5.1 INTRODUCTION

Advancements in large-scale integration technology have brought about new design considerations for future computer systems. Since hardware costs have become relatively small compared with software development and maintenance costs, design considerations for future computer systems may feasibly exploit novel architectures which are not hardware-constrained in an effort to enhance performance and to reduce software costs. Research in the areas of distributed processing [1]-[3], [18] and configurable computer system [4] [5] provided examples of such efforts to enhance performance by distributing tasks of a computation into a set of parallel processing elements and restructuring the hardware resources according to the needs of the program being executed. Also in the areas of theory of computation and software engineering a considerable amount of interest has been focused on the properties of computations [6]-[8] and ways of using these properties in a performance optimal manner [9], [10], [13], [14]. A leading consideration in this area has been the potential parallelism that may exist within the structure of a given computation [11], [12]. A logical outgrowth of research in these above areas may be future systems which will take advantage of structural properties of computation to achieve higher performance (shorter execution time).

In this paper a modeling framework is developed which is applicable to the study of the structural properties of computations. The model uses an expression format similar to Boolean logic expressions, but including a parallel construct. Algorithms are developed to extract parallel operations and compute expected execution time for computations. The speedup in expected execution time of a sample set of C programs is estimated by using the above results to restructure the programs in a more parallel form.

5.2 MODELING OF COMPUTATIONS

A. Computation Structure

A computation structure [14] consists of two parts, control and data, each represented by a directed graph called a precedence graph and a data flow graph, respectively.
A data flow graph consists of a set of nodes (containing a set S, of storage cells and a set P, of operators) and a set of edges indicating the input and output storage cells needed by the operators. Set S₁ and S₀ represent the initial and final value data cells, respectively. Fig. 1 illustrates a typical data flow graph.

The content of a cell is represented by using a bracket about its label. An operation can be defined as \( F_n ([A], [B]) \rightarrow C \) this means that the content of cell A and content of cell B are operated upon by operator a and the result is stored in cell C. Operators are either functional (have an output cell(s) or decisional (have no output data cells).

A precedence graph consists of a set of nodes, \( N_c \), containing operators, an entry node, an exist node, and a set of directed edges between nodes indicating the desired flow of control. An AND operator \( \wedge \) is used to synchronize parallel flow of control in which the flow can proceed to the next operator (s) only when all operations prior to the AND operator have been completed. An OR operator \( \vee \) is used to join the flow of nonparallel control edges, where control is passed to the next operator(s) if any one of the inputs is activated. Thus a precedence graph provides information about the execution ordering of the operators. Fig. 1 shows a sample precedence graph.

---

**Date flow graph**

![Data Flow Graph](image)

**Precedence Graph**

![Precedence Graph](image)

---

\[
\begin{align*}
Nd &= \{S, P\} \\
S &= \{A,B,C,D,E,F\} \\
P &= \{a,b,c,d,e\} \\
S_1 &= \{A,B\} \\
S_0 &= \{E\} \\
N_c &= \{a, b, c, d, e, ENTRY, EXIT\}
\end{align*}
\]

Fig. 1 Computation structure model.  
(a) Data Flow graph (data structure)  
(b) Data Flow graph (control structure)
B. Computation Expression

A computation expression models the same information as that of the computation structure by using a control expression and a set of data expressions. A control expression models the control flow of the computation, and the data expressions specify the data requirements for each operator.

1. Control Expression: Let \( A = \{a, b, c, \ldots\} \) be a set of distinct operator names in a computation. Control expression, \( CE \), is formed by combining \( A \) with a set of relational operators \( \{., v., ^, <, >, ()\} \) as explained below. Also \(+\) and \(-\) are concatenated to decision operator names to indicate alternative branching conditions, as shown in Example 1 below.

The cascade operator \( ., a.b.e. \) (henceforth abc) models the structure:

\[
\begin{array}{c}
\text{a} \\
\downarrow \\
\text{b} \\
\downarrow \\
\text{c}
\end{array}
\]

The operator on the left has higher precedence than those on the right. The cascade operator is associative, but is not commutative.

The OR operator \( v \) is used to model a merge of alternate control paths. The control will be passed to the right of the expression if the last operator in any \( a \ v \ b \ v \ c \) models the following control structure:

\[
\begin{array}{ccc}
\text{a} & \downarrow & \text{b} \\
\downarrow & & \downarrow \\
\text{v} & & \text{c}
\end{array}
\]

The AND operator \( \wedge \) is used to model a merge of parallel control paths. Control will be passed to the right only if the expression of operators in all paths have completed their execution.

\( a \wedge b \wedge c \) models the following control structure:
Both OR and AND operators are commutative and associative.

The loop operator $\Rightarrow$ models a loop in a control structure.

$\langle P \rangle = \lambda \lor P \lor PP \lor \ldots$ where $\lambda$ is the null expression. The loop operator is the same as the star operator in a regular expression [15].

The parentheses ( ) define the execution precedence for the control subexpressions. A parenthesized subexpression in a control expression would have higher precedence over term to its right.

Theorem 1: Control expression $CE$, an expression generated by combining a set of distinct operator names $A = \{a, b, c, \ldots\}$ with the relational operators $\{., \lor, \land, \Rightarrow, (\rangle$, is a regular expression.

Proof: The only difference between the definition of a regular expression and the above is the AND operator $\land$. Consider the expression $P \land Q$, where $P$ and $Q$ are regular expressions. Thus finite state automata $s_p$ and $s_q$ exist which will recognize $P$ and $Q$. Since $(P \land Q)$ can be recognized by a composite automaton $s_{pq}$ whose output is specified by the logical AND function of the outputs of $s_p$ and $s_q$, $P \land Q$ is also a regular expression.

Example 1: The CE for Fig. 1 is

$$CE = a < (b \land c) e \rightarrow (B \land c) e + d$$

2) Data Expression: A data expression $DE$ is a set that specifies the input and output data cells required for operation $i$.

$$DE_a = \{I_a, O_a\}$$

$I_a$ is the set of input data cells.

$O_a$ is the set of output data cells.
Example 2: For the data graph in Fig. 1

\[ \text{DE}_a = \{(A,B), (C)\} \]
\[ \text{DE}_b = \{(C), (C)\} \]
\[ \text{DE}_c = \{(B,C), (D)\} \]
\[ \text{DE}_d = \{(D), (E)\} \]
\[ \text{DE}_e = \{(B,D), \emptyset\}, \text{where } \emptyset = \text{the empty set.} \]

C. Generation of Control Expressions

The algorithm below develops a set of simultaneous expression equations which may be solved by recursive substitution.

Algorithm 1 – Generation of Control Expressions :

Step 1: Uniquely label all edges, \( i \), in the control structure.

Step 2: Obtain a partial control expression for each edge in the control structure, as shown below, where \( E_i \) is the partial control expression at edge \( i \).

Rule 1: Generation of the initial expression.

\[ E_i = \lambda \text{ where } \lambda \text{ denotes a null symbol.} \]

Rule 2: Generation of a serial expression.

\[ E_2 = E_1 \ a \]
Rule 3: Generation of a parallel expression.

\[ E_3 = E_1 \land E_2 \]

Rule 4: Generation of an OR expression

\[ E_3 = E_1 \lor E_2 \]

Rule 5: Generation of a loop expression.

\[ E_3 = E_j \lor E_{a-b} \]

\[ E_j = E_i \lor E_j \]
\[ E_i (a-b) \]
\[ E_k = E_i + = E_i <a-b> a + \]
Step 3: Starting at edge where a control expression is desired, recursively substitute all partial control expressions until no edge label is contained in the expression.

5.3. STRUCTURAL PROPERTIES OF COMPUTATIONS

This section develops a set of properties of computations that are useful in establishing conditions for the exploitation of parallelism in control structures.

A. Equivalence of Computations

To realize alternate serial/parallel configurations that may achieve the same computation, it is desirable to establish a notion of computation equivalence. Let the state of a computation be the content of its data cells. Let the history of a data cell be the sequence of values it possessed during the computation [13].

Definition 1: Two computations are data equivalent if

(a) their data flow graphs have equivalent data cells and operator functions; and
(b) for every selected initial state, both computations achieve the same final state and the same history for all of the data cells.

It should be noted that this notion of equivalence requires the data portion of the computation to be fixed. Other less-constrained definitions are possible, but are not considered in this paper.

B. Control Structure Properties of Computations

Let the following relations be defined for a pair of operators, (a,b) in a computation.

Definition 2:

\[ a \parallel b \quad a, b \text{ are independent, i.e.,} \]
\[ O_a \cap (I_b \cup O_b) = \phi \]
\[ \text{and} \]
\[ O_b \cap (I_a \cup O_a) = \phi; \]
\[ a \text{ and } b \text{ are in parallel;} \]
\[ a \succ b \quad a \text{ may precede } b \text{ for correct operation;} \]
\[ a - b \quad a, b \text{ are on alliterative control paths.} \]
1) Serial Control Structures:

Definition 3: A computation, \( C \), is deterministic if, for each pair of parallel operators \((a,b)\), \( a,b \subseteq C \), \( a\parallel b \) is true.

Observation: A serial computation is deterministic since no parallelism is present.

Theorem 2: Any equivalence-preserving transformation of a computation from serial to parallel form preserves determinacy.

Proof: Assume serial computation \( C_s \) is transformed into parallel computation \( C_p \) and that \( C_p \) is nondeterministic. Then an operator pair \((x_i \wedge x_j)\) exists, where \( x_i \neq x_j \) and \( x_i \wedge x_j \). Thus either

\[
O_i \cap (I_i \cup O_j) \neq \emptyset \text{ or } O_j \cap (I_i \cup O_j) \neq \emptyset.
\]

If \( O_i \cap O_j \neq \emptyset \) or \( O_j \cap O_i \neq \emptyset \) or \( O_i \cap O_j \neq \emptyset \),

The contents of \( O_i, O_p, O_i \wedge O_j \), respectively, depend on the execution order of \( x_i \) and \( x_j \), and may assume one of two values. Thus \( C_p \) and \( C_s \) are not equivalent.

Lemma 1: If, for a pair of operators \((a,b)\) on the same control path \( a\parallel b \), ten either \( e>b \) or \( b>a \) must exist to preserve determinacy.

Lemma 2: A pair of operators, \((a,b)\) where \( a>b \), \( b>a \), \( a\parallel b \) all exist, may be placed in parallel, denoted \( a\wedge b \). These properties provide a basis for restructuring serial control structures into parallel form.

2) Loop Control Structures:

Theorem 3: Restricting internal loop operators to internal loop positions is necessary to ensure computation equivalence when the operator semantics are unknown.

Proof: Assume an operator were moved outside the loop. Clearly there exists a loop exit (data) condition for which the quantity of executions of the operator would be different from in the original form. Thus, the history of the operators’ (or a dependent operator’s) data cell(s) may be different, causing inequivalence. If operator semantics were considered, a rare exception might exist.
Theorem 4: Let $L$ be a loop, $a_i$ CL and $p \subset L$ where $a_i$ and $p$ are operators. Then $P \land L$ if $f p \land a_i, \forall a_i \in XL$.

Proof: Obvious.

From these properties it is evident that we may first view each largest independent loop as a single operator during the analysis of a computation. Each nested loop may then be analyzed in a recursive or interactive fashion.

3) Decision Control Structures:

Theorem 5: Decision-dependent nondecision operators on one decision control path may not be moved to a decision independent position.

Proof: Since nondecision operators in a decision control path are conditionally executed, their output data cells are conditionally modified. By moving a nondecision operator to outside a decision control path, the output data cells for this operator then become unconditionally modified. Hence erroneous states may result for the computation, creating non-equivalence.

5.4 PERFORMANCE PROPERTIES OF COMPUTATIONS

Expected execution time is selected as a performance measure. First we develop a method of determining the expected execution time of a computation, and then consider the evaluation of alternative control structures.

A. Expected Execution Time, $E(T)$

Let each operator, $a_i$, be represented by a vector $a_i$.

$$a_i = \begin{bmatrix} t(a_i) \\ p(a_i) \end{bmatrix}$$

Where $t(a_i)$ is the expected execution time contribution of $a_i$, $P(a_i)$ is the conditional probability of $a_i$ executing given that its immediate predecessors(s) have completed.

Let each decision be modeled as a discrete probability distribution where each exist $j$ is assigned $p_j$ and $\sum p_j = 1.0$.

1) Determination of Operator Probabilities:

a) $p(a) = 1$ if the immediate predecessor of "a" is a nondecision operator;

b) if the immediate predecessor of "a" is a decision operator and "a"
follows decision outcome i;

(i) \( p(a) = p \), if outcome i is on a feedback (looping) path;
(ii) \( p(a) = P_i/P_T \), if outcome i is on a nonfeedback path where \( P_T \) is the total probability of leaving the loop or decision.

Refer to fig. 2 for sample operator probability calculations.

![Figure 2. Operator probability determination](image)

2) Calculation of \( E(T) \):

The expected execution time of a computation will be calculated by sequentially processing a CE in reverse polish form. To aid the calculation procedure, we first insert a "null operator" with zero execution time immediately following each decision exist which immediately leads to a single \( \land \) or \( \lor \) operator. (This removes the need for special cases in the algorithm).

\[ J=[0,1], \quad K=[1,0] \]

Then, the expected execution time contributions for subexpressions can be calculated according to the following rules.

a) Serial Control Expression: Let \( a_1, a_2, \ldots, a_n \) be a serial control expression.
The time cost equivalent composite operator is

\[ M = \left[ \frac{E(T)}{\prod_{i=1}^{n} J_{a_i}} \right] \]

b) Parallel Control Expression: Let \( (a_1 \land a_2 \ldots \land a_n) \) be a parallel control expression. Assume that each operator has fixed execution time.

\[ M = \left[ \frac{E(T)}{\prod_{i=1}^{n} J_{a_i}} \right] \left[ \max_{i=1 \to n} \left( \frac{k_{a_i}}{J_{a_i}} \right) \right] \]

The time cost equivalent composite operator is

\[ M = \left[ \frac{E(T)}{\prod_{i=1}^{n} J_{a_i}} \right] \]

c) Or Control Expression: Let \( a_1 \lor a_2 \lor \ldots \lor a_n \) be an OR control expression

\[ E(T) = \sum_{i=1}^{n} K_{a_i} \]

The time cost equivalent composite operator is

\[ M = \left[ \frac{E(T)}{\sum_{i=1}^{n} J_{a_i}} \right] \]

d) Loop Expression: Let \(<a>\) be the loop, where \( a \) represents the entire iterative portion. (Note that the nonfeedback operators are still represented in the expression after the decision brackets and will yet have another time contribution.)

\[ E(T) = \frac{k_a}{1 - J_a} ; \quad m = \left[ \frac{E(T)}{i} \right] \]
(In reverse polish form let the unary loop function be represented by \#.)

Algorithm 2 Calculation of expected execution time

Step 1 - Initialization
   a) Add null operators where required.
   b) Obtain the reverse polish form of the CE.
   c) Terminate the CE with $.

Step 2 – Scan the CE from left to right. If the leftmost symbol is
   a) Operator, push the operator on a stack and go to step 2:
   b) $, go to Step 3.
   c) $, go to Step 3.

Step 3: Pop operator M from the stack

Example 3 (from Fig. 2):

Original Expression:

\[ a < b - c - e > (b + d - v - c + f) $ \]

Reverse Polish Form:

\[ abc . e . # . bd . bc . f . v . $ \]

Result:

\[
M = \begin{bmatrix}
  l_{t_1} p_2 & l_{t_1} p_2 & (l_{t_1} + l_{t_1} + l_{t_1}) \\
  l_{t_{1+1}} & l_{t_{1+1}} & (l_{t_{1+1}} + l_{t_{1+1}}) \\
  l_{t_{1+1}} & l_{t_{1+1}} & (l_{t_{1+1}} + l_{t_{1+1}}) \\
  1 & 1 & 1
\end{bmatrix}
\]
B. Optimal Solution Structures

Assume that a computation, C, is given in serial form (no parallelism), and labeled such that operator \( a_i > a_j \) for \( i < j \).

For computation C, we can represent the operator relation \( > \) (precedence), \( \& \) (parallel), or \( - \) (alternative path) with a precedence table \( P \), a triangular array with a unique entry for each unique operator pair \( (a_i, a_j) \), where \( a_i, a_j \) are the row and column designation, and \( i < j \).

Definition 4: A computation, C, is in basic form if it realizes \( P \), where \( P \) contains the maximum number of \( \& \) relations.

Theorem 6: The basic form of a computation is unique.

Proof: Note that each operator pair may have only one unique relation. From Definition 2, the relation \( | \) is independent from all operators, but the pair in question. Thus, since C is given in a unique form, the conversion of \( > \) relations to \( \& \) relations is unique. Thus, the basic form of a computation realizes a comparison of unique relations and is itself unique.

Definitions 5: A computation, C is in \( n \)-parallel form if its control structure has, at most, \( n \) operators in parallel.

Lemma 3: \( n \)-parallel form is not unique, if \( n < \) maximum degree of parallelism (\( m \)).

Theorem 7: Basic form is a sufficient, but not a necessary condition for minimum expected execution time.

Proof:

Sufficiency: Clearly, since basic form is unique and realize all possible
parallel executions of operators, a maximum overlap of component operator execution occurs to ensure minimum expected execution time.

**Necessity:** By example let the basic form of C be $CE = (t_1 \cdot t_2) \land t_3$, where all $t_i$ are equal. $CE = t_1 \cdot (t_2 \land t_3)$ has less parallelism, but both $CE$ and $CE$ have execution time $2t_i$. Therefore, basic form is unnecessary to ensure minimum expected execution time.

**Definition 6:** An $n$-parallel set is a set of $n$ operators such that every pair of operators in the set may be in parallel. A maximum parallel set is a set which will not remain a parallel set if any new operator is added to set.

It is practical to consider solution structures optimal if they achieve a minimum execution time, but have less than the maximum degree of parallelism.

**Definition 7:** An optimal solution structure is a control structure with parallelism of $n \leq m$ which yields minimum execution time.

**Definition 8:** An $\alpha$ solution structure is a control structure which has a degree of parallelism of $n$, where $\alpha$ is an ordered set of numbers indicating the sequential variation of degree of parallel during execution.

For example, given a 4-parallel set $S = \{01, 02, 03, 04\}$, if 3-parallel form is desired, one may generate the following $\alpha$-solution structures:

a) $\{3,1\}$ solution structure:

$$(01 \land 02 \land 03) \land 04$$

b) $\{3,2\}$ solution structure

$$(01 \land (02 \land 03)) \land 04$$

**Definition 9:** An $\alpha$-solution structure family is a set of $\lambda$-solution structures

$\alpha = \{3,1\}$

$\alpha = \{3,2\}$

Definition 9: An $\alpha$-solution structure family is a set of $\lambda$-solution
structures which have identical control structure skeletons.

For example, a \{3,1\} solution structure family \((m=4, n=3)\) contains 4 members, of which 4 are unique.

\((01 \land 02 \land 03). 04, (01 \land 02 \land 04). 03
\)

\((01 \land 03 \land 04). 02, (02 \land 03 \land 04). 01.\)

Theorem 8: Given a set of \(m\) parallel operators and a desired amount of parallelism \(n, n<m\), the number of precedence relations (\(\triangleright\)), \(r\), is bounded by

\[
\frac{(m-n)(2m-n)-n\left(\frac{m^2}{n}\right)}{2} \leq r \leq \frac{m(m-1)-n(n-1)}{2}
\]

Proof: Proof. is established by constructing and assessing the \(\lambda\)-solution for each bound.

Upper bound: By construction, the largest quantity of precedence relations occurs when \(n\) operators are in parallel and \((m-n)\) operators are in series: \(CE=(x_1 \land ... \land x_m). x_{n-1} ... x_m\). The number of precedence relations is \(n(m-n)\) between parallel and serial operators.

\[
\frac{(m-n)(m-n-1)}{2}
\]
between serial operators.

Thus,

\[
n(m-n) + \frac{(m-n)(m-n-1)}{2}
\]

\[
= \frac{m(m-1)-n(n-1)}{2}
\]

Lower bound: By construction, the lower bound occurs when \(n\) parallel strings of serial operators are used, such that \(m=q_n+s\), where \(q\) is the largest integer \(q = \left[\frac{m}{n}\right]\). Now \(s\) string have \(q+1\) operators each, and the other \(n-s\) strings have \(q\) operators each. In this case the number of precedence relations is
\[
q(q-1) = 2 \sum_{i=0}^{n-1} \left( \frac{m}{n} \right)^{i} \quad \text{for } n \text{ strings of } q \text{ operators}
\]

Thus,

\[
(m - qn)q = (m - \left[ \frac{m}{n} \right]n)\left[ \frac{m}{n} \right] \quad \text{for the additional } S \text{ operators.}
\]

Observation : If the execution times were equal for all \( m \) operators, then any \( \alpha \)-solution structure family with the lower bound of precedence relations would be an optimal solution structure. This is true since the number of precedence relations and the amount of parallelism achieved are inversely related. Having the lower bound of precedence relations thus achieves a maximum amount of parallelism.

Theorem 9 : Assume that the execution times of the \( m \) operators are unknown and unequal. A \( \alpha \)-solution structure family with a lower bound of precedence relations yields minimum \( T_{\text{avg}} \):

\[
T_{\text{avg}} = \frac{\sum_{i=1}^{m} t_{ai}}{m!} \quad \text{where } t_{ai} \text{ is the execution time}
\]

for an \( \alpha \)-solution structure family member, and \( m! \) is the total number of possible members.

Proof : Consider the lower bound \( \alpha_{i} \)-solution structure and any second \( \alpha \) 2-solution structure. Order the \( \alpha_{i} \) and \( \alpha_{2} \) family members such that for each \( \alpha \) 2 member there corresponds a unique \( \alpha_{i} \) member where \( \alpha_{i} \) contains at least the same parallel relations as \( \alpha \) 2. Since each structure has \( m! \) members, and the lower bound structure has more parallel pairs, the ordering is possible. Now, clearly, \( t_{\alpha_{1}} \leq t_{\alpha_{2}} \) for each ordered pair, since \( \alpha_{1} \) contains the same and perhaps more parallelism.

Selecting an optimal member from a lower bound \( \alpha \)-solution structure family is known scheduling problem which is NP-hard [16], [17].
5.5 Extraction of Parallelism

In this section algorithms are presented to reconstruct a given serial control structure into a parallel form. This algorithms are organized as

Algorithm 3 : Analysis of parallelism.
Algorithm 4 : Synthesis of basic form.
Algorithm 5 : Synthesis of N-parallel form.

The approach used in Algorithm 3 is to first convert the given control structure to a free form, and then form the precedence table (section 5.4) maximizing the parallel (\(\land\)) operator relations. Algorithms 4 and 5 ten reconstruct a control expression in either basic form, or n-parallel form if a restriction to n parallel processing units is desired.

A. Transformation Rules

The following control structure transformation rules are useful in carrying out the algorithms in Section 5.5B. Each rule is presented with its appropriate control expression format. Let a,b,c be a subset of operators or operator subexpressions.

Rule 1 : Serial to parallel transformation

\[ a \cdot b \rightarrow a \land b \iff O_b \cap (I_a \cup O_a) = \phi \]
and

\[ O_a \cap (I_b \cup O_b) = \phi \]

Rule 2 : Parallel to serial transformation

\[ a \land b \rightarrow a \cdot b \text{ or } b \cdot a \]

Rule 3 : Right distributive transformation

\[ (a \lor b) \cdot c \leftrightarrow a \cdot c \lor b \cdot c \text{, where } c_i \text{ is a copy of } c. \]

Rule 4 : Associative transformation
B. Algorithms

Definition 10: A decision level number is an integer associated with each operator to denote which decision the operator most closely depends upon in a computation. Fig. 3 illustrates assignments of decision level number in a control structure.

1) Algorithm 3: Analysis of Parallelism

This algorithm assumes that a serial control structure has been defined, and that the data requirements are known for each operator. Note that a loop expression may contain one or several next lower level loop expressions, each of which might contain next lower level loop expressions, etc. We suggest that
the readers refer to the next example while reading through this algorithm.

Step 1 – Formulation of Tree Structure: Store the serial control expression at the root (level 1) of a tree structure to be constructed now. Scan the expression and generate a lower level tree node to store each next loop expression, including links between each node and its parent node. Repeat until lowest level loop expressions have been represented as individual nodes (leaf nodes).

Step 2 – Assignments to Decision Level Numbers to Operators: PERFORM the following procedure for expressions in all tree nodes.

If (expression = loop expression).
{
CONSIDER only the expression within the <> pair.
a) Initialize the decision level number to zero.
b) Scan the expression from left to right.
c) if a (or a + or – is detected, increment the decision level number by 1. If a) is detected, decrement the decision level number by 1.
d) The decision level number at the time an operator is scanned is then the decision level number for this operator.
}
else
{
a) Initialize the decision level number to zero.
b) Scan the expression from left to right.
c) If "is detected, increment the decision level number counter by 1.
d) The decision level number at the time an operator is scanned is then the decision level number for this operator.
}

Step 3 : Apply the right distributive transformation to each nodal control expression, including the exit operator.

Step 4 : FOR each node, prepare a precedence table, P, as described in Section 5.4, and specify its entries as follows:

a) Foreach P (a,p, a_j) entry

1) If α v β is in the node expression, where α, β are subexpressions
   and a_i ∈ α
   AND a_j ∈ β (this occurs when a_p a_j are on alternative paths)
Specify \( P(a_i, a_j) = \)

2) IF \( a_i \) is a decision operator
   AND \( a_j \) is a nondecision operator
   AND \( n_j > n_i \), where \( n_j, n_i \) are decision level numbers,
   Specify \( P(a_i, a_j) \Rightarrow \) (this preserves an operator's decision dependence)

3) IF \( P(a_i, a_j) = \phi \) (not yet specified)
   AND \( a_j \) is an EXIT operator
   Specify \( P(a_i, a_j) = > \)

4) IF \( P(a_i, a_j) = \phi \)
   AND \( 0_{a_i} \land (I_{a_j} \lor 0_{a_j}) = \phi \)
   AND \( 0_{a_j} \land (I_{a_i} \lor 0_{a_i}) = \phi \)
   Specify \( P(a_i, a_j) = || \)
   ELSE \( P(a_i, a_j) = > \) (this ensures determinacy)

b) Determine the maximum set of parallel, \( \wedge \), relation in \( P \) by applying
   the transitive property of \( > \) in the following manner.

\{
   FOR EACH row in \( P(a_i, a_j) \) starting with the bottom row, scan from
   right to left.
   IF \( P(a_i, a_j) = > \) and \( P(a_i, a_k) = > \)
      CHANGE \( P(a_i, a_k) \) to >
   IF \( P(a_i, a_j) = || \)
      CHANGE \( P(a_i, a_j) \) to \( \wedge \)
   IF \( P(a_i, a_j) = - \)
      DO nothing
\}

Procedure table, \( P \) now contains the maximum number of parallel
relations.

Example 4 : Parallelism analysis for the computation in Fig. 4 is as
follows :

Control expression = abc \((-<de_{2f}2>)\) d+e1v+f1g$
Three structure (Step 1) 000 11 1 00

abc (Le, v + f, g$)

Decision level number (Step 2) <d–e₂ f₂> d+

Application of right distributive property (Step 3)

000 1100 100

abc (–Le, g^v + fjg^)

<d–e₂ f₂ > d +

Oil

2) Algorithm 4 – Synthesis of Basic Form: Beginning with a precedence table set from Algorithm 3, this algorithm reconstructs the control expression and / or precedence graph such that basic form is displayed.

Let S = s₁, s₂, ... sₖ be the set of nodes in a tree.

i = a column index for a particular precedence table, P.

j, k = row indexes for P.

Step 1: WHILE S ≠ φ

{ 
  a) SELECT a node, Sᵢ, from S with a precedence table, P.
  b) FIND the immediate predecessor sets Pᵢ₀, for each operator aᵢ as follows.

  { 
    FOR each column i, beginning on the right (i=2)
    Pᵢ₀ = φ
    FOR each aᵦ where P(aᵦ, aᵢ) =>
      Put aᵦ into Pᵢ₀
    If aᵦ is a decision operator, concatenate aᵦ with + or – to denote the dependence of aᵢ
  }
  }

  { 
    FOR each set Pᵢ₀
    If aᵦ, aᵦ ∈ Pᵢ₀ and ak ∈ Pᵢ₀
    REMOVE ak from Pᵢ₀
  }

  Pᵢ₀ = φ

111
c) TO generate a control expression from the $P_{i0}$:
FOR each $a_i$ write a partial control expression:
IF $P_{ai} = \phi$, write $e_{ai} = \lambda$
IF $|P_{ai}| = 1$ and $a_j \in \text{Poi}$, write $E_{ai} = E_{aj}a_j$
IF $|P_{ai}| > 1$ and $a_j \in \text{Poi}$, write $E_{ai} = a_Ea_j$
WRITE $CE = V_{i} e_{Si1}$, where $\{S_i\}$ are the exit operators
SOLVE for $CE$ by iterative substitution.

d) TO generate a graph from the $P_{i0}$:
Define $P_i = P_{ai}$ where the $P_i$ are variable sets
While any $P_i = \phi$
{
SELECT all $a_i$ where $P_i = \phi$ and connect them to the existing graph:
### TABLE 1
MAJOR CHARACTERISTICS OF THE TEST PROGRAMS

<table>
<thead>
<tr>
<th>PROGRAM NAME</th>
<th>NUM OF SAC SIMT</th>
<th>NUM OF LOOPS</th>
<th>% OF SIMT INSIDE LOOPS</th>
<th>% OF FUNCTIONAL STATEMENTS COND.</th>
<th>% OF FUNCTIONAL STATEMENTS UNCOND.</th>
<th>% OF NON FUNCTIONAL STATEMENT</th>
<th>PROGRAM FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM</td>
<td>33</td>
<td>1</td>
<td>15.15</td>
<td>12.12</td>
<td>51.52</td>
<td>36.36</td>
<td>Find the triple exponentia smoothed series s of a given series x.</td>
</tr>
<tr>
<td>TRI</td>
<td>39</td>
<td>4</td>
<td>46.15</td>
<td>20.51</td>
<td>35.90</td>
<td>43.59</td>
<td>Decide types of triangles given 3 sides.</td>
</tr>
<tr>
<td>GRADE</td>
<td>34</td>
<td>4</td>
<td>26.47</td>
<td>14.71</td>
<td>44.11</td>
<td>41.18</td>
<td>Calculate class average and standard deviation based on a list of student names and test grades.</td>
</tr>
<tr>
<td>MDEALC</td>
<td>90</td>
<td>1</td>
<td>15.56</td>
<td>6.67</td>
<td>44.44</td>
<td>48.89</td>
<td>Deallocate storage in a memory management routine.</td>
</tr>
<tr>
<td>COVMA</td>
<td>43</td>
<td>10</td>
<td>34.88</td>
<td>23.26</td>
<td>37.21</td>
<td>39.53</td>
<td>Calculate the covariance matrix.</td>
</tr>
<tr>
<td>EX2</td>
<td>31</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>67.74</td>
<td>32.26</td>
<td>Perform arithmetic and standard deviation operations.</td>
</tr>
<tr>
<td>BINSRCH</td>
<td>30</td>
<td>1</td>
<td>30.00</td>
<td>13.33</td>
<td>36.67</td>
<td>50.00</td>
<td>Perform binary search.</td>
</tr>
<tr>
<td>BSORT</td>
<td>22</td>
<td>2</td>
<td>31.82</td>
<td>13.64</td>
<td>31.81</td>
<td>54.55</td>
<td>Perform bubble sort.</td>
</tr>
<tr>
<td>OSF</td>
<td>33</td>
<td>2</td>
<td>33.33</td>
<td>9.09</td>
<td>63.64</td>
<td>27.27</td>
<td>Perform integration of an equidistantly tabulated function by Simpson's rule.</td>
</tr>
<tr>
<td>DET3</td>
<td>32</td>
<td>2</td>
<td>15.63</td>
<td>12.50</td>
<td>46.88</td>
<td>40.62</td>
<td>Perform differentiation of an equidistantly tabulated function using Lagrangian interpolation formula.</td>
</tr>
</tbody>
</table>

If \( P_{o_i} = \phi \) construct

IF \( |P_{o_i}| = 1 \) \( a_j \in P_{o_i} \) construct
If \(|P_0| > |a_j| \) construct

\[
\{ \text{set of } a_j \}
\]

Delete the select \(a_i\) from all \(p_i\).

SELECT and connect the remaining unconnected operators as above.

e) Delete \(S_d\) from \(S\)

Step 2: SUBSTITUTE the control expressions (and/or subgraphs) from the leaf node to the root node.

Continuation of Example 4:

Immediate Predecessor Sets, \(P_i\) Precedence Graph

<table>
<thead>
<tr>
<th>i</th>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(\phi)</td>
</tr>
<tr>
<td>b</td>
<td>(\phi)</td>
</tr>
<tr>
<td>c</td>
<td>(a, b)</td>
</tr>
<tr>
<td>L</td>
<td>(c^-)</td>
</tr>
<tr>
<td>el</td>
<td>L</td>
</tr>
<tr>
<td>gl</td>
<td>(e_i)</td>
</tr>
<tr>
<td>$1$</td>
<td>(g_i)</td>
</tr>
<tr>
<td>f1</td>
<td>(c^+)</td>
</tr>
<tr>
<td>g2</td>
<td>(c^+)</td>
</tr>
<tr>
<td>$2$</td>
<td>(g_2, f_1)</td>
</tr>
</tbody>
</table>

Control Expression

\[
E_a = \lambda \\
E_b = \lambda \\
E_c = E_{aa} \land E_{aa} \\
E_L = E_{c^-} \\
E_{el} = E_{ll} \\
E_{gl} = E_{el} e_l \\
E_{g1} = E_{g1} g_l \\
E_{f1} = E_c c^+ \\
E_{g2} = E_c c^+ \\
E_{g2} = E_{g2} g_2 \land E_{nn} \\
CE = E_{g1} V E_{g2}
\]
Solving,

\[ CE = (a \land b) \lor e \lor g_1 \lor (a \land b)c + g_2 \land (a \land b)c + f_1 \]

Factoring

\[ CE = (a \land b) c - L e_1 g_1 \lor C + (g_2 \land f_1) \]

3) Algorithm 5-Synthesis of \( n \)-Parallel Form: This algorithm considers the problem of selecting a parallel structure for a computation whose degree of parallelism is \( n \), \( n < \text{max parallelism} \). Starting with a precedence table, and assuming that the operator execution times are unknown, the algorithm creates a modified precedence table which satisfies the bound \( n \) while maximizing the quantity of operator pairs which may be in parallel. The resulting structure may then be generated using Algorithm 4.

The algorithm described below uses a branch-and-bound procedure, which terminates when the first solution (maximum quantity of parallel operator pairs) is found.

LET \( S = \{ S_1, S_2, ... \} \) be set of solution nodes

where \( S_i = \{ \pi_i, C_i, L_i, A_i \} \) is a quadruple,

and \( \pi_i \) = the set of compatible (parallel) operators for node \( i \)

\( C_i \) = a cost integer indicating the total number of operator pairs whose parallel relation was removed to obtain.

\( L_i \) = a lower bound of cost required to obtain a solution using this node or a derivative of this node,

\( A_i \) = the set of operator pairs whose parallel relation were removed.

Let \( L_i = C_i + M_i - n \)

where \( m_i = \max_j \left[ |B_j| \right], B_j \in S_i \)

\( |B_j| \) = the number of operators in \( B_j \).

Step 1 - Initialization: FROM precedence table, \( P \), form the maximum parallel class of operators, \( M \). This is easily done by defining \( \land \) as a compatibility relation using well-known methods.
M = \{B_k\}, \quad B_k = \text{a block of } M.

\text{DEFINE}
\text{S} = \{S_1\}
\pi_j = M
\xi_0 = 0
\gamma_i = \xi_i + m_i - n
\lambda_1 = \phi \text{ (empty set)}.

\text{Step 2 - Node Selection/Testing: CONSIDER each minimum bound node, } S_i \text{ from } S.

\text{IF } \gamma_i = C_i \text{ go to Step 4, } S_i \text{ is a solution.}

\text{IF, for all such nodes, } \gamma_i > C_p \text{ select one and continue.}

\text{Step 3 - Node Expansion:}
\text{a) SELECT a block, } B_j \in \pi_i
\hspace{1cm} 1) |B_j| > n
\hspace{1cm} 2) |B_j| \text{ is minimum.}
\text{b) ASSUME } (B_j = \alpha_1, \alpha_2, ..., \alpha_r)

\text{FOR each } \alpha_n, \alpha_p, i \neq j, \text{ define a new solution node, } S_{k'}, \text{ and add it to } S.

\text{There will be } \binom{r}{2} \text{ such solution nodes.}

\text{TO do this}
\text{1) Consider changing } (\alpha_p, \alpha_j) \in P \text{ to } (\alpha_p > \alpha_j) \text{ and determine all implied operator pair relation changes caused by the transitive property of } > \text{ as follows.}
\hspace{1cm} a) \text{ For all } \alpha_k, k < i \text{ if } \alpha_k > \alpha_p \text{ then } \alpha_k > \alpha_j \text{ is an implied pair.}
\hspace{1cm} b) \text{ For all } \alpha_k, k < i \text{ if } \alpha_k > \alpha_p \text{ then } \alpha_i > \alpha_j \text{ is an implied pair.}
\hspace{1cm} c) \text{ Let } i = \text{ the least upper bound partition of } (\alpha_i, \alpha_j) \text{ and all implied pairs.}
\text{2) } A_k + A_i \cup \{(\alpha_p, \alpha_j) \text{ and all implied pairs}\}
\hspace{1cm} A_k = \overline{\alpha_k} = \overline{\alpha_i} \overline{\alpha_j}
\hspace{1cm} C_k = C_i + 1 + \text{number of implied pairs}
\hspace{1cm} L_k = C_k + m_k - n

\text{Go to Step 2.}
Step 4 – Modification of Precedence Table:

CREATE a modified precedence table from P and the solution node $S_r$. For each operator pair in $A_p$, change that pair relation in P from $\land$ to $\triangleright$.

5.6 EXPERIMENTAL RESULTS

The application feasibility of the analysis results is demonstrated by evaluating the control structure of ten C programs. The operator level for which parallelism was explored was the intermediate language output of a C compiler. Each operator was assigned unity time cost, and fixed and equal decision branching probabilities were used. Each compiled program was then analyzed for its potential parallel content, using the following measures.

### TABLE II

**EXPECTED TIMES AND EXECUTION PERFORMANCE IMPROVEMENTS FOR THE SET OF TEST PROGRAMS**

<table>
<thead>
<tr>
<th>PROGRAM NAME</th>
<th>$E(T)_s$</th>
<th>$E(T)_p$</th>
<th>$P_{\text{max}}$</th>
<th>$S$</th>
<th>$N$</th>
<th>$CC_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM</td>
<td>0.144</td>
<td>0.0815</td>
<td>6</td>
<td>1.769</td>
<td>25</td>
<td>2.5</td>
</tr>
<tr>
<td>TRI</td>
<td>6.137</td>
<td>4.354</td>
<td>7</td>
<td>1.410</td>
<td>1404</td>
<td>1.27</td>
</tr>
<tr>
<td>GRADE</td>
<td>2.508</td>
<td>2.667</td>
<td>4</td>
<td>1.315</td>
<td>706</td>
<td>1.19</td>
</tr>
<tr>
<td>MDEALC</td>
<td>0.092</td>
<td>0.0595</td>
<td>5</td>
<td>1.546</td>
<td>19</td>
<td>1.7</td>
</tr>
<tr>
<td>COVMA</td>
<td>0.041</td>
<td>0.025</td>
<td>8</td>
<td>1.64</td>
<td>8</td>
<td>2.0</td>
</tr>
<tr>
<td>EX@</td>
<td>0.120</td>
<td>0.046</td>
<td>11</td>
<td>2.609</td>
<td>11</td>
<td>6.73</td>
</tr>
<tr>
<td>BINSRCH</td>
<td>1.413</td>
<td>1.146</td>
<td>6</td>
<td>1.233</td>
<td>904</td>
<td>0.295</td>
</tr>
<tr>
<td>BSORT</td>
<td>170.938</td>
<td>134.495</td>
<td>4</td>
<td>1.271</td>
<td>701</td>
<td>52.00</td>
</tr>
<tr>
<td>QSF</td>
<td>4.288</td>
<td>3.983</td>
<td>8</td>
<td>1.077</td>
<td>316</td>
<td>0.09</td>
</tr>
<tr>
<td>DET3</td>
<td>2.763</td>
<td>2.747</td>
<td>4</td>
<td>1.006</td>
<td>10</td>
<td>1.66</td>
</tr>
</tbody>
</table>

$E(T)_s$ = the expected execution time of the serial form of the computation,
$E(T)_p$ = the expected execution time of the basic form of the computation,
$P_{\text{max}}$ = the maximum degree of parallelism found,
The characteristics of the sample set of programs tested are shown in Table 1.

A Interprocessor Communications

Interprocessor communications cost, $CC$, was evaluated by assuming that each sequential set of operators (straight-line segment) was assigned to a processor, such that interprocessor communications only occur for each execution of an AND node. On this basis, we may define an upper bound, $CC_m$, on a practical interprocessor communication cost as

$$CC_m = \frac{E(T)s - E(T)p}{N}$$

Where $N = \text{expected number of interprocessor communication that occur during execution}$. A cost of $CC_m$ would produce zero performance improvement in terms of execution time. Table II illustrates the potential speedup for the sample programs the cost upper bound. It can be seen that a wide range of speedup factors occur (from 2.609 to 1.006), indicating that algorithm semantics/selection is probably a significant factor in achieving a highly parallel organization.

5.7 CONCLUSION

An expression model for evaluating and extracting parallelism in control structures was developed. Algorithm for extracting and achieving variable degree of parallelism were presented, as well as for evaluating the expected execution time performance of a computation. An experimental application of the results was described in which a sample set of C programs was analyzed and their potential performance assessed in a hypothetical parallel environment. The modeling results have application in a variety of distributed processing situations.
5.8 REFERENCE


