Modeling and Optimum Time Performance for Concurrent Processing

Roland R. Mielke, John W. Stoughton, and Sukhamoy Som

GRANT NAG1-683
AUGUST 1988
I. INTRODUCTION

The development of a new graph theoretic model for describing data and control flow associated with the execution of large-grained algorithms in a special distributed computing environment is presented. The model is identified by the acronym ATAMM which represents Algorithm To Architecture Mapp­ing Model. The purpose of such a model is to provide a basis for estab­lishing rules for relating an algorithm to its execution in a multiprocessor environment. Specifications derived from the model lead directly to the description of a data flow architecture. The availability of the ATAMM model is important for at least three reasons. First, it provides a context in which to investigate algorithm decomposition strategies without the need to specify a specific computer architecture. Second, the model identifies the data flow and control dialog required of any data flow architecture which implements the algorithm. Third, the model provides a basis for cal­culating analytically performance bounds for computing speed and throughout capacity.

The problem domain of the ATAMM model consists of decision free algo­rithms with computationally complex primitive operations which are assumed to be implemented in a dedicated data flow environment. The algorithms are such as may be found in (but not limited to) large scale signal processing and control applications. The anticipated multiprocessor environment is assumed to consist of two to twenty processing elements for concurrent exe­cution of the various algorithm primitives.

The development of new computer architectures based upon distributed, multiprocessor organizations [1], [2] is motivated mainly by the requirement for increased speed and greater throughput capability in complex signal processing applications [3]. Recent advances in the production of
high-density microelectronics [4] has made possible the construction of parallel architectures consisting of identical, special purpose computing elements [5]. A number of models for describing the behavior of algorithms in this setting have been developed [6] - [8]. However, these models represent only the data flow and do not adequately display the complex issues of communication and control flow which must occur in any realization of the model. For this reason, it has been difficult to investigate how to effectively match the decomposition and scheduling of algorithms to the structure and control of parallel architectures. The importance of better understanding the relationship between algorithms and architectures is only now becoming recognized [9].

In Section II of the paper, the modeling process to describe algorithms in data flow architectures, ATAMM, is presented. The model consists of three Petri net marked graphs called the algorithm marked graph (AMG), the node marked graph (NMG), and the computational marked graph (CMG). In Section III, the operating characteristics of these graphs are investigated. A state variable description is presented and used to establish the graph properties of reachability, liveness, and safeness. Time performance measures for concurrent processing are defined in Section IV. The ATAMM model is used as the basis for calculating analytically lower bounds for these performance measures. Then in Section V, an operating strategy which achieves optimum time performance is developed. Several examples are presented to illustrate these concepts, and the results of experimental runs on actual multiprocessor hardware are reported.

II. ATAMM MODEL DEVELOPMENT

In this section the ATAMM model to describe concurrent processing of
decomposed algorithms is presented. The model consists of a set of Petri net marked graphs which incorporate general specifications of communication and processing associated with each computational event in a data flow architecture. First, a detailed description of the problem context is stated. This is followed by the definition of the ATAMM model consisting of the algorithm marked graph, the node marked graph, and the computational marked graph. Some familiarity with Petri nets [10] and marked graphs [11] is assumed in this presentation.

The problems of interest are decision-free, computationally complex problems as are often found in signal processing and control applications. A problem description normally results in the definition of a function given by the triple (X,Y,F). The set $X$ represents the set of admissible inputs, the set $Y$ represents the set of admissible outputs, and $F:X\rightarrow Y$ is the rule of correspondence which unambiguously assigns exactly one element from $Y$ to each element of $X$. Associated with a computational problem is one or more algorithms. An algorithm is an explicit mathematical statement, expressed as an ordered set of primitive operations, which explains how to implement the rule of correspondence $F$. In general, a given problem can be decomposed by several different primitive operator sets. Also, for a given primitive operator set, there are often different orderings of primitive operations which can be specified to carry out the problem. Of special interest are algorithm decompositions in which two or more primitive operations can be performed concurrently. For such decompositions, the potential exists for decreasing the computational time required to execute the problem by making available a set of identical computational resources capable of implementing each of the primitive operations.
The hardware environment for executing the decomposed algorithms is assumed to consist of $R$ identical processors or functional units (FUNs) where $R$ has a value in the range of two to twenty. This range of resources is suggested for practical reasons due to the large-grained aspect of the algorithm decomposition and the need to maintain small communication times relative to process times. Each FUN is a processor having local memory for program storage and temporary input and output data containers. Each FUN can execute any algorithm primitive operation. The FUNs share a common global memory (GLM) which may be either centralized or distributed. The coordination of FUNs in relation to data and control flow is directed by the graph manager (GRM). The GRM also may be centralized or distributed. Output created by the completion of a primitive operation is placed into global memory only after the output data containers have been emptied. That is, outputs must be consumed as inputs to successor primitive operations before allowing new data to fill the output locations. Assignment of a functional unit to a specific algorithm primitive operation is made by the GRM only when all inputs required by the operation are available in global memory and a functional unit is available.

An algorithm marked graph is a marked graph which represents a specific algorithm decomposition. Vertices of the algorithm graph are in a one-to-one correspondence with each occurrence of a primitive operation. The algorithm graph contains an edge $(i,j)$ directed from vertex $i$ to vertex $j$ if the output of primitive operation $i$ is an input for primitive operation $j$. Edge $(i,j)$ is marked with a token if an output from primitive operator $i$ is available as an input to primitive operator $j$. When constructing an algorithm graph, vertices (primitive operations) are displayed as circles, and edges (input-output signals) are displayed as directed line segments.
connecting appropriate vertices. The presence of a token on an edge is indicated by a solid dot placed on the edge. Source transitions and sink transitions for input and output signals are represented as squares. Sources for constants are not usually included in the algorithm marked graph; however, triangles are used for this purpose when necessary.

To illustrate the construction of an algorithm marked graph, consider the problem of computing the output of a discrete linear system given a sequence of inputs to the system. Let the system be described by the state equation

\[ x(k) = Ax(k-1) + Bu(k) \]

and output equation

\[ y(k) = Cx(k). \]

where \( x \) is a \( p \)-vector, \( u \) is an \( m \)-vector, and \( y \) is an \( r \)-vector. The primitive operations are defined as matrix multiplication and vector addition, and the natural algorithm decomposition resulting from the state equation description is selected. The algorithm marked graph for this decomposed algorithm is shown in Figure 1. The initial marking indicates that initial condition data are available.

The algorithm marked graph is a useful tool for representing decomposed algorithms and for displaying data flow within an algorithm. However, the algorithm graph does not display procedures that a computing structure must manifest in order to perform the computing task. In addition, the issues of control, time performance, and resource management are not apparent in this graph. These important aspects of concurrent processing are included in the ATAMM model through the definition of two additional graphs. The node marked graph (NMG) is defined to model the execution of a primitive
operation. The computational marked graph, obtained from the AMG and the NMG by a set of construction rules, integrates both the algorithm requirements and the computing environment requirements into a comprehensive graph model. These additional marked graphs are defined in the following.

The NMG is a Petri net representation of the performance of a primitive operation by a functional unit. Three primary activities, reading of input data from global memory, processing of input data to compute output data, and writing of output data to global memory, are represented as transitions (vertices) in the NMG. Data and control flow paths are represented as places (edges), and the presence of signals is notated by tokens marking appropriate edges. The conditions for firing the process and write transitions of the NMG are as defined for a general Petri net, while the read transition has one additional condition for firing. In addition to having a token present on each incoming signal edge, a functional unit must be available for assignment to the primitive operation before the read node can fire. Once assigned, the functional unit is used to implement the read, process, and write operations before being returned to a queue of available FUNs. The initial marking for an NMG consists of a single token in the "process ready" place. The NMG model is shown in Figure 2.

A computational marked graph (CMG) is constructed from the AMG and the NMG by the following rules.

1. Source and sink nodes in the algorithm marked graph are represented by source and sink nodes in the CMG.
2. Nodes corresponding to primitive operations in the algorithm marked graph are represented by NMGs in the CMG.
3. Edges in the algorithm marked graph are represented by edge pairs, one forward directed for data flow and one backward directed for
control flow, in the CMG. The initial marking for the edge pair consists of a single token in the forward-directed place if data are available, or a single token in the backward-directed place if data are not available.

The play of the CMG proceeds according to the following graph rules.

1) A node is enabled when all incoming edges are marked with a token. An enabled node fires by encumbering one token from each incoming edge, delaying for some specified transition time, and then depositing one token on each outgoing edge.

2) A source node and a sink node fire when enabled without regard for the availability of a FUN.

3) A primitive operation is initiated when the read node of an NMG is enabled and a FUN is available for assignment to the NMG. A FUN remains assigned to an NMG until completion of the firing of the write node of the NMG.

In order to illustrate the construction of a computational marked graph, the CMG corresponding to the algorithm marked graph of Figure 1 is shown in Figure 3. The computational marked graph is useful because it clearly displays the data and control flow which must occur in any hardware implementation of the model process, and because it provides a hardware independent context in which to evaluate process performance.

The complete ATAMM model consists of the algorithm marked graph, the node marked graph, and the computational marked graph. A pictoral display of this model is shown in Figure 4. In the next section, important operating characteristics of the ATAMM model are investigated.
III. MODEL CHARACTERISTICS

In the previous section, a marked graph model consisting of the AMG, the NMG, and the CMG is defined as a means to describe concurrent processing of decomposed algorithms. In this section the ATAMM model is studied analytically to determine important graph operating characteristics. First, a state description which expresses the next graph marking as a function of the present marking and a vector indicating which transition is to be fired is developed. Then, the marked graph properties of reachability, liveness, and safeness are considered for the CMG. Two excellent papers by Murata [11], [12] on properties of marked graphs are the source for much of the material presented in this section.

Let $G$ be a marked graph consisting of $m$ places and $n$ transitions. The $m$-vector $M_k$ denotes the marking vector for $G$ resulting from the firing of some sequence of $k$ transitions. The following two definitions are necessary to develop the state description of the CMG.

Definition 1: Complete Incidence Matrix. The complete incidence matrix for a marked graph $G$ is the $(nxm)$ matrix $A = [a_{ij}]$ having rows corresponding to transitions, columns corresponding to places, and where

$$a_{ij} = \begin{cases} +1 & \text{if place } j \text{ is incident at transition } i \\ -1 & \text{if place } j \text{ is not incident at transition } i \\ 0 & \text{if place } j \text{ is directed out of (into) the transition} \end{cases}$$

Definition 2: Elementary Firing Vector. An elementary firing vector $u_k$ is an $n$-vector having all zero entries except for the $i$th component which is 1 denoting that transition $i$ is the $k$th transition to fire in some transition firing sequence.

To gain insight to the state equation description, it is helpful to
consider the firing of transition \( k \). If \( a_{ki} = -1(1) \), place \( i \) is an input (output) place to transition \( k \). Therefore, transition \( k \) is enabled if \( M(i) = 1 \) for each input place. When transition \( k \) fires, one token is removed from each input place and one token is added to each output place. These observations lead to the following next state description for a marked graph.

**Property 1: Next State Description.** For a marked graph \( G \) with present marking vector \( M_{k-1} \) and elementary firing vector \( u_k \), the next marking vector is given by

\[
M_k = M_{k-1} + A^T u_k.
\]

The next state description can be used to express the graph marking resulting from the application of sequences of elementary firing vectors. This is done in the next definition and property.

**Definition 3: Firing Count Vector.** Let \( (u_1, u_2, \ldots, u_d) \) be a sequence of elementary firing vectors taking a marked graph \( G \) from an initial marking \( M_0 \) to a destination marking \( M_d \). The firing count vector \( x_d \) for this firing sequence is defined by

\[
x_d = \sum_{k=1}^{d} u_k.
\]

**Property 2: State Equation Description.** For a marked graph \( G \) with initial marking vector \( M_0 \), the marking vector resulting from the application of elementary firing vector sequence \( (u_1, u_2, \ldots, u_d) \) is given by
Using the state description of a marked graph as a basis, the property of reachability is investigated. Necessary and sufficient conditions for a CMG marking vector to be reachable from an initial marking are established, and it is shown that the number of tokens contained in any directed circuit of the CMG is invariant under transition firings.

**Definition 4: Reachability.** A marking $M_d$ is reachable from an initial marking $M_0$ if there exists a sequence of elementary firing vectors that transforms $M_0$ to $M_d$.

The following definition is required to state the reachability conditions for a CMG.

**Definition 5: Fundamental Circuit Matrix.** Let T be a tree of a connected marked graph G. The set of $(m-n+1)$ circuits, each uniquely formed by appending one cotree edge to the tree, is called the set of fundamental circuits of G for tree T [13]. The fundamental circuit matrix for G for tree T is the $(m-n+1) \times (m)$ matrix $B_f = [b_{ij}]$ having rows corresponding to fundamental circuits, columns corresponding to places, and where

$$b_{ij} = \begin{cases} +1(-1) & \text{if place } j \text{ is contained in } f\text{-circuit } i \text{ and} \\ & \text{the place and circuit directions agree} \\ 0 & \text{if place } j \text{ is not contained in } f\text{-circuit } i. \end{cases}$$

**Property 3: Reachability in the CMG.** In a computational marked graph G, a marking $M_d$ is reachable from an initial marking $M_0$ if and only if $B_f M_d = B_f M_0$, where $B_f$ is a fundamental circuit matrix for G.
Proof. It is shown in [11] (Theorem 3) that the property is true for marked graphs containing no token-free directed circuits. By the construction rules for the CMG, directed circuits occur in exactly four ways. First, each NMG consists of a directed circuit which contains an initial marking token in the "process ready" place. Second, a directed circuit is formed each time an NMG is linked to another NMG. Since one of the two linking places contains an initial marking token and both places are contained in the circuit, this circuit is never token free. Third, directed circuits exist in the CMG corresponding to interconnected feedforward paths in the algorithm marked graph. Each such circuit contains one or more backward-directed control edge containing one initial marking token. Fourth, directed circuits exist in the CMG corresponding to directed circuits in the algorithm marked graph. Each such circuit contains exactly one forward-directed edge containing one initial marking token representing initial condition data. Therefore, the CMG contains no token-free directed circuits and the property follows.

As a direct consequence of the reachability property of the CMG, it can be shown that the number of tokens in any directed circuit is constant. This characteristic is stated as Property 4.

Property 4: Token Count Invariance. In a CMG, the number of tokens contained in a directed circuit is invariant under transition firing.

Proof. Consider a directed circuit C of a CMG. The entries in the row of a circuit matrix B corresponding to C are +1 in columns representing edges in C and are 0 otherwise. If M is a marking vector, the component of BM corresponding to C is equal to the number of tokens in directed circuit C under marking M. Therefore, if M_d is any marking reachable from an initial marking M_0, it follows from Property 3 that BM_d = BM_0. That is, the number of
tokens in directed circuit C under initial marking $M_0$ is equal to the number of tokens under any marking $M_d$ reachable from $M_0$. This completes the proof.

Next, liveness and a closely related property called consistency are considered. It is shown that the CMG is live and consistent.

**Definition 6: Liveness.** A marked graph $G$ is said to be live for a marking $M$ if, for all markings reachable from $M$, it is possible to fire any transition of $G$ by progressing through some transition firing sequence.

**Property 5: Liveness in the CMG.** The computational marked graph is live for all appropriate initial marking vectors.

**Proof.** It is shown in [12] (Property 2) that a marked graph $G$ is live for a marking $M$ if and only if $G$ contains no token-free directed circuits in marking $M$. As stated in the proof of Property 3, for all appropriate initial markings $M_0$, the CMG contains no token-free directed circuits. Therefore, the property follows.

**Definition 7: Consistency.** A marked graph $G$ is said to be consistent if there exists a marking $M$ and a transition firing sequence $S$ from $M$ back to $M$ such that every transition occurs at least once is $S$.

**Property 6: Consistency in the CMG.** A connected computational marked graph $G$ is consistent. In addition, each transition of $G$ occurs an equal number of times in a firing sequence from a marking $M$ back to $M$.

**Proof.** From Property 2, if a CMG is consistent, then there exists a marking $M_d = M_0$ and a firing count vector $x_d > 0$ such that $A^T x_d = 0$. The converse is also true. The incidence matrix for a marked graph $G$ is an $(n \times m)$ matrix $A$. If $G$ is connected, then it is known [13] that the rank of $A$ is $n-1$, and thus the null space of $A^T$ has dimension one. It is observed that
each row of $A^T$ has one (1), one (-1), and all remaining terms are (0). Therefore, if $C_j$ denotes the $j^{th}$ column of $A^T$, it follows that

$$\sum_{j=1}^{n} C_j = 0.$$ 

Thus, there exists a vector $x_d = [k \ k \ ... \ k]^T$, $k > 0$, which uniquely satisfies $A^T x_d = 0$. This completes the proof.

The final graph property considered in this section is safeness. This property is first defined, and then it is shown that CMG is safe.

Definition 8: Safeness. A marked graph $G$ is said to be safe for marking $M$ if, for all markings reachable from $M$, no place contains more than one token.

Property 7: Safeness in the CMG. The computational marked graph is safe for all appropriate initial marking vectors.

Proof. By Property 4, the token count for each directed circuit of the CMG is invariant under transition firing. Therefore it is sufficient to show that each edge of the CMG belongs to at least one directed circuit containing a single token. By the construction rules for the CMG, all CMG edges can be classified into two groups, NMG edges and linking edges. NMG edges occur in groups of three and always form a directed circuit containing one token. Linking edges occur in pairs, one forward directed and one backward directed, and also form a directed circuit with the forward directed edges of the NMG. One of the linking edges, but not both, always contains one token while the forward directed edges of the NMG contain no tokens.
Therefore, each edge of the CMG is contained in a directed circuit with one token, and the property follows.

IV. PERFORMANCE ANALYSIS

The importance of the ATAMM model is that it establishes a context in which to investigate the performance of decomposed algorithms in multiprocessor data flow architectures. In this section, performance measures indicating computing speed and throughput capacity are defined. Bounds for these quantities are calculated analytically from the algorithm marked graph and the computational marked graph. This information is essential for efficiently matching algorithm decompositions with architecture implementations. The work presented in this section is an interesting application and extension of recent investigations of the performance of Petri nets [14], [15] and marked graphs [16].

It is assumed that a decomposed algorithm is implemented in a multiprocessor architecture containing R computing resources or functional units. Each functional unit is capable of performing any of the primitive operations whose sequence defines the decomposition. A computational task consists of completing the algorithm for one frame of data and is initiated when an input data token from the source node is encumbered. Task output occurs when a corresponding output data token is deposited at the output sink node. A task is completed when all computing associated with the task is completed. It should be noted that task output and task completion do not always coincide. In many iterative signal processing algorithms, computing to generate initial conditions for the next iteration often occurs after an output has been calculated. Task completion is usually indicated in the AMG or CMG by the return of the graph to some steady-state initial
marking. To facilitate measurement of throughput capacity, it is assumed that tasks are repeated periodically with new input data sets. New data sets are available continuously as input tokens from the input source node. Included in this problem class are iterative algorithms where the present task requires as inputs data from previous task calculations.

Concurrency in this problem setting occurs in two ways. First, different functional units may perform simultaneously several primitive operations belonging to a single task. This type of concurrency is referred to as vertical concurrency. Vertical concurrency has a direct effect on task computing speed. It is limited by the number of primitive operations that can be performed simultaneously in a given algorithm decomposition, and by the number of functional units available to perform the primitive operations. Second, different functional units may perform simultaneously primitive operations belonging to different tasks sequentially input to the computing system. Called horizontal concurrency, this type of concurrency has a direct effect on throughput capacity. It is limited by the capacity of the graph to accommodate additional task inputs, and by the number of functional units available to implement the tasks. In the following it is shown that the process of algorithm decomposition imposes bounds on the amount of vertical concurrency and horizontal concurrency possible in a given problem. If sufficient computing resources are available, operation at these bounds can be achieved. If the number of computing resources is limited, the bounds cannot be reached simultaneously and trade-offs between the amount of vertical concurrency and horizontal concurrency are possible.

Three performance measures for concurrent processing are defined. The first two parameters, TB10 and TT, are indicators of computing speed and reflect the degree of vertical concurrency. The third parameter, TBO, is a
measure of throughput capacity and thus reflects the degree of horizontal and vertical concurrency.

**Definition 9: TBIO.** The performance measure TBIO is the computing time which elapses between a task input and the corresponding task output.

**Definition 10: TT.** The performance measure TT is the computing time which elapses between a task input and the completion of all computation associated with that task.

**Definition 11: TBO.** The performance measure TBO is the computing time which elapses between successive task outputs when the graph is operating periodically in steady-state.

The remainder of this section is devoted to developing lower bounds for these performance measures.

Let G denote an algorithm marked graph representing a decomposed algorithm. The lower bound for TBIO is the shortest time required for a data token from the data input source to propagate through the graph to the data output sink. Similarly, the lower bound for TT is the shortest time required to complete all computing activity initiated by the injection of a data input source. These shortest times are the actual performance times when only a single task is active in the graph during any time interval (no horizontal concurrency), and as many computing resources as are required are available (maximum vertical concurrency). Under these operating conditions, lower bounds for TBIO and TT are calculated by identifying certain longest paths in a graph obtained from the algorithm marked graph. This new graph, called the modified algorithm graph \( G_M \), is defined and then used to determine lower bounds for TBIO and TT.

**Definition 12: Modified Algorithm Graph.** Let \( p_i \) be a place of G, directed from transition \( t_r \) to transition \( t_s \), which contains a token of the initial
marking. The modified algorithm graph $G_M$ is obtained from the graph $G$ by the following construction rules.

1. Place $p_i$ is deleted from $G$.
2. A new place $p_{i1}$, directed from the data input source to transition $t_s$, is added to $G$.
3. A new output sink $s_i$ different from all other output sinks, and a new place $p_{i2}$, directed from transition $t_r$ to $s_i$, are added to $G$.
4. The above rules are repeated for each place of $G$ containing a token of the initial marking.

Lower bounds for $T_{BIO}$ and $TT$ are presented in Theorem 1 and Theorem 2 respectively.

**Theorem 1: Lower Bound for $T_{BIO}$**. Let $P_i$ be the $i^{th}$ directed path in $G_M$ from the data input source to the data output sink, and let $T(P_i)$ denote the sum of transition times for transitions contained in $P_i$. Then,

$$T_{BIO_{LB}} = \text{Max} \{T(P_i)\},$$

where the maximum is taken over all paths $P_i$ in graph $G_M$.

**Proof.** Without loss of generality, let $t_f$ be the last transition in all paths $P_i$ directed from the data input source to the data output sink. Transition $t_f$ is enabled when each input place for $t_f$ contains a token. Since by assumption a computing resource is available, $t_f$ fires as soon as it becomes enabled. Let $p_q$ be the last input place for $t_f$ to acquire a token, and let $t_g$ be the input transition for place $p_g$. Continuing this labeling procedure results in a backward path construction process. This process is repeated, first at $t_g$, and then at each succeeding transition.
until the data input source is reached, identifying a path $P_j$. By the construction process for the path, it is clear that $T(P_j) = \text{Max}\{T(P_i)\}$, where the maximum is over all paths $P_i$ in $G_M$. It is also clear that $\text{TBIO}_{LB}$ can be no shorter than $T(P_j)$ so that $\text{TBIO}_{LB} \geq T(P_j)$. Since a computing resource is available when each transition in $P_j$ is enabled, the time between input and corresponding output can be no longer than $T(P_j)$ so that $\text{J} < \text{TBIO}_{LB}$. Therefore, $\text{TBIO}_{LB} = T(P_j) = \text{Max}\{T(P_i)\}$, where the maximum is over all paths $P_i$ in $G_M$. This completes the proof.

Theorem 2: Lower Bound for $\text{TT}$. Let $P_i$ be the $i^{th}$ directed path in $G_M$ from the data input source to any output sink, and let $T(P_i)$ denote the sum of transition times of transitions contained in $P_i$. Then,

$$\text{TT}_{LB} = \text{Max}\{T(P_i)\}$$

where the maximum is taken over all paths $P_i$ in graph $G_M$.

Proof. By the construction rules for graph $G_M$, a task is initiated when input data tokens are input from the data input source, and is completed when all output sinks have accepted tokens. Therefore, $\text{TT}$ is the time which elapses from injection of input tokens to the arrival of a token at the last fired output sink. Let $T(P_t) = \text{Max}\{T(P_i)\}$, $P_i$ in $G_M$, be the longest path time of paths from the data input source $s_I$ to any output sink, say $s_t$. Since a token must reach sink $s_t$ before a task is completed, it follows that $\text{TT}_{LB} > T(P_t)$. Since a resource is available for each transition to fire when enabled, and since $P_t$ is the longest path in $G_M$, it also follows that $\text{TT}_{LB} < T(P_t)$. Therefore, $\text{TT}_{LB} = T(P_t) = \text{Max}\{T(P_i)\}$, where the maximum is over all paths $P_i$ in $G_M$. This completes the proof.
To illustrate the application of Theorem 1 and Theorem 2, $TBI_0^{LB}$ and $TT^{LB}$ are computed for the algorithm graph shown in Figure 1. For this example, the following transition times are assumed: $T(1) = 4$, $T(2) = 1$, $T(3) = 5$, and $T(4) = 6$. The modified algorithm graph corresponding to Figure 1 is shown in Figure 5. The modified algorithm graph contains two paths directed from the data input source $s_1$ to the data output sink $s_0$. Path $P_1$ consists of edge set $\{1, 2, 3, 4\}$ with $T(P_1) = 10$, and path $P_2$ consists of edge set $\{5-1, 3, 4\}$ with $T(P_2) = 6$. Therefore, since $T(P_1) > T(P_2)$, path $P_1$ determines the lower bound for $TBI_0$ and $TBI_0^{LB} = 10$. The modified algorithm graph contains two additional directed paths from the data input source $s_1$ to the output sink $s_5$. Path $P_3$ consists of edge set $\{1, 2, 6, 5-2\}$ with $T(P_3) = 11$, and path $P_4$ consists of edge set $\{5-1, 6, 5-2\}$ with $T(P_4) = 7$. Since $T(P_3) > T(P_1) > T(P_4) > T(P_2)$, path $P_3$ determines the lower bound for $TT$ and $TT^{LB} = 11$.

Next a lower bound for the performance measure $TBO$ is presented. Let $G$ be a computational marked graph representing a decomposed algorithm. It is assumed that operating conditions for $G$ are set to maximize horizontal concurrency. That is, data tokens are continuously available at the data input source, and as many computing resources as needed can be called to perform primitive operations. With these conditions, the graph plays periodically in steady-state, and $TBO^{LB}$ is the shortest time possible between successive outputs.

**Theorem 3:** Lower Bound for $TBO$. Let $G$ be a computational marked graph and let $C_i$ be the $i$th directed circuit in $G$. The notation $T(C_i)$ denotes the sum of transition times of transitions contained in $C_i$, and $M(C_i)$ denotes the number of tokens contained in $C_i$. Then,
\[ \text{TBOL}_\text{LB} = \max \{ T(C_i)/M(C_i) \}, \]

where the maximum is taken over all directed circuits in \( G \).

Proof. Without loss of generality, let \( t_f \) be the output transition in \( G \) so that an output is produced each time \( t_f \) completes firing. Then \( \text{TBOL}_\text{LB} \) is the minimum firing period of transition \( t_f \). By Property 6, \( G \) is consistent so that all transitions of \( G \) fire periodically with minimum period \( \text{TBOL}_\text{LB} \). It is shown in [12] (pp. 58-60) that the minimum firing period of each transition of a marked graph is given by \( \max \{ T(C_i)/M(C_i) \} \), where the maximum is taken over all directed circuits \( C_i \) in \( G \). Therefore, the theorem follows.

The computational marked graph shown in Figure 3 is used to illustrate Theorem 3. This CMG contains many directed circuits. However, the directed circuit which contains all NMG nodes of transitions 2 and 4 contains only one token and maximizes the ratio \( T(C_i)/M(C_i) \). Therefore, the shortest time possible between successive outputs in this graph is \( \text{TBOL}_\text{LB} = 7 \). In the next section, a strategy for achieving optimum time performance is investigated.

V. STRATEGY FOR OPTIMUM TIME PERFORMANCE

A model describing decomposed algorithms for implementation in a distributed data flow architecture is described in Sections II and III, and performance measures are defined in Section IV. An important problem remaining is to develop an operating strategy for the ATAMM model which achieves optimum time performance with a minimum number of computing resources. Unfortunately, this problem is equivalent to a class of scheduling problems which is known to be NP-complete. Thus, there exists no algorithm for obtaining an optimum solution which is better than enumerating all possible solutions and then choosing the best one. However, an
important suboptimal operating strategy which achieves optimum time performance, but possibly requires more than the minimum number of computing resources, has been developed. This strategy is presented and illustrated by example in this section.

When presented with continuously available input data sets, the natural behavior of a data flow architecture results in operation where new data sets are accepted as rapidly as the available resources permit. That is, the architecture naturally operates at high levels of horizontal concurrency with the possible loss of capability for achieving high levels of vertical concurrency. This results in performance characterized by high throughput rates, \( T_{BO} = T_{BO}^{LB} \), but relatively poor task computing speed so that \( T_{BIOLB} \gg T_{BO}^{LB} \) and \( TT \gg TT_{LB} \). In many signal processing and control applications, it is important to achieve both high throughput rate and high task computing speeds. Often, designers are willing to provide extra hardware to realize optimum time performance. The suboptimal operating strategy presented in this section results in performance having the following characteristics.

1. When \( R > R_{Max} \), operation achieves \( T_{BIOLB} \), \( TT_{LB} \), and \( T_{BO}^{LB} \). \( R_{Max} \) is computed in implementing the strategy, and represents the minimum number of resources which insures maximum horizontal concurrency and maximum vertical concurrency under this strategy.

2. When \( R_{Max} > R > R_{Min} \), operation achieves \( T_{BIOLB} \) and \( TT_{LB} \), but \( TBO > TBO_{LB} \). The strategy preserves task computing speed or vertical concurrency at the expense of throughput rate or horizontal concurrency. \( R_{Min} \) is also computed in implementing the strategy, and represents the minimum number of resources needed to maintain vertical concurrency with limited horizontal concurrency.

3. When \( R_{Min} > R > 1 \), operation continues but performance degrades so
that $TBIO > TBIO_{LB}$, $TT > TT_{LB}$, and $TBO > TBO_{LB}$.

Implementation of the operating strategy is illustrated in Figure 6. All that is required is to limit the rate at which new input data are presented to the CMG. This is accomplished by adding a control transition connected in a directed circuit with the data input source. The control transition imposes a minimum delay of $D$ time units between inputs. Delay $D$ is chosen according to the following rule:

$$D = \begin{cases} 
TBO_{LB} & R > R_{\text{Max}} \\
TBO_{\text{Min}} & R_{\text{Max}} > R > R_{\text{Min}} \\
TCE & R_{\text{Min}} > R \geq 1.
\end{cases}$$

$TCE$ denotes the total computing effort required to complete the task, and $TBO_{\text{Min}}$, $R_{\text{Max}}$, and $R_{\text{Min}}$ are computed as part of the strategy design procedure.

The operating strategy design process consists of five steps. These steps are presented and explained in the remainder of this section. An operating strategy is developed for the example algorithm graph shown in Figure 7 to illustrate each step as it is presented.

**Step 1.** Choose a convenient transition firing rule. A rule to determine when an enabled transition in the CMG fires must be specified. A natural rule is to specify that enabled transitions fire when a computing resource is available. If conflict exists, such as when there are more enabled transitions than computing resources, then firing occurs according to a priority ordering of the transitions. For the example algorithm graph, the highest to lowest priority ordering of the transitions is chosen as $(5,4,3,-7,2,6,1)$.

**Step 2.** Determine $TBO_{LB}$. The performance bound $TBO_{LB}$ is found from the
computational marked graph by application of Theorem 3. The CMG corresponding to the example algorithm graph is shown in Figure 8. The directed circuit identified in this figure contains 6 transition time units and 2 tokens, and maximizes the ratio $T(C_i)/M(C_i)$ for all directed circuits. Therefore, $TBO_{LB} = 3$.

Step 3. Determine the resource utilization envelope of a single task required for maximum vertical concurrency at steady-state with $TBO = TBO_{LB}$.
The purpose of this step is to determine the number of computing resources required as a function of time to achieve maximum vertical concurrency in a single task. The envelope is determined by playing the graph assuming unlimited resources and an input rate of $TBO_{LB}$ until steady-state operation is reached. The resource utilization envelope is obtained by counting the number of computing resources used for a single task during each time interval. The play of the example algorithm graph under these conditions is shown in Figure 9, and the resulting resource utilization envelope is shown in Figure 10.

Step 4. Stabilize the resource utilization envelope by adding control places as necessary. If the time between inputs to the CMG is increased above $TBO_{LB}$, the resource utilization envelope may change from that observed in Step 3. Since knowledge of the envelope is required to calculate the number of required resources, additional places are appended to the AMG and the CMG to freeze the shape of the envelope. For example, the play of the example algorithm graph of Figure 8 with an injection time of 4 is shown in Figure 11. At this slower injection rate, transition 6 fires one time unit earlier. To prevent time movement of transition 6, a control place directed from transition 2 to transition 6 is added. This place prevents the firing of transition 6 until transition 2 has completed firing. Thus the resource
utilization envelope computed for an input period of \( TBO_{LB} \) is the envelope for all input periods \( TBO > TBO_{LB} \).

**Step 5.** Compute \( R_{Max} \), \( R_{Min} \), and \( TBOMo(R) \) using the resource utilization envelope. \( R_{Max} \) is determined by overlaying resource utilization requirements, each delayed by \( TBO_{LB} \) with respect to the previous one, as shown in Figure 12 for the example. \( R_{Max} \) is equal to the largest resource requirement during any time interval within the steady state operating period. \( R_{Min} \) is the minimum number of resources necessary to insure maximum vertical concurrency with no horizontal concurrency. This number is equal to the maximum resource requirement indicated in the resource utilization envelope for a single task. For the example problem, \( R_{Max} = 5 \) and \( R_{Min} = 3 \). The value of \( TBOMo \) for each resource number \( R \) between \( R_{Max} \) and \( R_{Min} \) inclusive, is determined by increasing the delay between overlapping resource utilization envelopes until the maximum resource requirement is \( R \). \( TBOMo \) is the smallest input delay to produce this resource requirement. For the example, the calculations of \( TBOMo \) for \( R = 4 \) and \( R = 3 \) are illustrated in Figure 13 and Figure 14 respectively. The results of these calculations are \( TBOMo(4) = 3.5 \) and \( TBOMo(3) = 4 \).

The performance of the example algorithm graph is summarized in Figure 15. Optimum time performance of \( TBIOLB = TT_{LB} = 7 \) and \( TBO_{LB} = 3 \) is achieved for \( R > R_{Max} = 5 \). At \( R = 4 \), \( TBIOLB \) and \( TT \) remain at the optimum values and \( TBOMo \) decreases to 3.5. At \( R = 3 \), \( TBIOLB \) and \( TT \) again remain at the optimum values and \( TBOMo \) decreases to 4. For values of \( R \) below \( R_{Min} \), time performance generally degrades. However, in this example \( TBIOLB \) and \( TT \) remain at 7 for \( R = 2 \), while \( TBOMo \) decreases to 6. Finally, at \( R = 1 \), performance degrades to \( TBIOLB = TT = TBO = TCE = 10 \). Another perspective of algorithm
performance is shown in Figure 16. This figure displays throughput rate, 
(I/TBO), as a function of the number of functional units R. The peak height 
of each bar indicates the maximum throughput rate which can be achieved with 
the indicated number of processors. The bars also indicate more clearly 
that operation at any throughput rate less than maximum is possible for a 
given number of functional units. This design procedure is easily applied 
to much larger algorithm graphs more representative of actual signal pro­
cessing and control problems.

VI. CONCLUSION

A new model useful for understanding the relationship between decom­
posed algorithms and data flow architectures has been presented. Named 
ATAMM for Algorithm to Architecture Mapping Model, the model consists of 
Petri net marked graphs called the algorithm marked graph, the node marked 
graph, and the computational marked graph. After establishing that the 
computational marked graph is live, safe and consistent, graph time 
performance measures of time between input and output (TBI0), task time 
(TT), and time between outputs (TBO) were defined. Then lower bounds for 
the performance measures were calculated analytically from the modified 
algorithm graph and the computational marked graph. A design strategy for 
achieving optimum time performance was proposed and illustrated with a 
design example.

Simulation tools and an actual hardware prototype have been developed 
to test and validate the ATAMM model. The simulation software package [17] 
consists of a PC-based computer model of the CMG. Algorithms are entered to 
the package by specifying the algorithm marked graph, and simulation output 
consists of a graphical display of the movement of tokens. An accompanying 
diagnostic software package [18] automatically computes and displays
performance measures and other performance data. A hardware prototype [19] has also been constructed to validate the ATAMM operating rules and to provide a benchmark for testing the simulation software. The architecture is shown in Figure 17 and is one of several candidates which could be used to perform concurrent operations according to the ATAMM rules. A primary motivation for this particular design was the availability of hardware. The system consists of S-100 crates having a 16-bit CPU card, multiple serial I/O channels, and 32K memory. A personal computer is used to host the system and to download algorithm graph descriptions to the system. A number of decomposed algorithms, including those presented here, have been investigated using these tools.

Continuing research is designed to generalize the ATAMM model and is focused in three main areas. The present model assumes that all functional units are identical and that each is able to perform all primitive operations. An important extension is to model the situation where there are two or more different groupings of processors where each group is able to perform only a subset of the required primitive operations. The present model represents only decision-free algorithms. Another important extension is to develop the capability to admit algorithms containing data-dependent branching points. Finally, methods for decomposing algorithms which result in good performance are being studied in the context of the ATAMM model.
REFERENCES


Figure 1. Algorithm marked graph for discrete system equation.
Figure 2. ATAMM node marked graph model.
Figure 3. ATMM computational marked graph model for discrete system equation.
Figure 4. ATAMM model components.
Figure 5. Modified algorithm graph for Figure 1.
Figure 6. Operating strategy implementation.
Figure 7. Algorithm graph for design example.
Figure 9. Graph play with TBO=3 and unlimited functional units.
Figure 10. Resource utilization envelope for design example.
Figure 11. Graph play with TBO=4 and no control edges.
Figure 12. Resource envelope overlay diagram with TBO=3.
Figure 13. Resource envelope overlay diagram with TBO=3.5.
Figure 14. Resource envelope overlay diagram with TBO=4.0.
Figure 15. Example algorithm graph performance analysis summary.
Figure 16. Performance margin for example algorithm.
Figure 17. Prototype hardware configuration for ATAMM validation.
The development of a new graph theoretic model for describing the relation between a decomposed algorithm and its execution in a data flow environment is presented. Called ATAMM, the model consists of a set of Petri net marked graphs useful for representing decision-free algorithms having large-grained, computationally complex primitive operations. Performance time measures which determine computing speed and throughput capacity are defined, and the ATAMM model is used to develop lower bounds for these times. A concurrent processing operating strategy for achieving optimum time performance is presented and illustrated by example.
End of Document