Computer Aided Hardware Design by Space-Time Mapping

by

Anneli Avatare

Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, SWEDEN
Computer Aided Hardware Design by Space-Time Mappings

by
Anneli Avatar

November 1989

SICS, Swedish Institute of Computer Science
Box 1263
S-164 28 KISTA
SWEDEN
Abstract
This is a thesis work for the M.Sc. degree in mathematics and computer science at the University of Stockholm, accomplished at SICS, Swedish Institute of Computer Science, Kista during the summer and autumn 1989.

This thesis deals with hardware synthesis by space-time mappings. It is a method to map the steps in algorithms to distinct events in a space-time. If the pattern of an algorithm is known in advance then the algorithm can be scheduled exactly beforehand and the algorithm can be implemented directly in the hardware.

When it is known where and when all steps in the algorithm will be executed then it can be determined where cells shall be placed and how they shall be connected to each other.

I have developed a prototype, an Interactive Space-Time Scheduler, that can be used as an aid to schedule algorithms manually in the space-time.

The user interface is developed in the Xerox/Interlisp-D environment.

Sammanfattning

Rapporten handlar om hårdvarusyntes med rum-tid avbildningar. Det är en metod att schemalägga delstegen i algoritmer till olika händelser i rum-tiden. Om mönstret för en algoritm är känt i förväg kan algoritmen schemaläggas exakt i förväg och implementeras direkt i hårdvaran.

När det är känt var och när alla steg i algoritmen skall exekveras kan man bestämma var celler skall placeras i rummet och hur de skall sammankopplas.

Jag har utvecklat en prototyp, en Interactive Space-Time Scheduler, som kan användas som ett hjälpmedel för att schemalägga algoritmer manuellt i rum-tiden.

Användargränssnittet är utvecklat i Xerox/Interlisp-D miljön.
Table of contents

1. Introduction ............................................................................................................. 3

2. Hardware synthesis by space-time mappings .................................................... 4
   2.1 Static algorithms ............................................................................................... 4
   2.2 Data dependence graphs .................................................................................. 5
   2.3 Single assignment programs ........................................................................... 6
   2.4 Space-time mappings ....................................................................................... 6
   2.5 Basic constraints ............................................................................................. 7
   2.6 Derivation of a synchronous system that implements
       a static algorithm ............................................................................................. 7

3. Systolic arrays ........................................................................................................ 12
   3.1 The basic principle of systolic arrays ............................................................. 12

4. The Interactive Space-Time Scheduler ............................................................... 15
   4.1 How to describe the algorithm to be implemented ......................................... 15
   4.2 Data dependence analysis ............................................................................. 17
   4.2.1 Depth-first search spanning forest ............................................................. 18
   4.3 Dependence graph .......................................................................................... 19
   4.4 Manual scheduling .......................................................................................... 19
   4.5 Space-time mapping ....................................................................................... 21
   4.6 Projection ......................................................................................................... 22
   4.7 Simulation of the scheduling ......................................................................... 23
   4.8 Graphical user interface .................................................................................. 23

5. Conclusions and further research ........................................................................ 26

Acknowledgements ................................................................................................... 29

References .................................................................................................................. 31

Appendix A: An example of a grammar

Appendix B: Users Guide to the ISTS
1. Introduction

This is a thesis work for the M.Sc. degree in mathematics and computer science at the University of Stockholm.

This thesis deals with hardware synthesis by space-time mappings.

There are algorithms, where the same operations are performed independent of input data, for example algorithms for multiplication with dense matrices. Such algorithms can be implemented directly in synchronous hardware. The algorithms can be described by *data dependence graphs* and they may be written in a single assignment language. Every node in the graph represents a unique partial calculation of the algorithm.

The algorithms can be scheduled exactly beforehand, because the pattern of the execution is known in advance. We can establish exactly where and when the steps in the algorithm shall be performed. The implementation in synchronous hardware can be found with the help of space-time mappings, a method to map the steps in an algorithm to distinct events in a space-time, telling where and when the steps shall be performed. The space is a set of coordinates where cells can be placed. Every time one or more assignments may take place at a cell.

The space-time mapping must satisfy two basic constraints:

- Data must be produced before consumed.
- No two actions mapped at the same event

When we know where and when all the steps in the algorithm will be executed, then we can determine where cells must be placed and how they shall be connected to each other.

My task was to develop a prototype, the *Interactive Space-Time Scheduler* (ISTS) that can be used as an aid to schedule algorithms manually in the space-time. The algorithms are defined in a simple single assignment language and every single assignment represents a node in the *data dependence graph*. I have done a graphics user interface where the user can place the nodes manually in a space-time coordinate system. Every partial calculation in the algorithm gets a space-time coordinate.

When the space-time mapping of the algorithm is complete then the ISTS derives a description in form of cells in every space-point, where an event is ever performed and communication links with delay between the cells.
2. **Hardware synthesis by space-time mappings**

There are many algorithms where the same operations are performed independent of input data. Those algorithms can be implemented directly in hardware. The algorithms can be implemented very effectively, because they need a minimum of control at the execution. The implementation in synchronous hardware can be found with the help of space-time mappings.

### 2.1 Static algorithms

Some algorithms, or parts of an algorithm, are static, i.e. the same operations are performed every time the algorithm is executed and the operations are performed independent of the input data.

Static algorithms can be programmed in common imperative programming languages, for example Pascal, provided that jumps and loops where we don't know how many times the loop is iterated are not allowed. IF-THEN-ELSE sentences are not allowed if we don't know in advance if the condition is true or false. LOOP sentences where we don't know in advance how many times the sentences are executed are not allowed either.

**Example 2.1: Static algorithms:**

```plaintext
(READ x)
x := x + 2
y := sin(x)
```

Regardless of the value of the input x, the two assignments will always be performed and the algorithm is accordingly static.

```plaintext
s := 0
FOR i := 1 TO 5 DO
    s := s + a(i)
```

It can be predicted for which values of i the FOR-LOOP is executed and the algorithm is accordingly static.

Other examples of static algorithms are:
- Multiplication of dense matrices
- Fast Fourier Transform
- Some signal and image processing algorithms.
Example 2.2: Non-static algorithms

```
(READ n)
IF n>5 THEN sentence1 ELSE sentence2
```

When \( n \) is unknown, it cannot be predicted which of the sentences that will be performed. Accordingly the algorithm is non-static.

```
(READ n)
s := 0
FOR i := 1 TO n DO
    s := s + a(i)
```

When \( n \) is unknown, it cannot be predicted for which values of \( i \) the FOR-LOOP is iterated. Accordingly the algorithm is non-static.

Non-static algorithms can make use of special properties of the input. For example an algorithm for sparse matrix multiplication, which takes notice of if many elements are zero, is non-static.

2.2 Data dependence graphs

A static algorithm can be described by a directed acyclic graph (see figure 2.1).

![Directed acyclic graph](image)

Figure 2.1: Directed acyclic graph

Every node represents a unique partial calculation. An arc between two nodes indicates that they are dependent of each other, the output data from one node is input data to the other node. This graph is usually named data dependence graph. In figure 2.1 there is an arc from node-1 to node-3. That means that the calculation in node-1 must be performed before the calculation in node-3. Nodes which produce values must be performed before nodes which consume these values.

The graph must be acyclic. Otherwise, suppose that there is an arc from node-3 to node-1 in figure 2.1. Node-3 cannot perform its calculation before the input from node-1 is produced but node-1 can neither perform its calculation before the input data from node-3 is produced. Node-1 is waiting for node-3 and node-3 is waiting for node-1. This is impossible. Therefore it cannot be a cycle in the graph.
The data dependence graph contains information about the parallelism in the algorithm. Calculation in different nodes can be performed in parallel if there is no path between the nodes, i.e. they are independent of each other. The calculation in node-4 and node-5 in figure 2.1 can be performed if suitable in parallel, because there is no path between them.

2.3 Single assignment programs
A way to describe algorithms closely related to data dependence graphs is to use single assignment programs. Such programs consist of assignments, i.e. a variable is assigned a value. In opposite to common imperative programming languages, where a variable can be assigned different values several times, a variable in a single assignment program can only be assigned a value once during the program execution.

One single assignment corresponds exactly to one node in the data dependence graph. Suppose that we have two single assignments s1 and s2. If s2 uses the variable produced by s1, then there is an arc from s1 to s2. It is therefore suitable to describe static algorithms with single assignment languages. The single assignment languages have the advantage of simpler semantics. Single assignment programs can simply be considered as equation systems that define the values of all assigned variables. The values of the variables which are not assigned, but present at the right side of some assignment are input data to the program.

Example 2.3: The following single assignments

\[
\begin{align*}
x & : 2 \\
y & : z \\
u & : x + y
\end{align*}
\]

can be seen as a system of equations

\[
\begin{align*}
x & = 2 \\
y & = z \\
u & = x + y
\end{align*}
\]

the values of x, y and u are uniquely defined, given a value for z.

The order in which the assignments are written is not important, because the sequence of the execution is given by the data dependence between the assignments.

2.4 Space-time mappings
We know the pattern of the execution in advance for a static algorithm. Therefore it can be scheduled exactly beforehand, i.e. we can establish exactly where and when each step in the algorithm shall be performed. From this we can design a computer system that supports the scheduling, i.e. looks for that all the steps in the algorithm are performed at desired time and place, established by the scheduling. Consequently the first step when designing a system that implements a static algorithm, can therefore be to schedule all the steps in the algorithm.
A method of scheduling is to map the steps in an algorithm to distinct events in a space-time (telling where and when the steps shall be performed). We call this scheduling a space-time mapping. A space is a set of coordinates where processor elements or cells can be placed. Every time one or more assignments may take place at a cell. If the cells are controlled by a global clock signal, so that we have a synchronous system, we can talk about a global discrete time expressed as a number of elapsed clock cycles. The time is represented by a natural number. The set of pairs <time,space-coordinate> forms a space-time. Accordingly a two-dimensional space will give a three-dimensional space-time. For example a two-dimensional surface, like a layer of a chip, corresponds to a two-dimensional space.

2.5 Basic constraints
The space-time mapping of a data dependence graph \( <V,E> \), where \( V \) is a set of nodes and \( E \) is a set of arcs, can be described as a function \( F: V \rightarrow T \times R \). The function \( F \) is sometimes expressed as two functions, a time-mapping \( F_t: V \rightarrow T \) and a space-mapping \( F_r: V \rightarrow R \). \( F(v) = <F_t(v), F_r(v)> \), telling when and where \( v \) will be performed.
F must adhere to the following constraints:

- If there is an arc from \( v_1 \) to \( v_2 \), then \( v_1 \) must be performed before \( v_2 \), \( F_t(v_1) < F_t(v_2) \). Data must be produced before consumed.

- \( F \) must be 1-1, i.e. two different actions are not allowed to be mapped to the same event: \( F(v_1) <> F(v_2) \), when \( v_1 <> v_2 \).

2.6 Derivation of a synchronous system that implements a static algorithm
Suppose that we have a space-time mapping of an algorithm, then a synchronous system can be derived by the definition of the following items:

1. Cells
   Locate all space-points where an event is ever performed. Place a cell in all those points.

2. Communication links
   Every arc \( (v_1,v_2) \) in \( E \) needs a link from the cell \( F_r(v_1) \) to the cell in \( F_r(v_2) \). Add a link between the cells with a delay \( F_t(v_2) - F_t(v_1) \). If there already is a link with this delay between \( F_r(v_1) \) and \( F_r(v_2) \), no additional link with the same delay is needed. If \( F_r(v_1) = F_r(v_2) \) then we need a local memory, i.e. a link from the cell to itself. The communication links must have unique names, so the cells know to which links they shall write to and read from respectively.
3. **In-ports**
   Go through all variables and look for those that are not assigned. They are called *free variables*. The free variables provide input data to the system from somewhere outside. Put in-ports to all those cells which use free variables. If an assignment uses more than one free variable, the cell must be provided with an in-port for each free variable.

4. **Out-ports**
   All cells with variables that are output data from the system must be provided with out-ports, one for each variable. In this implementation all variables that are assigned but not used by other assignments are output data from the system. There are also other possibilities.

A system, designed as above will implement the static algorithm correctly and according to the space-time mapping given, if the following is valid:

- The system is provided with input data at the right in-ports and at the right time.

- Each cell performs for every time step its scheduled assignment. The assignment must read its input data from the right in-ports and write its output data to the right out-ports.

- Output data from the system is written at the right out-port at the right time.

A system of this type can be very efficient, since the dynamic control is eliminated. The synchronization between the different partial calculations will be built statically to the system. The data will arrive at the right time to be used.
Example 2.4:  The single assignments are:

\[
\begin{align*}
  x_2 &= x_1 + 4 \quad (1) \\
  y_2 &= y_1 \times x_2 \quad (2) \\
  z_1 &= x_2 + 10 \quad (3) \\
  z_2 &= y_2 - 5 \quad (4) \\
  x_3 &= 7 - z_1 \quad (5) \\
  y_3 &= z_1 \times z_2 \quad (6) \\
  z_3 &= x_3 + y_3 \times 2 \quad (7)
\end{align*}
\]

Those single assignments establish the data dependence graph in figure 2.2.

![Data dependence graph](image)

**Figure 2.2: Data dependence graph**

We see that the graph is acyclic. An example of a scheduling of the graph in figure 2.2 is seen in figure 2.3.

![Space-time mapping](image)

**Figure 2.3: Space-time mapping**
The derivation of the system:

1. Place a cell in every space-point where an event is ever taking place. As we see in figure 2.3 cells must be placed in space-points 1, 2 and 3. We call them p1, p2 and p3.

2. Decide the communication links between the cells and calculate the delay on the links. The set of arcs in the data dependence graph is \{(1,2),(1,3),(2,4),(3,6),(4,6), (3,5),(5,7),(6,7)\}. Now we can derive the communication links between the cells in the system.

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fr(1) = p2</td>
<td>Fr(2) = p1</td>
<td>F_t(2) - F_t(1) = 1</td>
</tr>
<tr>
<td>Fr(1) = p2</td>
<td>Fr(3) = p3</td>
<td>F_t(3) - F_t(1) = 2</td>
</tr>
<tr>
<td>Fr(2) = p1</td>
<td>Fr(4) = p1</td>
<td>local calculation</td>
</tr>
<tr>
<td>Fr(3) = p3</td>
<td>Fr(6) = p2</td>
<td>F_t(6) - F_t(3) = 1</td>
</tr>
<tr>
<td>Fr(4) = p1</td>
<td>Fr(6) = p2</td>
<td>F_t(4) - F_t(6) = 1</td>
</tr>
<tr>
<td>Fr(3) = p3</td>
<td>Fr(5) = p3</td>
<td>local calculation</td>
</tr>
<tr>
<td>Fr(5) = p3</td>
<td>Fr(7) = p2</td>
<td>F_t(5) - F_t(1) = 1</td>
</tr>
<tr>
<td>Fr(6) = p3</td>
<td>Fr(7) = p3</td>
<td>local calculation</td>
</tr>
</tbody>
</table>

There are two links between p3 and p2 with exactly the same delay, thus we need to use only one of them.

3. Decide the in-ports to the system. The free variables are x_1 and y_1. These two variables are used by the assignments in node-1 and node-2. Thus an in-port must be provided at both p1 and p2.

4. The only node that does not transmit its produced value is node-7, producing the variable z_3. p2 must be provided an out-port.

The links must be given unique names, so the cells know to which of the links they shall write to and read from respectively.
The resulting hardware configuration is shown in figure 2.4.

![Hardware configuration diagram](image)

**Figure 2.4: Hardware configuration**

The program in the cells:

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>(&lt;c1,c3&gt;:= in2+4)</td>
<td>skip</td>
</tr>
<tr>
<td>(&lt;m,c2&gt;:= in1*c1)</td>
<td>skip</td>
<td>skip</td>
</tr>
<tr>
<td>m:= m-5</td>
<td>(&lt;c4,m&gt;:= c3+10)</td>
<td>m:= 7-m</td>
</tr>
<tr>
<td>skip</td>
<td>m:= c4*c2</td>
<td>skip</td>
</tr>
<tr>
<td>skip</td>
<td>out:= c4+m*2</td>
<td>skip</td>
</tr>
</tbody>
</table>

Here, \(m\) is a local memory location. \(<m,c2>:= in1*c1\) means that \(p1\) reads the input data from the in-ports \(in1\) and \(c1\). \(p1\) writes the result \(in1*c1\) at the local memory location \(m\) and on the out-port \(c2\). \(skip\) means that nothing is done.

It should also be mentioned that it is possible to base an asynchronous system on a space-time mapping. It will look like the corresponding synchronous system with the same placement of the cells, the same coupling between them and the same sequential program in every cell. The difference is that the communication links will not have a delay that makes the data arrive at the right time, but the communication links need buffers. In an asynchronous system an instruction is performed when all the input data is accessible. The data is transferred with a handshaking mechanism.
3. Systolic arrays

Systolic arrays are a kind of hardware structures that can be suited for VLSI-implementation of regular static algorithms. In a systolic array data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory.

3.1 The basic principles of systolic arrays

A systolic array consists of a set of interconnected cells. Every cell is considered to be able to perform a simple operation. Cells in a systolic array are interconnected in a regular pattern (see figure 3.1). In most cases they are 1-dimensional or 2-dimensional. The cells communicate only with their neighbours, accordingly there are no long links. The data flows from cell to cell in a pipelined fashion. The in-ports and out-ports to the systolic array are at the boundary cells. Systolic arrays are controlled by a global clock signal. In every time step the cells are reading their in-ports, perform their executions and then write the results on the out-ports. The results on the out-ports are in turn input data to their neighbour cells in the next time step.

![Figure 3.1: Systolic array](image)

There are two families of computational tasks: compute-bound and I/O-bound computations. The computation is compute-bound if the number of operations is larger than the total number of input and output elements, otherwise it is I/O-bound. With systolic arrays a wide class of compute-bound computations can be speeded up.

Figure 3.2 illustrates the basic principle of a systolic array. Suppose that the bandwidth between the host and the system is 10 million bytes/second. At least two bytes are read from or written to the host for each operation. The maximum rate will be only 5 million operations/second, no matter how fast the system can operate. If we change the single processing element towards an array of processing elements then we can achieve a higher computation throughput without increasing the memory bandwidth. The data moves through the pipelines in the array.
How can we be ensured that once a data item is brought out from memory it can be used of all cells in the array? The advantage of the systolic approach is that each input data can be used a number of times.

**Example 3.1:** A sequence of weights \( \{w_1, w_2, \ldots, w_k\} \) and the input sequence \( \{x_1, x_2, \ldots, x_n\} \) are given and the result sequence \( \{y_1, y_2, \ldots, y_n\} \) are computed

\[
y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1}
\]

Each \( x_i \) is multiplied by each of the \( k \) weights, the problem is compute-bound. If \( k \) is large and the \( x_i \) is input separately from memory for each multiplication, then the memory bandwidth becomes a bottleneck. A systolic array can solve this I/O bottleneck by making multiple use of each \( x_i \) fetched from memory.

We suppose that \( k=3 \). The weights are preloaded to the cells and stay there throughout the computation. The partial results \( y_i \), move systolically from cell to cell in the left-to-right direction and the inputs \( x_i \) move systolically in the opposite direction. Both the input and the result flow in the system (see figure 3.3). Compare it with classical pipelined systems where only the results flow.
Systolic arrays

Consecutive $x_i$'s and $y_j$'s are separated by one time cycle. Continuing from the situation in figure 3.3, the following is performed during the next five cycles in the array.

\[ \begin{array}{ccc}
  & w_1 & w_2 & w_3 \\
  \text{Cycle 1:} & y_1 := w_1 x_1 \\
  \text{Cycle 2:} & y_1 := w_1 x_1 + w_2 x_2 \\
  \text{Cycle 3:} & y_2 := w_1 x_2 \\
  \text{Cycle 4:} & y_2 := w_1 x_2 + w_2 x_3 \\
  \text{Cycle 5:} & y_3 := w_1 x_3 \\
  & y_3 := w_1 x_3 + w_2 x_4 \\
\end{array} \]

The first output is produced from the system at cycle 3, $y_1$ is output.

Systolic arrays are suitable to synthesize with space-time mapping methods because:

- Systolic algorithms are mostly static so they can be described by a data dependence graph that can be scheduled.

- In a systolic array every cell is usually considered to be able to perform a simple computation, suitably to be described by a single assignment.

- Systolic arrays are synchronous and regular. As they are synchronous, they can be described by a space-time.

- The regularity of systolic arrays leads to that systolic arrays for regular algorithms can be found even if limiting to the classes of simple mappings. This facilitates the searching for optimal mappings. Sometimes the optimization can be automated. See [12].
4. The Interactive Space-Time Scheduler

The task has been to design a prototype, an *Interactive Space-Time Scheduler* (ISTS) that can be used as an aid to schedule algorithms manually in space-time. The ISTS will automatically derive a description of the space-time mapping of an algorithm in form of cells in every space-point, where an event is ever performed and with communication links with delay between the cells. The implementation of the ISTS was carried out according to figure 4.1. The program is written in Common Lisp in the Xerox/Interlisp-D environment and necessary Interlisp-D functions are imported.

![Diagogram of the Interactive Space-Time Scheduler (ISTS)](image)

**Figure 4.1:** Implementation of the Interactive Space-Time Scheduler (ISTS)

4.1 How to describe the algorithm to be implemented

The algorithms are written in a single assignment language. The grammar is found below. It is an experimental simple grammar, that allows only certain types of arithmetical expressions.

Terminal symbols are noted between apostrophes, capital letters represent classes of terminal symbols and non-terminal symbols are written in lowercase letters. Phrases are surrounded with [ ] if they occur 0 or 1 times, * means repetitions of 0 or more times and + means repetitions 1 or more times. ( ) is used for grouping.
The Interactive Space-Time Scheduler

definition ::= 'def' IDENT '=>' assignment-list '.
assignment-list ::= 'begin' assignment
               (,'
               assignment)*
               'end'
assignment ::= variable '=>' expression
expression ::= [SIGN] term (ADDSUB term)*
term ::= factor (MULDIV factor)*
factor ::= variable
       | NUMBER
       | (' expression ')
variable ::= IDENT
IDENT = (letter|digit) (letter|digit)*
NUMBER = digit+
ADDSUB = +| -
MULDIV = * | /
SIGN = + | -

Example 4.1:
A program written in the single assignment language:

```
DEF A = BEGIN
    x := 4,
    y := 3+4*x,
    z := x-y
END.
```

Available is a syntax analyzer that verifies that the program agrees with the
description of the grammar. The output from this parser is a kind of syntax tree, an
internal Lisp representation. The tree for the definition A in the example 4.1 above
can be seen as:

```
(:definition (:ident a)
    (:assignment (:ident x) (:expression (:number 4)))
    (:assignment (:ident y) (:expression (:addition (:number 3)
        (:multiplication (:number 4) (:id x)))))
    (:assignment (:ident z) (:expression (:subtraction (:id x) (:id y)))))
```

When the parser detects an unexpected symbol then an error message is generated
and the parsing is interrupted.
4.2 Data dependence analysis
The data dependence graph is built up from the syntax tree.

Example 4.2:

\[
\begin{align*}
\text{DEF } A &= \text{BEGIN} \\
&\quad x := z + y, \\
&\quad y := z - v, \\
&\quad z := 4 \\
\text{END}.
\end{align*}
\]

The Syntax tree for the program above:

\( \langle \text{:definition (:ident a)} \rangle \)
\( \quad \langle \text{:assignment (:ident x) (:expression (:addition (:id z) (:id y)))} \rangle \)
\( \quad \langle \text{:assignment (:ident y) (:expression (:subtraction (:id z) (:id v)))} \rangle \)
\( \quad \langle \text{:assignment (:ident z) (:expression (:number 4)))} \rangle \)

The :ident variables (for example x) are those variables which are assigned values, i.e. they are assigned the :expression (for example (:addition (:id z) (:id y))). They are defined as nodes. Every node knows from which variables it consumes values, i.e. all the :id (for example z) variables in the :expression expression. Every :id variable knows that it produces its value to the :ident variable in the assignment.

The free variables are those that are not assigned a value. The :id variables in :expression that are not defined as an :ident variable are free variables.

A variable cannot be assigned a value more than once. When a variable is found that already has been assigned a value, the program is interrupted and an error message is given.

In example 4.2, the x, y and z are defined as nodes (see figure 4.2). Node-x uses input data from node-y and node-z. Node-x is not input data to any node. Node-y uses input data from node-z and the value that is produced is input data to node-x. Node-z doesn't use any input data from any node, but it produces input data to node-x and node-y. The only free variable is v and it is used by node-y.

![Figure 4.2: The data dependence in the algorithm in the example 4.2](image-url)
4.2.1 Depth-first search spanning forest

The next step is to examine if the graph is acyclic or not. The following algorithm is used.

The technique is called depth-first search because it continues searching in the forward (deeper) direction as long as possible. See [2]. The vertices and the arcs are visited systematically.

Suppose we have a directed graph \( G \) in which all vertices are initially marked unvisited. Depth-first search works by selecting one vertex \( v \) of \( G \) as a start vertex, \( v \) is marked visited. Then each unvisited vertex adjacent to \( v \) is searched in turn, using depth-first search recursively. Once all vertices that can be reached from \( v \) have been visited, then the search of \( v \) is complete. If some vertices remain unvisited, we select an unvisited vertex as a new start vertex and repeat this procedure until all vertices of \( G \) have been visited.

There are four types of arcs. Tree arcs lead to unvisited arcs during the depth-first search and they form a depth-first spanning forest for the given directed graph. The solid arcs in the graph in figure 4.3 (right) form a depth-first spanning forest for the graph to the left. Back arcs go from a vertex to one of its ancestors in the spanning forest. Forward arcs are nontree arcs and they go from a vertex to a proper descendent. Cross arcs go from a vertex to another vertex that is neither an ancestor nor a descendent.

![Figure 4.3: A directed graph (left) The depth-first spanning forest (right)](image)

How do we distinguish the four types of arcs? We can assign all the vertices a number when they are visited. All the descendents of vertex \( v \) are assigned a number greater than or equal to the number assigned to \( v \). Forward arcs go from low-numbered to high-numbered vertices. Back arcs go from high-numbered to low-numbered vertices. Cross arcs go from high-numbered to low-numbered vertices. Tree arcs are special since they lead to unvisited arcs during the depth-first search.
A template for the algorithm can be seen in figure 4.4, where connected-to[v] are all vertices that v is connected to.

```
FOR v := 1 TO n DO
    mark[v] := unvisited;
FOR v := 1 TO n DO
    IF mark[v] = unvisited THEN
        dfs(v)

PROCEDURE dfs(v:vertex);
VAR w: vertex;
BEGIN
    mark[v] := visited;
    dfsnumber[v] := count;
    count := count + 1;
    FOR each vertex w in connected-to[v] DO
        IF mark[w] = unvisited THEN
            dfs(w)
END;
```

Figure 4.4: A template for the algorithm

This technique can be used to test for acyclicity. If a back arc is encountered during a depth-first search of the graph, then the graph has a cycle.

If there is a cycle in the data dependence graph, then an error message is generated and the user has to check the program in which the algorithm is specified.

### 4.3 Dependence graph

If there is no cycle in the graph, then the algorithm is defined as a *directed acyclic graph* with a fix set of nodes. Every node is represented as a unique partial calculation in form of a single assignment.

### 4.4 Manual scheduling

It is the user only that defines the scheduling of the nodes that define the algorithm by placing the nodes in a space-time coordinate system. The assignments are defined as items in a menu (see figure 4.5). To every assignment belongs a node.

```
Assignments:
X2:=(X1 + 4)
Y2:=(Y1 * X2)
Z1:=(X2 + 10)
Z2:=(Y2 - 5)
X3:=(7 - Z1)
Y3:=(Z1 * Z2)
Z3:=(X3 + (Y3 * 2))
```

Figure 4.5: A menu with assignments
The user makes a choice among the assignments in the menu and places every node in the space-time coordinate system (see figure 4.6). First the user places a node in the space, i.e. *where* the calculation shall be performed. Next the node is placed in the time direction, i.e. *when* the calculation shall be performed. Lines are drawn from the node to the axes, so the location of the node can be seen. Once all the nodes have been placed in the space-time coordinate system, every node knows its position. The scale intervals on the axes and the length of the axes can be changed. The direction of the y-axis can also be changed.

![Prompt print window](image)

**Figure 4.6: The space-time coordinate system**

The system checks automatically that the basic constraints are satisfied, that data must be produced before consumed and that no two actions are mapped at the same event. A node is displayed grey if it uses a value that is not produced yet.

In figure 4.7 the node with the assigned variable $z_1$ is placed incorrectly in relation to the node with the assigned variable $x_2$. $z_1$ uses $x_2$ before $x_2$ has produced its value. If a node is placed where a node already exists, then an error message will be given and the node has to be put in another position.
The dependency in the graph can be displayed as directed arcs between the nodes (see figure 4.8). The scheduling can be optimized by moving the proper nodes, either in the space or in the time direction, or in both directions one at a time.

4.5 Space-time mapping
When all the nodes are placed correctly into the space-time coordinate system we have generated a space-time mapping of the given algorithm (see figure 4.8).
4.6 Projection
When the space-time mapping of the algorithm is complete the projection can take place (see figure 4.9). A cell is placed in every space-point where an event is ever performed. The communication links, with a calculated delay, between the different cells are defined. Only one connection is displayed between two cells even if there are several links with different delays between them. There is only one link with the same delay between two cells. A link with a certain delay is used for all communication between the cells with that delay. The user can get information from a messages window about all links with different delays in a connection that goes from a cell to another. Every link has a unique name for example "1(3 . 4)(2 . 3)". The first digit (1) tells about the delay on the link and the following that the link goes from the cell in the space-point (3 . 4) to the cell in the space-point (2 . 3).

![Figure 4.9: Projection in space](image)

Every cell knows its in-ports and out-ports. The in-ports to the total system are those variables that are not assigned a value. The out-ports from the total system are those variables that are assigned a value but not used by any other assignments. All nodes that are projected on a cell are sorted by the time.

Some information about the synchronous system is saved in a text file. Space-coordinates, in-ports, out-ports and the code are such information saved for each cell. Also in-ports and out-ports to the total system are saved.
4.7 Simulation of the algorithm

To test the algorithm a simulation function can be generated of the algorithm. The sequence of the execution of the assignments are given by the data dependence between the assignments. The free variables are input parameters to the function and the values of the output variables are output data from the function.

4.8 Graphical user interface

The ISTS may be seen as a tool and an aid, when scheduling algorithms. It is important that the user feels that the system is well in hand and that it is possible to decide upon what shall be done next.

In this section some fundamental aspects in Human-Computer Interaction regarding the ISTS are considered.

Feedback: The user can see that a command is appreheended and which consequences a command has all the time.

The cursor is displayed as an hour-glass when an operation will take some moments extra.

Some help about the commands are given.

Reversible actions: The last action is reversible. Maybe the user regrets a performed command when the consequences are displayed.

The last command can always be undone and then the undo command can be undone until a new command is performed.

Abort commands: It is possible to cancel a command that isn't completed.

Interaction forms: The information between the user and the computer can be transferred in different ways. Some different common cases can be distinguished.

*Symbolic interaction* occurs when the user communicates by selecting symbols for the operation and the data.

In the ISTS symbolic interaction occurs when the user selects items in the menus.

*Analog interaction* occurs when a pointing device is used to control a process in the computer.

In the ISTS an analog interaction occurs for example when the user moves one or more nodes and when the user places a node in the space-time coordinate system.
Direct manipulation means that the user can manipulate the object directly with the help of a pointing device. Direct manipulation is often an analog interaction.

In the ISTS direct manipulation occurs when the user moves the axes with help of the mouse. It is immediately displayed how an axis is moved.

The communication between the user and the computer is a kind of dialog.

In the ISTS there are different types of dialogs.

A main menu is displayed continuously. To that main menu there are submenus linked in a hierarchy. A grey triangle indicates that there are several levels of menus available (see figure 4.6).

A pop-up menu is a kind of menu that pops up at special locations and belongs to an object. Each node in the ISTS has a pop-up menu that is displayed when the node is selected. All the commands in this menu are bound to that specific node (see figure 4.10).

![Figure 4.10: Pop-up menu](image)

The system sometimes needs information from the user. The system gives a prompt so that the user knows what kind of information is needed.

In the ISTS this kind of question-answer dialog appears when the system needs information about which file shall be loaded or which file the information shall be saved on. The ISTS writes the question in the ISTS's prompt print window (see figure 4.6).

A number pad menu is used when changing the scale intervals on the axes and when the ISTS requires values for the input-parameters, then such a menu will appear (see figure 4.11).
All single assignments are defined as items in a menu (see figure 4.12). The user chooses assignments by selecting a line with the assignment and places the node in the space-time coordinate system.

**Figure 4.12: Assignment menu**

**Modes:**

A mode is a state in the interaction between the user and the system that decides how the input from the user is going to be interpreted. One ought to avoid modes, but sometimes they are natural.

In the ISTS we can say that the system goes into another mode when nodes are moved and placed, i.e. scheduled. The cursor changes to crosshairs and information and help about what to do is given.
5. Conclusions and further research
The ISTS was developed to investigate if it could be suitable for scheduling algorithms.

The report describes shortly the background to space-time mappings and systolic arrays that are suitable to synthesize with space-time mappings. In the end the ISTS prototype is described.

The current ISTS can be used to schedule algorithms manually in the space-time coordinate system. The algorithms are described in a single assignment language. The grammar is simple and experimental and allows only certain types of arithmetical expressions. From the space-time mappings a description can be derived in form of cells in every space-point where an event is ever performed. Communication links with delay between the cells are defined. For every cell the in-ports, out-ports and the assignments are saved.

For making the ISTS more applicable the following ought to be done in the future:

Single assignment language
The grammar for the single assignment language can be extended. See appendix A for an example.

Parameterized single assignments
Parameterized single assignments would be useful.

Example: multiplication of a 5x5 matrix:

```
DEF matmul5x5 =
BEGIN
  i,j=1,...,5: c(i,j,0):= 0,
  i,j,k=1,...,5: c(i,j,k):=c(i,j,k-1)+a(i,k)*b(k,j)
END.
```

In this example the parameterizing makes it possible to summarize 150 single assignments into two parameterized single assignments.

Parameterized definitions
It would be useful if the definitions could be invoked with parameters. The definition in the example above can for instance be generalized according to the following:

Example: multiplication of a nxn matrix, n>0:

```
DEF matmul(n) =
BEGIN
  i,j=1,...,n: c(i,j,0):= 0,
  i,j,k=1,...,n: c(i,j,k):=c(i,j,k-1)+a(i,k)*b(k,j)
END.
```
The call matmul(5) gives the same result as a call of matmul5x5, but matmul is more flexible because it can be invoked for any n > 0. See [9].

The single assignment editor as an interpreter
The single assignment editor, where the definitions are written, should work in the same way as an interpreter for functional or logic languages:

Then, all the definitions are stored when read. When a definition is called with the current parameters, if any, all the assignments are defined, the assignments menu is created and the ISTS window is opened. This makes it easy to interactively test and modify definitions and to combine them. Definitions could for instance call other definitions.

The code in the cells
The intermediate code that is generated for the cells should operate on in-ports, out-ports and local memory instead of single assignment variables.

See example 2.4, the cell instruction for the single assignment z1 := x2 + 10 is <c4,m> := c3 + 10. It means that the cell reads input data from the in-port c3 and write the result on the out-port c4 and at the local memory location m.

Generating parallel programs and hardware descriptions
From the intermediate code, that has been described above, parallel programs and hardware descriptions can be generated.

OCCAM-program can be generated for every cell if the hardware is to be implemented as an asynchronous transputer network.

VHDL-code can be generated if the system is to be implemented directly in silicon. VHDL is a hardware description language. There are CAD-tools and simulators available that use VHDL-code, that makes it suitable as output format from ISTS since the output then can be used as input to synthesize tools at lower levels.

Automating the scheduling
The scheduling can be automated. The system then finds the space-time mappings of the algorithms without user interaction, or with only partial user interaction.

It is possible to implement for example a method for optimizing systolic arrays with respect to execution time. See [12]. Other automated processes for scheduling can also be thought of.
Conclusions and further research

Interaction
The interaction can be improved for instance by:

- grouping of the nodes into larger objects and by hiding the inner structure, so the user does not have to see the inner structure of that group.

- more general 3-D graphics that permits the user to view the coordinate system from an arbitrary view-point.
Acknowledgements

I am grateful to all who provided assistance during this work. In particular, I would like to thank my supervisors Björn Lisper at SICS, on whose Ph.D. thesis [11] this work was based, and Yngve Sundblad at KTH/NADA.

Many thanks to Björn Backlund and Olof Hagsand, who assisted me at SICS.
References


References


Appendix A

The grammar for the single assignment language can be extended to:

definitions ::= definition '.' (definition)*
definition ::= 'def' IDENT '="' statement'.'
statement ::= assignment
  | IDENT
  | 'begin' statement (',', statement)* 'end'
assignment ::= variable '="' expression
  | variable-list '="' expr-list
expr-list ::= '<' expression (',', expression)* '>
expression ::= i-expr
  | b-expr
i-expr ::= [SIGN] i-term
  | i-expr ADDSUB i-term
i-term ::= i-factor
  | i-term MULDIV i-factor
i-factor ::= variable
  | NUMBER
  | '(' i-expr ')
b-expr ::= 'not' b-term
  | b-expr 'or' b-term
b-term ::= b-factor
  | b-term 'and' b-factor
b-factor ::= variable
  | b-const
  | '(' b-expr ')
  | condition
b-const ::= 'true'
  | 'false'
condition ::= i-expr [RELOPS] i-expr
variable-list ::= '<' variable (',', variable)* '>
variable ::= IDENT

IDENT = (letter|digit) (letter|digit)*
NUMBER = digit+
RELOPS = =|<>|<|>|<=|>=
ADDSUB = +| -
MULDIV = * | /
SIGN = + | -
[This page intentionally left blank]
Users Guide to the
Interactive Space-Time Scheduler

by
Anneli Avatar

November 1989
SICS, Swedish Institute of Computer Science
Box 1263
S-164 28 KISTA
SWEDEN
Table of contents

1. ISTS-The Interactive Space-Time Scheduler .................................................. 3
   1.1 How to use the ISTS ................................................................................. 3
2. Using the single assignment language ............................................................... 4
3. Writing the single assignment program ............................................................. 4
   3.1 Using Assnedit ......................................................................................... 5
   3.1.1 Assnedit menu commands ................................................................. 5
4. Mapping the algorithms in the space-time ......................................................... 6
   4.1 Placing the nodes into the space-time coordinate system ......................... 6
   4.2 Moving a node ....................................................................................... 8
   4.3 Moving nodes in group ......................................................................... 9
5. Displaying the data dependency ...................................................................... 11
6. Displaying the projection in space .................................................................. 11
7. Making a function of the algorithm .................................................................. 12
8. Running the program interactively ................................................................... 13
9. Saving information about the system ............................................................... 13
10. Changing the scale on the axes .................................................................... 14
11. Changing the axes ...................................................................................... 14
12. Undoing last event ..................................................................................... 15
13. Aborting a command .................................................................................. 15
14. Working with several ISTS ......................................................................... 15
15. Help ........................................................................................................... 15
16. A summary of the menus and the commands ................................................ 15
   16.1 The attached menu .............................................................................. 16
   16.2 The node commands ......................................................................... 18
   16.3 The right button default window menu ............................................. 19
1. **ISTS - The Interactive Space-Time Scheduler**

ISTS can be used as an aid to schedule algorithms in the space-time.

1.1 How to use the ISTS

The scheduling is carried out according to the following items:

- When ISTS is started, a window is opened with the x, y and t axes drawn and a menu attached to the window.
- The algorithms are defined in a single assignment language. The assignments are defined as items in a menu. To every assignment belongs a node.
- The nodes are placed into the space-time coordinate system. If all the nodes are placed correctly, then there is a space-time mapping of the algorithm.
- The data dependency in the graph can be displayed as directed arcs drawn between the nodes.
- The space-time mapping can be projected in the space. A cell is placed in every space-point where an event is ever performed. The necessary communication links between the cells are drawn.
- A function can be generated from the algorithm.
- Some information about the system can be saved in a text file.
2. Using the single assignment language

You define your algorithms in the following single assignment language:

\[
\begin{align*}
definition & ::= \ 'def' \ IDENT \ '=>' \\
& \hspace{1cm} assignment-list \ '.'
\end{align*}
\]

\[
\begin{align*}
assignment-list & ::= \ 'begin' \ assignment \\
& \hspace{1cm} (',', assignment)*'end'
\end{align*}
\]

\[
\begin{align*}
assignment & ::= \ variable'=' \\
& \hspace{1cm} expression
\end{align*}
\]

\[
\begin{align*}
expression & ::= \ [SIGN] \ term \\
& \hspace{1cm} (ADDSUB term)*
\end{align*}
\]

\[
\begin{align*}
term & ::= \ factor \\
& \hspace{1cm} (MULDIV factor)*
\end{align*}
\]

\[
\begin{align*}
factor & ::= \ variable \\
& \hspace{1cm} | \ NUMBER \\
& \hspace{1cm} | \ '(expression)'
\end{align*}
\]

\[
\begin{align*}
variable & ::= \ IDENT
\end{align*}
\]

\[
\begin{align*}
IDENT & = (letter|digit) (letter|digit)* \\
NUMBER & = digit+ \\
ADDSUB & = + | - \\
MULDIV & = * | / \\
SIGN & = + | -
\end{align*}
\]

Example:

\[
\begin{align*}
\text{DEF A} & = \text{BEGIN} \\
& \hspace{1cm} x:= 4, \\
& \hspace{1cm} y:= x+4*x, \\
& \hspace{1cm} z:= x-y \\
\text{END.}
\end{align*}
\]

3. Writing the single assignment program

Select one of the sub commands to ASSIGNMENT MENU command.

LOAD FILE: If you already have saved the program in a file, this command will load the file and create an assignment menu. The ISTS will ask you for a filename.
ASSIGNEDIT: A single assignment editor ASSIGNEDIT will be opened and you can write a single assignment program for the algorithm. See the ASSIGNEDIT figure 2.

You place the assignment menu there you want.

You can only present one algorithm to each ISTS window.

3.1 Using ASSIGNEDIT

In the ASSIGNEDIT (see figure 2) you can write your single assignment programs interactively. You define your program for the algorithm and then parse the program, see the menu commands, item 3.1.1.

![Figure 2: ASSIGNEDIT](image)

3.1.1 Asssignedit menu commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>Exits the ASSIGNEDIT. All changes made are aborted.</td>
</tr>
<tr>
<td>PARSE</td>
<td>The parser controls that the syntax of the program is correct. If the syntax of the program is correct the ASSIGNEDIT is closed and the assignments menu appears, if not you have to look over your program again. No error messages are generated. If you want to save the program you must do it before parsing. The ASSIGNEDIT will warn you so you can save the program if you want.</td>
</tr>
<tr>
<td>GET</td>
<td>Loads a new file to edit in the ASSIGNEDIT. The ASSIGNEDIT will ask you for a filename in the prompt print window. If you are working on an unsaved file the ASSIGNEDIT will warn you so you can save the file if you want.</td>
</tr>
</tbody>
</table>
PUT The single assignment program is saved in a file. The ASSIGNEDIT will ask you for a file name in the prompt print window.

4. Mapping the algorithms in the space-time

![Image of the coordinate system](image)

**Figure 3**: Space-Time coordinate system

4.1 Placing the nodes into the space-time coordinate system

The assignments are defined as items in a menu (see figure 4). To every assignment belongs a node. You select an assignment by placing the cursor over a line in the menu and press the left or middle mouse button and then you can place the node in the space-time coordinate system.

```
Assignments:
X: = W
Y: = (X + 4)
Z: = (X * 5)
V: = (Z * Y) + 3
```

**Figure 4**: Assignments menu

When you select an assignment in the menu the cursor changes to crosshairs and the cursor is moved to the ISTS window.
First you place the node in the space:
To start, press and hold the left or middle mouse button and move the cursor. You can see how lines are drawn to the x-axis and the y-axis, the (x,y) coordinates, as you move the cursor. When you are satisfied with the position you release the button. The cursor is still displayed as crosshairs.

Next you place the node in the time direction:
To start, press and hold the left or middle mouse button and move the cursor in the time direction. You can see how a box is stretched out as you move the cursor, you can see where the node is located. When you are satisfied with the time position you release the button and the node is moved to the nearest scale marks on the axes.
Lines are drawn from the node to the axes, so you can see where the node is located. The node has a position (x,y,t).
The assignment is marked so you know that you have placed the node.

When you press the left or middle mouse button at a node then the identifier, the position and the assignment for the node are written in the prompt print window.

Now you can place the rest of the nodes in the same way.
The ISTS checks automatically that the nodes do not use data that is not yet computed. A node is displayed in grey if it is placed illegally (see figure 5).

**Figure 5:** The grey node is placed illegally.
If you place a node where a node already is placed then an error message is given and you must place the node again in a new position.

You cannot place a node outside the coordinate system. The ISTS controls that the node is inside, i.e. in the positive octant, $x \geq 0$, $y \geq 0$ and $t \geq 0$. When you have placed all the nodes correctly into the space-time coordinate system, you will have a space-time mapping of the algorithm.

### 4.2 Moving a node

When you place the cursor over a node and press the middle mouse button then a menu will pop-up (see figure 6), the node commands menu. The node can be moved in both the space and time direction, one at a time, or you can move the node in one direction only. The cursor changes to crosshairs.

![Figure 6: The nodes pop-up menu](image)

If you want to move the node in both space and time: you select the MOVE command.

**First you move the node in the space:**
To start move, press and hold the left or middle mouse button and move the cursor in the space. You can see how the node is moved and you also see the old location of the node. When you are satisfied with the new position in the space then you release the button. The cursor is still displayed as crosshairs.

**Next you move the node in the time direction:**
To start move, press and hold the left or middle mouse button and move the cursor in the time direction. When you are satisfied with the new time position then you release the button. The node is moved to the nearest scale marks on the axes.

Instead of using MOVE command you can place the cursor over a node and press the left mouse button and do as above.

If you select the SPACE command:
The node is only moved in the space as above.
If you select the TIME command:  
The node is only moved in the time direction as above.

4.3 Moving nodes in group

To move nodes in group, you select the SPACE command or the TIME command in the submenu to MOVE NODES command (see figure 7). The nodes can be moved in either the space or the time direction. You select the nodes you want to move in a group according to the alternatives below. You move one of the nodes and then all the other selected nodes are moved the same distance. The ISTS checks that none of the selected nodes will be placed outside the space-time coordinate system.

![Figure 7: The MOVE NODES command](image)

Select the SPACE or TIME command.  
The cursor changes to crosshairs.

There are two alternatives to select nodes to a group:

1. Select nodes by drawing a rubber-band rectangle around the nodes.  
   To start, press and hold the left or middle mouse button and a rectangle is stretched out as you move the cursor. Release the button to stop. The cursor is still displayed as crosshairs. All nodes inside the rectangle are highlighted. Now, move the cursor and select a highlighted node to be moved by pressing the left or middle mouse button at a highlighted node.

If the TIME command was selected:  
To start move, press and hold the left or middle mouse button and move the cursor in the time direction. When you are satisfied with the time position you release the button. Then all the other nodes inside the rectangle are moved the same distance.
If the SPACE command was selected:
To start move, press and hold the left or middle mouse button and move the cursor in the space. When you are satisfied with the position you release the button. Then all the other nodes inside the rectangle are moved the same distance.

2. Select nodes by picking.
Hold the left or right shift key and pick the nodes you want to move by pressing the left or middle mouse button at the nodes. All nodes that you pick are highlighted. The first selected node is used to move the group. You need not remember which the first selected node was the system will do that for you.

If the TIME command was selected: To start move the first selected node, press and hold the left or middle mouse button and move the cursor in the time direction. Release the button when you are satisfied with the position. Then all the other selected nodes are moved the same distance.

If the SPACE command was selected: To start move the first selected node, press and hold the left or middle mouse button and move the cursor in the space. When you are satisfied with the position you release the button. Then all the other selected nodes are moved the same distance.
5. Displaying the data dependency

Select the CONNECT NODES command. The dependency in the graph is displayed, directed arcs are drawn between the nodes (see figure 8).

![Figure 8: Data dependency graph](image)

6. Displaying the projection in space

Select the PROJECTION ? command. This command tells the system to draw or remove the projection in space, it toggles.

When the projection is "on", a cell is displayed in every space-point where an event is ever performed. The communication links between the cells are drawn. Only one connection is drawn between to cells even if there are several links with different delays between them (see figure 9).
If you move the cursor over the point (see figure 9) on the connection and press the left or middle mouse button, then a message window appears. All links and their delays in the connection are written in the message window. Every link has a unique name for example "1(1.2)(3.4)" The first digit indicates the delay on the link and the remaining that the link goes from the cell in the space-point (1.2) to the cell in space-point (3.4) (see figure 10).

![INFO:WINDOW](image)

Figure 10: Information about the lines in the cable

7. Making a function of the algorithm

Select the MAKE FUNCTION command and a simulation function is generated of the algorithm.

The sequence of the execution of the assignments are given by the data dependence between the assignments. The input
variables are input parameters to the function, for example W in the function in figure 11.

```
(CL:DEFUN \DHS-AUTO-0440 (W)
  (LET (X Y Z V)
    (SETQ X W)
    (SETQ Y (+ X 4))
    (SETQ Z (CL:* X 5))
    (SETQ V (+ 3 (CL:* Z Y)))
    (CL:VALUES V)))
```

Figure 11: The function given by the algorithm in figure 8

8. Running the program interactively

Select the RUN PROGRAM command, then the algorithm is executed interactively. The ISTS asks for the values of all input variables. The number pad menu shown in figure 12 will appear for every input variable. You enter a value for the titled variable. When you are done, select the OK command. When all the input variables have got values, the ISTS answers with a list of the output variables and their values in the prompt print window (see figure 3).

![Number pad menu](image)

Figure 12: Number pad menu

9. Saving information about the system

Select the SAVE command. Some information about the system is saved in a text file. The ISTS will ask you for a filename to save the information in. The space-coordinate, in-ports, out-ports and the assignments are saved for every cell and in-ports and out-ports to the total system are also saved.
10. Changing the scale on the axes

Select the SCALE command if you want to change the scale on the axes, one at a time. Select one of the submenu commands X-AXIS, Y-AXIS or T-AXIS if you only want to change the scale for one of these axes (see figure 13). A number pad menu will appear. The current number of intervals are displayed. Enter a new number of intervals on the selected axis. When you are done, select the ok command. The scale on the selected axis will be changed.

![Figure 13: The SCALE command](image)

You can also place the cursor over the arrowhead at some of the axes and press the middle mouse button. Then a number pad menu will appear as above.

11. Changing the axes

You can change the length of the x-axis and t-axis by placing the cursor over the arrowhead for the x-axis or the t-axis and press and hold the left mouse button. You can see how the axis is moved when you move the cursor. When you are satisfied with the length you release the button. The length of the selected axis is now changed.

You can change the length and the direction of the y-axis by placing the cursor over the arrowhead for y-axis and press and hold the left mouse button. You can see how the y-axis is moved when you move the cursor. When you are satisfied with the length and the direction of the y-axis you release the button. The length and the direction of the y-axis are now changed.

The axes cannot be drawn outside the window. If you move the cursor outside the window the endpoint of the axis will stay inside the window.
12. Undoing the last event

The last event can always be undone. Select the UNDO command. You get information about which the last command was and may confirm if it shall be undone. After that it is possible to UNDO undone until a new command is performed.

13. Aborting a command

You can abort a command that is not completed. Press the right mouse button when you want to cancel a command.

14. Working with several ISTS

If you are working with several ISTS at the same time and you don't now which assignment menu belongs to which ISTS, then you can select the ASSIGNMENT MENU command and the menu will appear flushing.

15. Help

When you press and hold the left or middle mouse button at a command in a menu an explanation of this command is written in the black prompt window.

16. A summary of the menus and the commands

The ISTS different menus are:

- The attached menu that always is displayed.
- The node commands. A pop-up menu that appears when you press and hold the middle mouse button at a node.
- The right button default window menu. A pop-up menu that appears when you press and hold the right mouse button inside a window.
16.1  The attached menu

**UNDO**  The last event can be made undone. The user gets information about which the last command was and may confirm if it shall be undone. After that it is possible to UNDO undone until a new command is performed.

**ASSIGNMENTS MENU**  A menu is created with all the single assignments that define the algorithm (see figure 4) which shall be scheduled. Select one of the subcommands.

**LOAD FILE**  If you already have saved the program in a file this command will load the file and create an assignment menu. The ISTS will ask you for a file name.

**ASSIGNEDIT**  A single assignment editor ASSIGNEDIT will be opened and you can write a single assignment program of the algorithm. See figure 2.

**CONNECT NODES**  All nodes that are scheduled are connected with directed arcs. The data dependence graph is displayed. You can see how the nodes are dependent of each other.

**MOVE NODES**  Nodes can be moved as a group either in space or time. See item 4.3 above. Select one of the subcommands.

**SPACE**  The nodes are moved in the space.

**TIME**  The nodes are moved in the time direction.

**PROJECTION**  Tells the system to draw or remove the drawing of the projection of the space-time mapping in space. A toggling command.

**NODE LINES**  Tells the system to draw or remove the lines from the nodes to the axes. A toggling command.
MAKE FUNCTION A simulation function is generated of the algorithm.

The sequence of the execution of the assignments are given by the data dependence between the assignments. The input variables are input parameters to the function.

RUN PROGRAM The algorithm is executed interactively.

The system asks for the values of all input variables.

The system answers with all variables and their resulting values in the ISTS prompt print window.

SAVE The information about the system is saved in a text file. Space-coordinates, in-ports, out-ports and the code are saved for each cell. Also in-ports and out-ports to the total system are saved.

SCALE Sets the number of intervals on the axes, one axis at a time.

X-AXIS Sets the number of intervals on the x-axis.

Y-AXIS Sets the number of intervals on the y-axis.

T-AXIS Sets the number of intervals on the t-axis.

GRID? Tells the grid to be drawn or not in the ISTS. A toggling command.

INSPECT Inspects the window.

QUIT Exits ISTS
16.2 The node commands

MOVE  Moves the node first in the space and, then in the time direction.

SPACE  Moves the node in the space.

TIME  Moves the node in the time direction.

LINES  Draws guide-lines from the node to the axes.

SPACE  Draws guide-lines from the node to the axes in space.

TIME  Draws a guide-line from the node to the time axis.

BOX  Draws a box, i.e. guide-lines are drawn in the different planes to show the node with an illusion of the third dimension.

INSPECT  Inspects the node. Information about its position, assignment and the adjacent nodes are written in a window.
16.3 The right button default window menu

An explanation of the commands:

Close  Removes the window from the screen.
Snap   Copies a portion of the screen into a new window.
Paint  Allows drawing in a window.
Clear  Clears the window by erasing everything within the window.
Bury   Puts the window beneath all the other windows that overlap it.

Redisplay  Redisplays the window contents.
Hardcopy   Sends the contents of the window to a printer or to a file.
Move      Allows the window to be moved to a new position on the screen.
Shape     Reshapes the window.
Shrink     Reduces the window to an icon.
Expand     Changes an icon back to its original window.