no select node is used in a loop and no redundant tokens are provided).

Step Three: Remove all the operation nodes from the program. For an operation node and the two edges associated with the operation node as shown in Figure 1.c, they are replaced by a single edge. The flow of the edge is set to \( e_i \) and the time cost of the edge is set to \( C_{op} + C_0 \), where \( C_{op} \) is the time cost of the operation node.

Step Four: Remove all the nodes that are neither input nodes nor output nodes from the program. To remove one node, the following procedure is used.

4.1 For a fork node as given in Figure 1.f, if its incoming edge is connected to an input node, the fork node and the incoming edge are deleted; the start node of each outgoing edge is set to the start node of the incoming edge; and the time cost of the \( j \)'th outgoing edge is set to

\[
C_i + C_f + C_{oj}
\]

where \( C_f \) is the time cost of the fork node. Figure 3.b shows the result.

4.2 For an and node as given in Figure 1.d, if all its incoming edges are connected to input nodes, the and node and the incoming edges are deleted; the start node of the outgoing edge is set to the start node of any incoming edge; and the time cost of the outgoing edge is set to

\[
C_{max} + C_{and} + C_0
\]

where

\[
C_{max} = \max(C_{1}, \ldots, C_{in})
\]

4.3 For an or node as given in Figure 1.e, if all its incoming edges are connected to input nodes, the or node and the incoming edges are deleted; the start node of the outgoing edge is set to the start node of any incoming edge; and the time cost of the outgoing edge is set to

\[
\text{COST} = \sum_{j=1}^{n} e_{ij}C_i + C_{or} + C_0
\]

where \( C_{or} \) is the time cost of the or node. Figure 3.d shows the result.

Figure 3: Rules of replacement
4.4 For a distribute node as given in Figure 1.1, if all its incoming edges are connected to input nodes, the distribute node and the incoming edges are deleted; the start node of each outgoing edge is set to the start node of any incoming edge; and the time costs of the two outgoing edges are set to

\[ C_x = \max(C_i, C_j) + C_d + C_i \]

\[ C_y = \max(C_i, C_j) + C_d + C_j \]

where \( C_d \) is the time cost of the distribute node. Figure 3.e shows the result.

4.5 For a loop as given in Figure 2, if all the incoming edges of the loop are connected to input nodes, remove the loop using the following procedure.

4.5.1 Set the time cost of the edge marked with \( e_{ij} \) to \( x_j + C_{ij} \) (originally, the time cost of the edge is \( C_{ij} \) where \( x_j \) is an unknown that indicates the time at which a token leaves the \( j \)th distribute node that controls the loop.

4.5.2 Remove all the nodes that are in the body of the loop (these nodes should be either fork nodes or and nodes). To remove one node, one can first find a node whose incoming edges are all connected to some distribute nodes that control the loop; and then apply step 4.1 (for a fork node) or step 4.2 (for an and node). Once all the nodes in the loop body are removed, all the \( 2n \) edges marked with \( e_{ij}' \), \( e_{ij}'' \) and \( e_{ij}'' \) in Figure 2 should directly be connected to the \( n \) distribute nodes that control the loop. Generally speaking, the time costs of these \( 2n \) edges (i.e., \( C_{ij}', C_{ij}, C_{ij}'' \) and \( C_{ij}'' \)) are functions of \( x_1, \ldots, x_n \) (refer to step 4.5.1). If the \( n \) tokens required for the \( k \)th iteration leave the \( n \) distribute nodes at time \( x_1, \ldots, x_n \), then the \( n \) tokens required for the \( (k+1) \)th iteration leave the \( n \) distribute nodes at time

\[ \max(C_{ij} + C_m, C_{ij} + C_m, C_m + C_m) + C_d \]

where \( 1 \leq j \leq n \), \( C_m \) is the time cost of an or node and \( C_d \) is the time cost of a distribute node.

4.5.3 Replace the loop with the structure shown in Figure 3.f. The start node of each edge in Figure 3.f is set to the start node of any edge marked with \( e_{ij} \) or \( e_{ij} \) (\( 1 \leq j \leq n \)) in Figure 2. The quantity \( x_j' \) (\( 1 \leq j \leq n \)) is calculated using the following procedure:

for \( j \) from 1 to \( n \) do

\[ x_j' = \max(C_{ij} + C_m + C_m + C_m, C_m + C_m) + C_d; \]

for \( k \) from 1 to \( e_{ij}' \) do

begin

\[ T_j' = \max(C_{ij} + C_m + C_m + C_m, C_m + C_m); \]

for \( j \) from 1 to \( n \) do

\[ x_j' = T_j'; \]

end;

for \( j \) from 1 to \( n \) do

\[ x_j' = x_j. \]

In the above procedure, \( e_{ij}' \) indicates the number of times the body of the loop is executed. Therefore,

\[ e_{ij}' = \cdots = e_{in}' = e_{1}' = \cdots = e_{in} \]

As mentioned before, \( C_{ij}', C_{ij}'' \) (\( 1 \leq j \leq n \)) are functions of \( x_1, \ldots, x_n \). If the execution of the program starts at time zero, the initial value of \( x_j \) (\( 1 \leq j \leq n \)) in the above procedure indicates the time at which the token required for the first iteration leaves the \( j \)th distribute node; and \( x_j' \) indicates the time at which the token generated by the last iteration leaves the loop via the \( j \)th distribute node.

Step Five: Determine the time cost of the program. After the first four steps, the program only contains input and output nodes. If the program contains \( n \) edges (each edge connects an input node and an output node), the flow of the \( i \)th edge is \( e_i \) and the time cost of the \( i \)th edge is \( C_i \), then the the cost of the whole program is:

\[ \sum_{i=1}^{n} \max(e_i, C_i) \]

In the next section, we use an example to illustrate how this technique can be used to derive the time costs of data flow programs.

5. Example

Consider the integration by the trapezoidal rule which is used as an example in [10].

\[ s \leftarrow [f(a) + f(b)]/2; \]
\[ x \leftarrow a + h; \]

for \( i \) from 1 to \( n-1 \) do
begin
  $s \leftarrow s + f(x)$;
  $x \leftarrow x + h$;
end;

$s \leftarrow s \ast h$;

The corresponding data flow program is shown in Figure 4.a. All the operations in the program are specified as follows:

- \texttt{op1: $[v(x)+f(b)]/2$ }
- \texttt{op2: +h}
- \texttt{op3: Sm-1}
- \texttt{op4: s f(x)}
- \texttt{op5: +1}
- \texttt{op6: *h}

Suppose the time costs of \texttt{fork}, \texttt{or} and \texttt{and} nodes are zero; the time cost of any outgoing edge of a \texttt{fork}, \texttt{or} or \texttt{and} node is zero; the time cost of any other edge is $C$; the time cost of \texttt{operation} node $i$ is $C_i$, and the time cost of \texttt{distribute} nodes is $C_d$.

Further assume $C_1 \geq C_4 \geq C_23$. Let $e_1 = 1$, $e_5 = 1$ and $e_{23} = n-1$.

First, all flows are computed using the flow balance operation. All \texttt{operation} nodes are then removed. The resultant graph after the flow balance operation and the removal of \texttt{operation} nodes is shown in Figure 4.b.

The \texttt{and} node and the \texttt{fork} node which are outside the loop are then removed. After the removal of these two nodes, all the incoming edges to the loop are connected to \texttt{input} nodes. An unknown $x_j$ ($1 \leq j \leq 3$) is then added to the time cost of the outgoing edge associated with the $j$'th \texttt{distribute} node which controls the loop. Figure 4.c shows the result.

All the nodes in the body of the loop (i.e., \texttt{fork} nodes and \texttt{and} nodes) are then removed. The result is shown in Figure 4.d, where

\begin{align*}
  y_1 &= \max(x_1, x_2) + C_4 + 2C \\
  y_2 &= x_1 + C_2 + 2C \\
  y_3 &= x_2 + C_2 + 2C \\
  y_4 &= x_3 + C_2 + C_3 + 3C
\end{align*}

Since all the nodes in the loop body are removed, the loop and the edges associated with the loop as shown in Figure 4.d are replaced with the structure as given in Figure 4.e, where $x_1'$, $x_2'$ and $x_3'$ are computed using the following procedure (refer to step 4.5.3 of the technique given in Section 4):
The time cost of the data flow program is then:

\[ \max(x_1', x_2' + C, x_3' + C) \]

where \( x_1' \), \( x_2' \) and \( x_3' \) are computed using the above procedure.

Since it has been assumed that \( C_1 \geq C_4 \geq C_2,3 \), it can be shown that in each iteration of the loop in the above procedure, \( x_1 \geq x_2 \geq x_3 \). Therefore, \( x_1' \geq x_2' \geq x_3' \). Using these properties, it can be found that if \( n-1 = 0 \), then

\[
\begin{align*}
x_1' &= A_1 \\
x_2' &= A_2 \\
x_3' &= A_3
\end{align*}
\]

if \( n-1 > 0 \), then

\[
\begin{align*}
x_1' &= \max(A_1 + (n-1)B_1, A_2 + B_3 + (n-2)B_1, A_3 + (n-1)B_3) \\
x_2' &= \max(A_2 + (n-1)B_2, A_3 + B_3 + (n-2)B_2, A_3 + (n-1)B_3) \\
x_3' &= A_3 + (n-1)B_3
\end{align*}
\]

where

\[
\begin{align*}
A_1 &= C_1 + 2C_d + C_d \\
A_2 &= \max(C_2, C_3) + 2C_d + C_d \\
A_3 &= C_3 + 2C_d + C_d \\
B_1 &= C_4 + 2C_d + C_d \\
B_2 &= C_2 + 2C_d + C_d \\
B_3 &= C_2 + C_3 + 3C_d + C_d
\end{align*}
\]

Therefore, if \( n-1 = 0 \), the time cost of the data flow program becomes:

\[
x_1' + C_6 + 2C = A_1 + C_6 + 2C
\]
if \( n-1 > 0 \), the time cost of the data flow program becomes:

\[
x_1 + C_6 + 2^*C = \max \{ A_1 + (n-1)^*B_1, A_3 + B_3 + (n-2)^*B_1, A_3 + (n-1)^*B_3 + C_6 + 2^*C \}
\]

where

\[
\begin{align*}
A_1 &= C_1 + 2^*C + C_d \\
A_3 &= C_3 + 2^*C + C_d \\
B_1 &= C_4 + 2^*C + C_d \\
B_3 &= C_2 + C_3 + 3^*C + C_d
\end{align*}
\]

6. Conclusion

A technique was proposed in this paper to analyze the time cost of data flow programs. Generally speaking, this technique provides numeric results. However, it can provide analytic results if the data flow programs under investigation are loop-free. This means a close form symbolic expression can be derived using the technique to represent the time cost of a loop-free data flow program.

Much effort has been spent on the time cost analysis of von Neumann style programs. This paper reports our first attempt in analyzing the time cost behavior of data flow programs. Many extensions are possible. For example, one could do the analysis assuming the underlying computer system is static or contains a limited number of processors; instead of analyzing the accurate time cost, one could analyze the average time cost; etc. Currently, our research work in this area aims at analyzing the time cost of data flow programs through simulation and developing more general analytic approaches.

8. References


