MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo No. 51

Informal Outline of a Theory of
Parallel Computation Schemata for Simple Algorithms

Jack B. Dennis

September 1970

This research was done at Project MAC, MIT, and was supported in part by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01), and in part by the National Science Foundation under grant GJ-432.
Informal Outline of a Theory of Parallel Computation Schemata for Simple Algorithms

Jack B. Dennis

Work on computation schemata in the Computation Structures Group has evolved from the thesis research done by Van Horn [7], Rodriguez [5], Luconi [3], and Slutz [6] at Project MAC, and has been considerably influenced by the original studies of Yanov [8] and, more recently, the work of Karp and Miller [2], and the work of Paterson [4]. Two questions have been of greatest interest to us: What sort of constraints must be met in representation of parallel computations so that unique results of computations may be guaranteed? Under what conditions is it possible to determine whether two representations (schemata) describe identical classes of computations? For the class of schemata we have considered, we have satisfactory answers to the first question, and have gained a much better understanding of the second.

A computation schema represents the manner in which functional elements and decision elements are interconnected, and their action sequenced, to define an algorithm. The functional elements of a schema are called operators. Each operator $a$ evaluates some unspecified function of an $m$-tuple of input variables and assigns values to an $n$-tuple of output variables.

$$g(a): \{m\text{-tuples}\} \rightarrow \{n\text{-tuples}\}$$
The unspecified function associated with an operator \(a\) is denoted by \(g(a)\). The decision elements of a schema are called **deciders**: Each decider \(d\) tests some unspecified predicate \(p(d)\) for an \(m\)-tuple of input variables

\[
\begin{align*}
\text{input cells} & \quad \overset{\uparrow}{\text{d}} \quad \overset{\text{p(d)}}{\downarrow} \\
1 & \quad \vdots \\
\vdots & \\
m &
\end{align*}
\]

\(p(d) : \{m\text{-tuples}\} \rightarrow \{\text{true, false}\}\)

A computation schema has two parts — a data flow graph and a control. The data flow graph defines the interconnections through which results of each operator application are passed on as arguments for further transformations and tests. The variables of a schema are represented in the data flow graph by boxes called **cells**. There is also a circle for each operator and a diamond for each decider. Directed arcs join the operators to their output cells and represent the connections to each operator and decider from its input cells.

A data flow graph:

- **Input cells**
  
  \(I = \{m_1, m_2\}\)

- **Output cells**
  
  \(O = \{m_4, m_5\}\)
The cells of the schema are identified by the letters \( m_1, m_2, \ldots \). Certain cells are designated as input or output cells. Values are assigned to the input cells before a computation begins; upon completion, the result is the set of values present in the output cells. Several operators, say \( a \) and \( b \), may have the same associated function letter: \( g(a) = g(b) \). In this way, a schema may require that two operators, \( a \) and \( b \), always implement the same transformation, although the particular transformation is unspecified. Similarly, each decider \( d \) designates a predicate letter \( p(d) \). The function letters and predicate letters of a schema make up two finite sets \( G \) and \( P \).

The control of a computation schema is a specification of the sequences in which operators and deciders are permitted to act. In particular, the control will indicate how further progress of computations by the schema is affected by the results of actions by the deciders. A variety of different representations have been used for the control in the literature on the subject. We shall begin by considering properties of the set of action sequences permitted by the schema. Let \( A \) be the set of operators. For each decider \( d_i \), let \( t_i \) stand for an action by the decider for which the outcome is \( \text{true} \); let \( f_i \) stand for an action by decider \( d_i \) for which the outcome is \( \text{false} \). Let \( T = \{ t_1, t_2, \ldots \} \), \( F = \{ f_1, f_2, \ldots \} \). Then any string of letters in the set

\[
V^* \quad \text{where} \quad V = (A \cup T \cup F)
\]

might be a control sequence of the schema. By the control set of the schema, we mean the set of those sequences in \( V^* \) that are permitted by the control.

Before further characterizing properties of control sets let us consider a
few examples of schemata. For these examples we shall represent the control through the use of precedence graphs [1].

\[ S_1: \text{data flow graph} \]

\[ \begin{array}{c}
\text{precedence graph} \\
\text{begin} \\
\text{end}
\end{array} \]

This is an elementary schema because there are no deciders in the data flow graph. A sequence in the control set of \( S_1 \) must contain each node (operator instance) in the precedence graph in an order consistent with the partial ordering indicated by the graph.

\[ C_1: \begin{cases}
(a_1, a_2, a_3, a_1, a_4, a_5) \\
(a_1, a_3, a_2, a_1, a_4, a_5) \\
(a_1, a_2, a_3, a_4, a_5) \\
(a_1, a_3, a_2, a_4, a_1, a_5) \\
(a_1, a_3, a_2, a_4, a_1, a_5) \\
(a_1, a_2, a_4, a_3, a_1, a_5) \\
(a_1, a_2, a_4, a_3, a_1, a_5) \\
(a_4, a_1, a_2, a_3, a_1, a_5)
\end{cases} \]
When deciders are present, it is no longer true that all sequences in the control set are of the same length.

\[ S_2: \text{ data flow graph} \]

\[ \text{precedence graph} \]

The diamond nodes in the precedence graph connect to two subordinate precedence graphs that specify alternative computations according as the designated decider has a true or false outcome. Operator \( a_2 \) in the data flow graph is an identity operator; the associated function \( f(a_2) \) is always the identity function.

\[ C_2: \left\{ \begin{array}{c}
(a_1, a_2, f_1, a_3, a_4, a_5) \\
(a_1, a_2, f_1, a_4, a_3, a_5) \\
(a_1, a_2, t_1, a_6, f_2, a_3, a_5) \\
(a_1, a_2, t_1, a_6, t_2, a_4, a_5)
\end{array} \right\} \]
Since no iteration is present, the control set \( C_2 \) is finite.

Iteration is indicated in a precedence graph by a pie-shaped node connected to a subordinate precedence graph.

\[ S_3: \text{ data flow graph} \]

![Data Flow Graph](image)

The computation specified by the subgraph is repeated until the decider acts with a false outcome. The control set is \( C_3 \).

\[
C_3 = \{ (a_1 f_1), (a_1 t_1 a_2 a_3 f_1), (a_1 t_1 a_3 a_2 f_1), (a_1 t_1 a_2 a_3 t_1 a_2 a_3 f_1), \ldots \}
\]

\[
C_3 = a_1 ([t_1 (a_2 a_3 U a_3 a_2)]^\infty) f_1
\]
To convert a computation schema into a specification of a particular algorithm it is necessary to specify the functions and predicates designated by the function letters in $G$ and the predicate letters in $P$. Of course, the specified functions and predicates must have domains and ranges consistent with the topology of the data flow graph, and in agreement wherever the value of a function may be the argument of a function or predicate. Such a specification of functions and predicates is called (after Yanov) an interpretation of a schema.

Two properties of schemata are of particular interest to us. A schema $S$ is determinate* if, for any interpretation of the function and predicate letters, $S$ determines a functional relation of output tuples to input tuples. In order to say whether two schemata $S_1$ and $S_2$ describe the same computations, we must be able to relate the interpretations of the function and predicate letters in $S_1$ and $S_2$. For this purpose let

$$G = G_1 \cup G_2 \quad P = P_1 \cup P_2$$

Then $S_1$ and $S_2$ are equivalent* schemata if, for any interpretation of the functions and predicate letters in $G$ and $P$, $S_1$ and $S_2$ determine precisely the same relation of output tuples to input tuples. (Note that equivalence applies to nondeterminate schemata as well as to determinate schemata.)

The following schemata illustrate the definition of determinacy.

*These definitions are weaker than those used by Karp and Miller in [2], hence our results on determinacy and equivalence are stronger than would follow by application of their work.
Schema $S_4$ is not determinate because the sequences

$$\alpha = (a_1, a_2, a_3) \quad \beta = (a_2, a_1, a_3)$$

are both in $C_4$ but correspond to different compositions of functions

$$\alpha: \quad m_4 = g_3(g_1(m_1, m_2), g_2(g_1(m_1, m_2), m_2))$$

$$\beta: \quad m_4 = g_3(g_1(m_3, m_2), g_2(m_1, m_2))$$
It is easy to choose functions to associate with the function letters so the functions defined by these compositions are distinct. Schema $S_5$ is determinate even though the sequences

$$\alpha = (a_1 a_3 a_1 a_2) \quad \beta = (a_1 a_3 a_2 a_1)$$

leave different values in cell $m_3$. This is because the final value in cell $m_3$ does not participate in determining the final value put in the output cell $m_4$. Yet sequences $\alpha$ and $\beta$ determine the same composition of functions as the relation of output values to input values.

$$\alpha:\quad m_4 = g_3(g_1(m_1, m_2))$$
$$m_2 = g_2(g_1(m_1, m_2))$$
$$m_3 = g_1(m_1, m_2)$$

$$\beta:\quad m_4 = g_3(g_1(m_1, m_2))$$
$$m_2 = g_2(g_1(m_1, m_2))$$
$$m_3 = g_1(m_1, g_2(g_1(m_1, m_2)))$$

To develop insight into the question of determinism and equivalence we have devised the notion of data-dependence graph or dadep graph for short. A dadep graph of a schema sets forth separately each action by an operator or decider. For a particular control sequence of a schema the final value placed in each output cell will be the result of some cascaded composition of functions. A dadep graph is just a graph representation of the cascade composition associated with each output cell.
Let us construct the dadep graph for schema \( S_6 \) from its unique control sequence \( \alpha = (a_1 \ a_2 \ a_3 \ a_4) \).

We start by setting down a copy of each input cell of the schema. (The letters denoting input cells are underlined.) Then we add a copy of an operator and its output cells for each succeeding element of the control sequence:
Each cell added to the dadep graph is labelled as in the data flow graph and this label is erased from the cell copy previously bearing it. In the case of an identity operator, a second label is given to the latest copy of its input cell, and no copy of the operator is made.

For schemata that include deciders, there will be a cascade composition of functions associated with each action of a decider as well as each output cell. A determinate schema with k deciders could have $2^k$ distinct dadep graphs -- one for each combination of decisions that might occur in the course of some computation. For the schema $S_2$ there are just three dadep graphs because a decision of false by $d_1$ results in the absence of any action by $d_2$. 

(a) 

(b) 

(c)
In general a schema that represents an iteration defines an infinite set of dadep graphs. In the case of $S_3$ the three simplest dadep graphs are:

(a) 
(b) 
(c)

Nondeterminate computation is represented by a schema when there is a cell that could be assigned a value by one operator either before or after a value is assigned to or read from the cell by another operator or decider action. When this can happen we say the schema has conflict.
Let $O(a)$ denote the set of output cells of an operator $a$, and let $I(x)$ denote the set of input cells of $x$ which may be an operator or a decider. Then there is conflict between $a$ and $x$ if there are sequences

$\omega \vdash a \vdash \phi$ and $\omega \vdash x \vdash \psi$

in the control set of a schema such that

$O(a) \cap I(x) \neq \emptyset$

There is conflict in both schemata $S_4$ and $S_5$, as demonstrated by these pairs of control sequences:

$S_4: \begin{array}{c}(a_1 \ a_2 \ a_3) \\ (a_2 \ a_1 \ a_3)\end{array} \quad S_5: \begin{array}{c}(a_1 \ a_3 \ a_1 \ a_2) \\ (a_1 \ a_3 \ a_2 \ a_1)\end{array}$

$O(a_1) \cap I(a_2) = \{m_1\} \quad O(a_1) \cap I(a_2) = \{m_3\}$

$O(a_2) \cap I(a_1) = \{m_2\}$

Schemata whose control sets can be represented by precedence graphs share two important properties: Whenever there is the possibility of two actions proceeding concurrently, the occurrence of one of the actions does not inhibit or block the other. This property is called persistence. Expressed as a property of the control set, persistence requires that, if

$\omega \vdash x \vdash \omega$ and $\omega \vdash y \vdash \omega$
are prefixes of some control sequences, then

\[ \omega \vdash xy \text{ and } \omega yx \vdash yx \]

are also prefixes of some control sequences. The second property is this:

If either of two actions may occur before the other, then the order in which they occur has no effect on the subsequent course of the computation.

In terms of control sequences

\[
\begin{align*}
\omega \vdash xy & \quad \omega \vdash yx \\
\vdash \psi \quad \vdash \psi & \in C
\end{align*}
\]

implies

\[
\begin{align*}
\omega \vdash yx & \quad \omega \vdash xy \\
\vdash \psi & \in C
\end{align*}
\]

A schema satisfying this condition on its control set is said to be **commutative**.

Any schema (as in the examples) with a control that can be represented by a precedence graph is both persistent and commutative. (The converse is not true, however.)

By an adaptation of known methods it is not difficult to show that any computation schema that is persistent, commutative, and free of conflict is guaranteed to be determinate. A more interesting problem is to determine the circumstances for which the conflict-free property is a necessary condition for schemata to be determinate. We have found two natural restrictions on schemata sufficient that any determinate (and persistent, commutative) schema is necessarily conflict free. The first of these restrictions amounts to requiring that each action by any operator or decider in a schema participate in determining some output value. A schema meeting this restriction is said to be **normal**. The second restriction disallows control sets that permit repetition of a computation or test for the same m-tuple of input values.
A schema meeting this restriction is said to be repetition free.

To be normal, a schema must first be well defined. That is, in any situation that an operator or decider may act, all of its input variables must have been assigned values. The properties well-defined and normal are readily explained in terms of dadep graphs. For a schema to be well-defined, it must have no dadep graph in which some cell that is not a copy of an input cell is not written by some operator. Schema $S_8$ is not well-defined because there is a copy of cell $m_2$ in the dadep graph that is read by not written, and $m_2$ is not an input cell of the schema.

For an elementary schema to be normal, each cell in any dadep graph that is a copy of an input cell, or is written by a copy of an operator must be connected by a directed path through the dadep graph to the final copy of some output cell. Schema $S_9$ is not normal unless cell $m_2$ is considered an output cell.
For a schema with deciders to be normal, each input cell to a copy of a decider in any dadep graph must be considered as a final copy of an output cell in applying the rule given above. Furthermore, each decision must have some effect on the future course of the computation. Consider a partial dadep graph for $S_2$ that is constructed from a prefix of a control sequence such that the next action is a test by decider $d_2$: 
There must be at least two distinct ways of completing the dependency graph, of which one is associated with a true outcome of the test and the other is associated with a false outcome.

Schemata $S_1$, $S_2$, and $S_3$ are all normal. They are also determinate and conflict-free. Nevertheless, it is possible for a normal schema to have conflict and still be determinate.
Schema $S_{10}$ generates two distinct dadep graphs corresponding to the two orders in which $a_1$ and $a_3$ may act. Yet the two dadep graphs define the same composition of functions.

$$g_2(g_1(x_1, x_2), g_1(x_1, x_2))$$

This is possible because the function designated $g$ is applied twice to the same input values. This is known as an operator repetition. A schema may also exhibit a decision repetition.
Schema $S_{11}$ has these dadep graphs:

(a) \hspace{8cm} (b) \hspace{8cm} (c) \hspace{8cm} (d)

There is conflict between operators $a_1$ and $a_2$ at cell $m_4$; however, the two dadep graphs (b and c) generated by the conflict cannot represent any computations because the repeated tests of $p_1$ must yield identical outcomes.
It is evident that repetitions are not especially useful in representing algorithms. Hence we feel justified in restricting our attention to repetition-free schemata. We have been able to show that in any schema that is persistent, commutative, normal and repetition free, the conflict-free property is necessary for the schema to be determinate.

For an elementary schema that is well defined, normal, repetition-free and determinate, all execution sequences yield the same dadep graph. In fact the dadep graph is a canonical form for this class of schemata. Thus the equivalence of any two elementary schemata can be tested by constructing their dadep graphs.

In the case of a normal, repetition free schema that has deciders but no iteration, the class of computations represented is described by a finite set of dadep graphs, as shown for the schema $S_2$ earlier. Each pair of input values will be processed as shown in that one of the dadep graphs for which
the evaluation of predicates agrees with the truth values given at decision points of the graph.

We can construct a table of two columns, called a conditional expression list, that characterizes the computations represented by a schema. Each row of the table corresponds to one data graph. In the left-hand column we write a disjunction of the predicates that must be satisfied by the input variables for the corresponding data graph to describe the computation. In the right-hand column we write the compositions of functions that specify the corresponding dependence of output values on input values. For \( S_2 \) we have:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \overline{p}_1(x_2) )</td>
<td>( g_5(g_2(g_1(x_1, x_2)), g_3(g_1(x_1, x_2))) )</td>
</tr>
<tr>
<td>( p_1(x_2) \cdot \overline{p}_2(g_4(x_2)) )</td>
<td>( g_5(g_1(x_1, x_2), g_3(g_1(x_1, x_2))) )</td>
</tr>
<tr>
<td>( p_1(x_2) \cdot p_2(g_4(x_2)) )</td>
<td>( g_5(g_2(g_1(x_1, x_2)), g_1(x_1, x_2)) )</td>
</tr>
</tbody>
</table>
Now consider the schema

\[ S_{13}: \]

The dagap graphs of \( S_{13} \) are

(a) \hspace{1cm} (b) \hspace{1cm} (c) \hspace{1cm} (d)
and the corresponding conditional expression list is:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{p}_1(x_2) \cdot \overline{p}_2(g_4(x_2))$</td>
<td>$g_5(g_2(g_1(x_1, x_2)), g_3(g_1(x_1, x_2)))$</td>
</tr>
<tr>
<td>$\overline{p}_1(x_2) \cdot p_2(g_4(x_2))$</td>
<td>$g_5(g_2(g_1(x_1, x_2)), g_3(g_1(x_1, x_2)))$</td>
</tr>
<tr>
<td>$p_1(x_2) \cdot \overline{p}_2(g_4(x_2))$</td>
<td>$g_5(g_2(g_1(x_1, x_2)), g_1(x_1, x_2))$</td>
</tr>
<tr>
<td>$p_1(x_2) \cdot p_2(g_4(x_2))$</td>
<td>$g_5(g_1(x_1, x_2), g_3(g_1(x_1, x_2)))$</td>
</tr>
</tbody>
</table>

This table specifies the same class of computations as does Table 2, for we have the logical equivalence

$$\overline{p}_1(x_2) = \overline{p}_1(x_2) \cdot \overline{p}_2(g_4(x_2)) + \overline{p}_1(x_2) \cdot p_2(g_4(x_2))$$

Thus schemata $S_2$ and $S_{13}$ are equivalent. In general, noniterative schemata may be tested for equivalence by constructing their conditional expression lists.

Since an iterative schema has an infinite set of dadep graphs, its conditional expression list is infinitely long. For schema $S_3$ we have:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{p}_1(g^2_1(x_2))$</td>
<td>$\overline{g}_1(x_2)$</td>
</tr>
<tr>
<td>$p_1(g^2_1(x_2)) \cdot \overline{p}_1(g_3(g^2_1(x_2)))$</td>
<td>$g_2(x_1, \overline{g}_1(x_2))$</td>
</tr>
<tr>
<td>$p_1(g^2_1(x_2)) \cdot p_1(g_3(g^2_1(x_2))) \cdot \overline{p}_1(g_3(g_1^2(x_2)))$</td>
<td>$g_2(x_1, g_2(x_1, \overline{g}_1(x_2)))$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We can show that, in general, two schemata are equivalent if and only if their conditional expression lists agree in the sense illustrated by our demonstration of equivalence for $S_2$ and $S_{13}$. When the lists are finite the existence of a decision procedure is clear. At this time we do not know whether or not a decision procedure can be found for the general equivalence problem of persistent, commutative schemata.

References


