Semantics of a Parallel Computation Model
and its Applications in Digital Hardware Design

by

Zebo Peng

This paper was published as a part of the Proceedings of the
1988 International Conference on Parallel Processing
Pennsylvania State University, August 15-19, 1988

RESEARCH REPORT
CADLAB, September 1988

Postaddress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden
The Department of Computer and Information Science
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

( Linköping Studies in Science and Technology. Dissertations. )

No 94  Hans Lunnell: Code Generator Writing Systems, 1983.
No 97  Andrzej Lingas: Advances in Minimum Weight Triangulation, 1983.
*No 109  Peter Fritzson: Towards a Distributed Programming Environment based on Incremental Compilation, 1984.

Licentiate of engineering Theses:

( Linköping Studies in Science and Technology. Theses. )

No 73  Ola Strömberg: A Structure Editor for Documents and Programs. 1986.
No 108  Rober Bilos: Incremental Scanning and Token-based Editing. 1987
No 113  Ralph Rönquist: Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118  Mariam Kamkar, Nahid Shahmehri: Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126  Dan Strömberg: Transfer and Distribution of Application Programs. 1987
Semantics of a Parallel Computation Model and its Applications in Digital Hardware Design

by

Zebo Peng
E-mail: spe@ida.liu.se

Abstract: This paper describes a parallel computation model based on a data/control flow notation which consists of separate but related sub-models of data path and control. The data path is formulated as a directed graph. The control structure, on the other hand, is modelled as a Petri net. This model is used for specification and synthesis of digital hardware with a high degree of concurrency and parallelism. The semantics of the proposed model is defined in terms of its interactions with the environment. That is, two pieces of hardware are considered to be semantically equivalent if they interact with an environment in the same way. This allows manipulation of the internal structure of the hardwares to improve performance as well as reduce cost. A set of transformations for the model which preserve its semantics is presented. A sequence of such transformations can be used to move a design from an abstract description to a final implementation.

This research has been supported by the Swedish National Board for Technical Development (STU).

This paper was published as a part of the Proceedings of the 1988 International Conference on Parallel Processing Pennsylvania State University, August 15-19, 1988

Postaddress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden
1. Introduction

One approach to the design of complex digital hardware for VLSI implementation is to use top-down synthesis technique. A synthesis approach starts the design with an abstract specification and refines it step by step towards a physical implementation by adding details [6]. Automated synthesis of parallel systems requires a parallel computation model to support the description of the system being designed. Such a computation model must be able to express the existence of multiple hardware resources for data storage, computation, and communication. At the same time, it must be able to represent the existence of multiple control flows and synchronization schemes.

This paper describes a parallel computation model in which a data path is used to represent the available hardware resources for data manipulation. The organization of this set of hardware resources to perform the prescribed computation is defined as a control structure which specifies the partial ordering of the given operations. Those operations which are not ordered, i.e., do not dependent on each other, can be carried out in parallel by physically distributed hardware resources. The control structure is formulated as a Petri net in the proposed model.

One important task of a hardware synthesis process is to perform design optimization. As such, there must be as much freedom as possible to alter parts of the control as well as data path in ways that do not change the behavior of the given system. For this possible, we must be able to characterize the behaviors of a system and define precisely the concept of equivalent systems. The semantics of the proposed model is defined in terms of its interactions with the environment. That is, only the external events are relevant to the semantics of the system. In this way, the internal structure of the digital system can be change without changing its semantics. The system's interaction with its environment is in turn defined based on two factors. First the functional relationship between each output variable and its relevant input variables must be the same; secondly the temporal relationship between input/output operations should not be different. This definition differs from other approaches which consider only the input/output functional relation in terms of the values being exchange between a system and its environment.

Most of other parallel system models have concentrated only on the synchronization aspect, or the partial ordering of communications, of parallel systems [5], [2]. For example, a Petri nets could be used to represent event/condition system where a partial ordering of the occurrence of events is specified but the contain of the events are ignored [5]. CCS (a Calculus for Communicating Systems) defined by Milner [2], on the other hand, models the occurrence of potentially concurrent events as a shuffle (interleaving) of those events; i.e., the events can occur in either order. As such, it has the composition explosion problem. That is when several agents are composed together, the possible number of behaviors are of the exponential order of the number of agents. Consequently the complexity of the behavioral expressions is also increase exponentially. Further, the
computational aspects are also abstracted away in CCS. Our model, on the other hand, model both the computations and their synchronizations, which are necessary for synthesis of hardware systems.

Another description model for hardware synthesis which also used external events to characterize semantics of a system has been proposed by McFarland [1]. However, it uses regular expression to formulate the event structures. Consequently it is difficult to deal with concurrent event structures. We are more interested in synthesis of algorithms (finally implemented as hardware) which are expressed as partially ordered events.

2. Definition of the Computation Model

The proposed computation model is based on the concepts of data flow and control flow. The data flow part is modelled as a data path, which represent the existence of multiple hardware resources to perform different operations. The control flow, on the other hand, dictates the partial ordering of these operations. In a parallel computation, there exist more than one control signal streams which move on with their own paces and synchronize with each other only when necessary. The partial ordering relationship between different set of operations is modelled by a Petri net notation.

Definition 2.1: A data path, D, over an algebraic structure is a five-tuple, $D = (V, I, O, A, B)$,

where $V = \{V_1, V_2, ..., V_n\}$ is a finite set of vertices each of which represents a data manipulation node;

$I = I(V_1) \cup I(V_2) \cup ... \cup I(V_n)$ with $I(V_j) =$ the set of input ports associated with vertex $V_j$;

$O = O(V_1) \cup O(V_2) \cup ... \cup O(V_n)$ with $O(V_j) =$ the set of output ports associated with vertex $V_j$;

$P = I \cup O$ is the set of ports; it is assumed that $I \cap O = \emptyset$.

$A \subseteq O \times I = \{ (O, I) \mid O \in O(V_i), I \in I(V_j), i,j = 1,2,\ldots,n \}$, is a finite set of arcs each of which represents a connection from an output port of a vertex to an input port of another vertex or the same vertex;

$B : O \rightarrow OP$, is a mapping from output ports to operations. $OP = \{OP_1, OP_2, \ldots, OP_m\}$ is a set of operations which define the functional relation between an output port of a vertex and its input ports. The set of operations are divided into the sequential set $SEQ$ and the combinatorial set $COM$.

Intuitively, a data path is a directed graph with each node having possibly multiple input ports and output ports. The nodes are used to model data manipulation units, for example data storages, arithmetic operators, or communication channels. The arcs are used to model the connections of these data manipulation units.
Therefore, the above definition is concerned mainly with the structure rather than the function of the data path. How the data path is used to perform computation is not explicitly defined. We assume that there exists an implicit interpretation of the underlying algebraic structure which supports the computation rules. Such an algebraic structure should consist of a domain of values for constants and variables, an assignment of values to the constants and a function definition for each operator. This algebraic structure is not considered here as it does not directly affect the basic formulation of the model. Further, to define the semantics of the system independent of any particular interpretation makes it possible to cope with different implementation environments. However, we assume that some modules exist in a module library which can perform the defined operations of the data path.

The notion of ports here is used as a basic abstraction of the input/output behavior of a data manipulation unit and thus separates the implementation of the operation associated with the vertices from the specification. The operation of the vertices are defined only by the relation between the output ports and the input ports. It is assumed that the output port will present a value which has the given relationship with the values present in the input ports.

**Definition 2.2:** A data/control flow system, \( \Gamma \), is a seven-tuple, \( \Gamma = (D, S, T, F, C, G, M_0) \)

where \( D = (V, I, O, A, R) \) is a data path;

\[ S = \{ S_1, S_2, \ldots, S_n \} \] is a finite set of \( S \)-elements, or control states (places);

\[ T = \{ T_1, T_2, \ldots, T_m \} \] is a finite set of \( T \)-elements, or transitions;

\( F \subseteq (S \times T) \cup (T \times S) \) is a binary relation, the flow relation.

\( C : S \rightarrow 2^A \) is a mapping from control states to sets of arcs of the given data path; an arc \( A_i \) is controlled by a control state \( S_j \) if \( A_i \in C(S_j) \).

\( G : O \rightarrow 2^T \) is a mapping from output ports of data path vertices to sets of transitions; a transition \( T_i \) is guarded by output port \( O_j \) if \( T_i \in G(O_j) \).

\( M_0 : S \rightarrow \{0, 1\} \) is an initial marking function.

The definition of the data/control flow model is based on the marked Petri net notation. The Petri net \( S \)-elements are used to capture the control state concept. When a control state holds a token, a control signal will be generated to control the corresponding arcs in the data path specified by the control mapping function \( C \). As there could be more than one control state which holds tokens, there exist multiple control signals in the systems. Further, the flow of these control signals (the temporal relation between signals) is defined by a partial ordering structure, which is captured by the flow relation \( F \). To express the control flow being affected by the results of some internal computation, we must be able to use conditional signals (as results of some computation) to affect the
control flow. For this purpose, the guarding condition concept is introduced into the Petri net notation; a transition may be guarded by a condition produced from the data path represented as the output port of some vertices.

Definition 2.3: For a data/control flow system $\Gamma = (D, S, T, F, C, G, M_0)$:

1. $X = S \cup T$ is the set of control structure elements.
2. $F^+ = \{ F^n \mid n \in \mathbb{N}^+ \}$, where $F^0$ = identity and $F^n = F \circ F^{n-1}$ for $n \in \mathbb{N}^+$, is the transitive closure of $F$.
3. $S_i \Rightarrow S_j$ iff $(S_i, S_j) \in F^+$; $\Leftarrow = (\Rightarrow)^{-1}$.
4. $\alpha = \Rightarrow \cup \Leftarrow$. $S_i$ and $S_j$ are said to be in sequential order if $S_i \alpha S_j$.
5. $\parallel = (S \times S \setminus \alpha)$. $S_i$ and $S_j$ are said to be in parallel order if $S_i \parallel S_j$.

The data path consists of two kinds of elements, the nodes together with their ports representing the data manipulation units and the arcs representing the connection between those units. Each arc is controlled by, or said to be associated with, some control signals coming from the control Petri net. We can also associate the data manipulation units with the control signals by the following definition.

Definition 2.4: $V_k$ is said to be associated with $S_j$ if

$$\exists (i, \delta) \in A \mid (i \in I(V_k)) \cap ((i, \delta) \in C(S_j)).$$

By this definition, only the input ports of a vertex are significant for the associative relation. The output ports are irrelevant here because an output port can send data to more than one place at a time without resulting in conflicts. A single input port, on the other hand, cannot receive signals simultaneously from more than one resource.

The set of vertices and arcs associated with a control state $S$ forms a subgraph of the data path graph. This graph is called the associated graph of $S$.

Definition 2.5: The arcs and vertices associated with control state $S_i$, denoted by $ASS(S_i)$, are said to be active under $S_i$.

Intuitively, the arcs representing the data paths (e.g., a bus) are open, i.e., allow signal to pass, when their associated control signals are on; the associated data manipulation units, on the other hand, will perform predefined operations.

Before we go to the formal definition of the concepts of semantics and semantic equivalence, let us look at some simple examples. Under the above formulation, a simple adder with two input ports and one output port can be modelled as a vertex $V_1$ with $I(V_1) = \{ Pu, Ps \}$, $O(V_1) = \{ Po \}$. A register can be modelled as a vertex $V_2$ with $I(V_2) = \{ Ps \}$ and $O(V_2) = \{ Po \}$. A data path which connects the output of the adder to the
register can be modelled as an arc $A_I = (P_{oi}, \text{Pis})$, which states that the output port of the $V_I$ component is connected to the input port of $V_z$.

If the output of the adder is only fed into the register when control state $S_I$ is on, then $A_I \in C(S_I)$ and $\{V_z, A_I\} \subseteq \text{ASS}(S_I)$. Note that $V_I$ need not necessarily be associated with $S_I$; if, for example, the adder has a local accumulator, a series of additions can be performed and finally the sum be fed into register $V_z$ when $S_I$ is on. When the sum is being sent to $V_z$, $V_I$ can continue with another addition associated with, e.g., $S_2$ without conflict.

3. Semantics of the Model

We now turn our attention to the definition of the semantics of the proposed computation model. The basic idea is that we can characterize the semantics of a system by the external events, i.e., its interactions with the outside world. An external event is either a read or write operation of the externally accessible ports. The semantics of a hardware system is defined as a set of events observed in its external ports.

Before formally giving the definition of semantics of the computation model, we have to define the behaviors of the system which is in turn based on the execution rules of the control Petri net and its interaction with the data path.

**Definition 3.1**: Given a data/control system $T = (D, S, \mathbf{T}, \mathbf{F}, C, G, P_0)$, its behavior is defined as below:

1. A function $M : S \rightarrow \mathbb{N}$ is called a marking of $T$ ($\mathbb{N} = \{0, 1, 2, \ldots \}$). A marking is an assignment of tokens to the $S$-elements.
2. Initially there is a token in each of the initial control states, or the set of $S$-elements $S_i$ such that $M_0(S_i) = 1$ as defined by the initial marking $M_0$.
3. A transition $T$ is enabled at a marking $M$ iff for every $S$ such that $(S, T) \in \mathbf{F}$, $M(S) \geq 1$; that is, all the $T$-elements' input control states have at least one token.
4. A transition $T$ may be fired when it is enabled and the guard condition is true (i.e., the output port which guards $T$ has a TRUE value). If a transition has more than one guard condition, an OR operation is applied to them; therefore, if any guard condition is true, the transition's guard condition as a whole is true.

5. Firing an enabled transition $T$ removes a token from each of its input control states and deposits a token in each of its output control states.
6. If no token exists in any of the control states, the execution is terminated.
7. $\mathcal{V}(P)$ is the data value present at port $P$.
8. When a control state, $S$, holds a token, its associated arcs in the data path will open for data to flow; i.e., the data value presents at the input port, $I$, is equal to
the corresponding output port, \( O \), which is denoted as \( \mathcal{V}(I) \rightarrow S \mathcal{V}(O) \).

9. For every vertex \( V \), \( \mathcal{V}(O) := OP(\mathcal{V}(I(V))) \), where \( OP \in B(O) \). The assignment operator, \( := \), means that if \( OP \) is sequential it takes the last defined value of the expression; otherwise it takes the present value of the expression.

10. If all the pending arcs of an input port are not active, its value is undefined. If the operation of an output port is not a sequential one and the output port depends on an undefined input value, its value is also undefined.

The possible existence of an undefined value and the intrinsic non-deterministic properties of the Petri net firing sequence together result in difficulties in determining the behavior of a system. We would like to exclude the nondeterministic properties by the following definition.

**Definition 3.2:** A data/control flow system \( \Gamma = (D, S, T, F, C, G, M_o) \) is properly designed if:

1. \( \text{ASS}(S_i) \cap \text{ASS}(S_j) = \emptyset \), if \( S_i \parallel S_j \).

2. There should not be more than one token appearing at the same control state; that is, the Petri net must be safe.

3. If \( (S, T_i) \in F, (S, T_e) \in F, T_i \in G(P_{oi}), \) and \( T_e \in G(P_{oe}) \), then \( \mathcal{V}(P_{oi}) \text{ AND } \mathcal{V}(P_{oe}) = \text{FALSE} \). That is, the Petri net must be conflict-free.

4. The subgraph that belongs to a control state should not include a combinatorial loop.

5. \( \forall S_i \in S \text{ ASS}(S_i) \) must include at least one sequential vertex.

This definition singles out those data/control flow systems which are safe, conflict-free and well-behaved. From now on we only consider properly designed systems.

**Definition 3.3:** For a data path \( D = (V, I, O, A, R) \), there is a set of external vertices, \( V_e \), which only have either one single input port (the set of output vertices, \( V_o \)) or one single output port (the set of input vertices, \( V_i \)). The set of ports of the external vertices \( V_e \) are called **external ports**. The set of arcs, \( A_e \), which connect to the external ports, are called **external arcs**.

**Definition 3.4:** A **external event** is a pair \( (A_i, w) \), with \( A_i \) being an external arc and \( w \) a value passed over the arc. A external event is controlled by, or labelled with, the Petri net control state that is associated with the arc. That is, the external event happens at the time when the associated control state has a token.

**Definition 3.5:** Given a data/control flow system \( \Gamma = (D, S, T, F, C, G, M_o) \), its **external event structure** is defined as \( S(\Gamma) = (E, \prec, \times) \) where
E = \{E_i, E_2, ..., E_n\} is a set of external events;

\prec \subseteq (E \times E) is a binary relation, the precedent relation. E_i \prec E_j with E_i = (A_i, v_i) and E_j = (A_j, v_j), iff E_i occurs before E_j and S_i \Rightarrow S_j, where A_i \in C(S_i) and A_j \in C(S_j);

\simeq \subseteq (E \times E) is a binary relation, the concurrent relation. E_i \simeq E_j with E_i = (A_i, v_i) and E_j = (A_j, v_j), iff E_i and E_j occurs at the same time and A_i \in C(S), A_j \in C(S).

An external event structure specifies all the possible external events of a system as well as the temporal relationship between them. If two external events are in the precedent (concurrent) relation, they must always occur in the specified order (simultaneously). On the other hand, if two events are not in either of the two relations, they can occur in any order and are said to be in a casual relation. In a distributed system with a set of modules, for example, the temporal relations between some of the external events of two different modules can best be expressed as having a casual relation. Trying to force a total ordering on events of different modules will simply introduce unnecessary constraints and make it difficult to implement the system.

In the above discussion, we assume that when an external event occurs whose operation is to obtain a value from the outside world, the environment will supply a value of the appropriate type to the system. We also assume that a sequence of such values is implicitly predefined for each input vertex, when an external event structure is specified.

**Definition 3.6:** The semantics of a data/control flow system \(\Gamma\), denoted also by \(S(\Gamma)\), is defined by its external event structure.

### 4. Semantics Equivalence

Two systems are considered to be semantically equivalent if they behave identically with respect to the corresponding external ports; their internal behavior does not matter.

**Definition 4.1:** Two data/control flow systems \(\Gamma\) and \(\Gamma'\) are *semantically equivalent*, denoted by \(\Gamma \equiv \Gamma'\), iff \(S(\Gamma) = S(\Gamma')\).

For the purpose of synthesis, however, the above semantic equivalence relation is still too weak. In general, it is undecidable whether two systems are equivalent to each other by this definition. It is very difficult, or simply impossible in some cases, to analyze a data/control flow system and obtain the complete external event structure as specified by definition 3.5. We have thus to introduce a stronger equivalence relation which requires every data dependence operation to be carried out in exactly the same order. This latter requirement is stronger than necessary. For example, two addition operations can be carried out in reverse order without changing the outcome of the computation. This strong definition, however, greatly reduces the complexity of the synthesis process and still provides enough room for the optimization algorithm to make large changes in the
described system.

**Definition 4.2:** The domain of a control state $S$, denoted as $\text{dom}(S)$, is defined as the set of vertices that have some output port connected to an arc controlled by $S$. The codomain of $S$, denoted as $\text{cod}(S)$, is defined as the set of vertices which have some input port connected to an arc controlled by $S$. The operations performed on a control state $S$ are the set of operations defined on the output ports of its codomain. The subset of vertices of the codomain of $S$ that consists of some sequential output ports is called the result set of $S$ and denoted as $\mathbf{R}(S)$.

**Definition 4.3:** $S_i$ and $S_j$ are directly data dependent, denoted as $S_i \leftrightarrow S_j$, if one of the following is true:

(a) $\mathbf{R}(S_i) \cap \text{dom}(S_j) \neq \emptyset$.
(b) $\mathbf{R}(S_j) \cap \text{dom}(S_i) \neq \emptyset$.
(c) $\mathbf{R}(S_i) \cap \mathbf{R}(S_j) \neq \emptyset$.
(d) $S_i$ and $S_j$ are in a control dependence relation; i.e., $M(S_i)$ depends on a subset of $\mathbf{R}(S_j)$ or vice versa.
(e) $C(S_i)$ and $C(S_j)$ both contain some external arcs.

**Definition 4.4:** The transitive closure of $\leftrightarrow$, denoted by $\diamond$, i.e., $\diamond = \leftrightarrow^+$, is called a data dependence relation.

The data dependence relation is defined as the relationship between the operations which will contribute "data" to each other; in other words, two operations are data dependent if they must be executed in the predefined order in order to retain the semantic integrity of the prescribed computation. Those sets of control signals which are not in a data dependence relation, however, can be arranged in any order without changing the semantics of the system.

**Definition 4.5:** Given $\Gamma = (D, S, T, F, C, G, M_o)$ and $\Gamma' = (D, S, T', F', C, G, M_o)$, $\Gamma$ and $\Gamma'$ are data-invariantly equivalent to each other, iff

for every $S_i \Rightarrow S_j$ and $S_i \diamond S_j$ in $\Gamma$ ($S_i \in S, S_j \in S$),

we have $S_i \Rightarrow' S_j$ and $S_i \diamond' S_j$ in $\Gamma'$;

and vice versa.

The above definition ensures that two operations are performed in parallel only if they are data independent and all of the data dependent operations in the two systems are performed exactly in the same order. Therefore the data-invariant equivalence relation satisfies the semantic equivalence relation. This means that we can reconstruct the control structure (without changing the data path) of a hardware system to improve system performance, for example, by carrying out as much operations in parallel as possible.
Theorem 4.1: The data-invariant equivalence relation satisfies the semantic equivalence relation.

Proof 4.1: see the appendix.

Definition 4.6: Given $\Gamma = (D, S, T, F, C, G, M_o)$ with $D = (V, I, O, A, B)$ and $\Gamma' = (D', S, T, F, C, G', M_o)$ with $D' = (V', I', O', A', B')$, $\Gamma$ and $\Gamma'$ are control-invariantly equivalent to each other, iff $\Gamma'$ is the result of a vertex merger of $V_i$ into $V_j$ of $\Gamma$, both $V_i$ and $V_j$ have the same operational definition and port structure, and their associated control states are in sequential order. The result of a vertex merger is defined as:

$$V' = V - \{V_i\}.$$  
$$I' = I - \{I(V_i)\}.$$  
$$O' = I - \{O(V_i)\}.$$  

$A'$ is the same as $A$ except that each $(O_i, I)$ with $O_i \in O(V_i)$ is replaced by $(O_i, I)$ with $O_i \in O(V_j)$ and each $(O, I_i)$ with $I_i \in I(V_i)$ replaced by $(O, I)$ with $I_i \in I(V_j)$.

$G'$ is the same as $G$ except that each $T \in G(O_i)$ is substituted by $T \in G(O_i)$.

The intrinsic property of a merger operation is to share hardware resources by operations so as to improve the implementation in terms of cost. For example two addition operations can be implemented with the same adder by merging the two addition vertices together. By merging communication channels together we can also create structure components like buses in the implementation.

As a merger is only performed when the two vertices have their associated control states in sequential order, they will not attempt to use the vertex at the same time. As such the two sets of operations can share the same operator safely. Because the two vertices to be merged also have the same operational definition and port structure, the merger will not change the computational aspect of the given system.

Theorem 4.2: The control-invariant equivalence relation satisfies the semantic equivalence relation.

Proof 4.2: see the appendix.

5. Hardware Synthesis

This section discusses briefly the application of the proposed parallel computation model in a hardware synthesis environment. For a detailed description of the synthesis algorithms and comparisons to other related works, please see [3] and [4].

To synthesize hardware from some algorithmic description of its behavior, we first transform the description into the data/control flow notation. Based on such a formal
description, some formal analysis techniques can first be used to check whether the systems are properly designed before the synthesis process starts [4].

The major part of the synthesis process is carried out by a sequence of control-invariant and data-invariant transformations as defined in the previous section. Since both transformations do not change the semantics of the system, they can freely be applied to transform a design to satisfy certain given criteria. For example, adding one more control flow path in the Petri net and possibly additional data manipulation units in the data path will allow more operation units to operate at the same time, thus increasing the parallelism of the computation.

The synthesis algorithm starts with a preliminary design and transforms it step by step towards an optimal one. As from each step there are usually several ways to go, it is necessary to have some strategy to guide the transformation process. A critical path analysis technique is used for this purpose. The set of transformation, analysis, and optimization algorithms has been designed and implemented in the CAMAD design aid system [3], [4].

6. Conclusions

We have given the formal definition of a data/control flow model for parallel computation and its semantic equivalence notation. The concept of semantic equivalence is defined based on two criteria. First the functional relationship between each output variable and its relevant input variables must be the same; secondly the temporal relationship between input/output operations should also be the same.

Unlike other computation models used mainly for descriptive and analysis purposes, the proposed model addresses issues of design directly and allows graphical representations of the structures as well as behaviors of hardware system. To apply this model for hardware synthesis, we have introduced two basic transformations which change the internal structure of the hardware but keep the data dependency operations in the predefined order. The requirement that all data dependency operations be carried out in the predefined order is actually stronger than necessary. For example, two addition operations can be carried out in a reversed order without changing the outcome of the computation. It, however, greatly reduces the complexity of the synthesis process. The use of such a formal computation model to represent the design of parallel hardware has led to the efficient use of CAD and automatic tools in the synthesis process.

Appendix

Proof 4.1: Let \( \Gamma = (D, S, T, F, C, G, M_0) \), \( \Gamma' = (D, S, T', F', C, G, M_0) \), and \( \Gamma \) and \( \Gamma' \) are data-invariant equivalent to each other, i.e., for every \( S_i \Rightarrow S_j \) and \( S_i \bowtie S_j \) in \( \Gamma \) (\( S_i \in S, S_j \in S \)), we have \( S_i \Rightarrow' S_j \) and \( S_i \bowtie' S_j \) in \( \Gamma' \); and vice versa. We will show that the external event structure of \( \Gamma \) and that of \( \Gamma' \) are the same.
Suppose that a sequence of external events, \( \{A_i, w_i\}, \{A_i, w_2\}, \{A_i, w_3\}, \ldots \), are observed in arc \( A_i \) which is associated with control state \( S \) in system \( \Gamma \). As the data path of system \( \Gamma' \) is the same as that of \( \Gamma \), \( A_i \) should also be present in \( \Gamma' \) as an external arc and controlled by \( S \) in \( \Gamma' \).

For the values exchanged over \( A_i \), we have two situations:

1. If \( A_i \) is connected to an input vertex, the function of the external events is to input data from the environment. The values passed over the arc are then provided by the environment. As we assume that the sequence of such values provided for each input vertex is fixed when we check the semantic equivalence relation between different systems, the same sequence of external events will be observed in system \( \Gamma' \).

2. If \( A_i \) is connected to an output vertex, the function of the external events is to output data to the environment. The values passed over \( A_i \) are, therefore, determined by the computation performed by the systems.

Let \( A_i = (O, I) \), and when \( M(S) = 1 \), an external event \( \{A_i, w\} \) occurs with \( w = \mathcal{V}(O) \) (definition 3.4). If \( O \in O(V) \) and \( V \) is an input vertex (i.e., \( A_i \) connects an input vertex directly to an output vertex), \( \mathcal{V}(O) \) depends again on the environment. Therefore, both systems exchange the same values at \( A_i \).

If \( O \in O(V) \) and \( V \) is not an input vertex, we have \( \mathcal{V}(O) := OP(\mathcal{V}(I(V))) \), where \( OP \in \mathcal{B}(O) \); and \( \mathcal{V}(I(V)) \) for each \( I(V) \) (definition 3.1). As \( V \in \text{dom}(S) \) and \( V \in \mathcal{R}(S_i) \) (we have assumed that \( V \) is a sequential vertex, without loss of generality), we have \( S \sim S_i \) and, therefore, \( S_i \Rightarrow S \). Since both systems have the same data path and \( S_i \Rightarrow S \) in both situations, the values exchanged at \( A_i \) should be the same provided that each \( \mathcal{V}(O) \) for the corresponding systems is the same.

To show that \( \mathcal{V}(O_i) \) is the same for \( \Gamma \) and \( \Gamma' \), we can use the same proof process as above. This recursive procedure will also converge to the situation where \( V \) is an input vertex. At that time the same argument as from (1) can be applied again. Therefore, the same sequence of external events will be observed in \( A_i \) of both system \( \Gamma \) and \( \Gamma' \).

From (1) and (2), it is clear that the sequence of external events observed at \( A_i \) of \( \Gamma' \) is exactly the same as that of \( \Gamma \) in any situation. As \( A_i \) can be any arbitrary external arc, this means that the sequence of external events which occur at every external arc is the same for both systems.

As \( \Gamma \) and \( \Gamma' \) have the same number of corresponding external arcs, it follows from the above result that the complete sets of external events for both systems are the same.

Next let us look at the partial relation between the external events of the two systems. Suppose that \( E_i < E_j \) with \( E_i = \{A_i, w_i\} \) and \( E_j = \{A_j, w_j\} \) in \( \Gamma \). That is, \( E_i \) occurs before \( E_j \) and \( S_i \Rightarrow S_i \) where \( A_i \in C(S_i) \) and \( A_j \in C(S_j) \). By definition, we have \( S_i \Rightarrow S_j \) in \( \Gamma' \),
where $A_i \in C(S_i)$ and $A_j \in C(S_j)$, because $S_i \odot S_j$ (thus $S_i \odot' S_j$).

Assuming $E_i$ occurs before $E_i$ in $\Gamma'$, then we must have $S_j \Rightarrow S_i$ in both $\Gamma'$ and $\Gamma$. That is, $S_i$ and $S_j$ are in a loop situation. Consequently, there exists a total ordering between the external events associated with these two control states, and it should be the same in both $\Gamma$ and $\Gamma'$. Thus the assumption that $E_i$ occurs before $E_i$ in $\Gamma'$ is a contradiction. Therefore we have also $E_i$ occurs before $E_j$ in $\Gamma'$. That is, $E_i < E_j$ is also in $\Gamma'$.

Finally, we show that the concurrent relations of both systems are also the same as follows.

If $E_i = (A_i, w_i)$ and $E_j = (A_j, v_j)$ occur at the same time and $A_i \in C(S_i)$, $A_j \in C(S_j)$ in $\Gamma$, then we should have $A_i \in C(S_i)$, $A_j \in C(S_j)$ in $\Gamma'$, because the control mapping $C$ is the same for both systems. Consequently, $E_i$ and $E_j$ should also occur at the same time in $\Gamma'$ as they are associated with the same control state. Therefore, both system have the same concurrent relation.

Since both system $\Gamma$ and $\Gamma'$ have the same external event set, the same precedent relation, and the same concurrent relation, $\$($\Gamma$) = $\$($\Gamma'$). That is, they are semantically equivalent to each other.

**Proof 4.2:** Let $\Gamma$ and $\Gamma'$ be control-invariant equivalent to each other. That is, (a) $\Gamma'$ is resulted from a vertex merger of $V_i$ into $V_j$ of $\Gamma$, (b) both $V_i$ and $V_j$ have the same operational definition and port structure, and (c) their associated control states are in sequential order.

Assume that the merger of $V_i$ and $V_j$ changes the semantics of the system. That is, $\$($\Gamma$) $\neq$ $\$($\Gamma'$), or ($E$, $\prec$, $\succ$) $\neq$ ($E'$, $\prec'$, $\succ'$). Because the control structures of both systems are the same, the temporal relationship between any two control states remain the same for both systems.

As the number of arcs also remains the same after the merger operation and they are controlled by the same control states, the number of external events and their temporal relation remain the same for both systems. That is, the precedent relation and concurrent relation of both systems are the same. Therefore, the only possible difference between the two external event structures is that some of the external events have different values.

For the external events that occur at an arc connected to an input vertex, the same argument of Proof 4.1(1) can be used to prove that both $\Gamma$ and $\Gamma'$ have the same values passed in these external events.

For the external arcs that are connected to an output port, let $A = (O, I)$ with $I \in I(V_o)$ and $V_o \subseteq V_o$; $\{(A, w_i), (A, w_0), (A, w_0), \ldots\} \subseteq E$ and occur in the listing order in $\Gamma$; and $\{(A, w_j), (A, w_0), (A, w_0), \ldots\} \subseteq E'$ and occur in the listing order in $\Gamma'$. 
Let also \( (A, v_A) \) and \( (A, v_B) \) occur when \( M(S) = 1 \) in \( \Gamma \) and \( \Gamma' \) respectively. We have \( v_A = \gamma(O) \) in \( \Gamma \) and \( v_B = \gamma(O) \) in \( \Gamma' \).

If \( O \in O(V) \) and \( V \) is an input vertex in \( \Gamma \), we have also \( O \in O(V) \) and \( V \) as an input vertex in \( \Gamma' \). Since in both cases \( \gamma(O) \) depends on the environment, \( v_A = v_B \).

If \( O \in O(V) \), \( V \) is not an input vertex, and \( V \neq V_i \), we have \( \gamma(O) := OP(\gamma(I(V))) \), where \( OP \in B(O) \) both in \( \Gamma \) and \( \Gamma' \). By definition 3.1, \( \gamma(I_i) \rightarrow S_i V(O_i) \) for each \( I_i \in I(V) \). As both system have \( S_i \Rightarrow S \) (see Proof 4.1), \( v_A = v_B \), provided that each \( \gamma(O_i) \) for both systems is the same.

If \( O \in O(V) \), \( V \) is not an input vertex, and \( V = V_i \) in \( \Gamma \), we have \( \gamma(O) := OP(\gamma(I(V_i))) \), where \( OP \in B(O) \) in \( \Gamma \) and \( \gamma(O) := OP(\gamma(I(V_j))) \), where \( OP \in B(O) \) in \( \Gamma' \). Since (a) both \( V_i \) and \( V_j \) have the same operational definition and port structure; (b) \( \gamma(I_i) \rightarrow S_i \gamma(O_i) \) for each \( I_i \in I(V_i) \) in \( \Gamma \) with \( \gamma(I_i) \rightarrow S_i \gamma(O_i) \) for each \( I_i \in I(V_j) \) in \( \Gamma' \); and (c) both systems have \( S_i \Rightarrow S \); we have \( v_A = v_B \), provided that each \( \gamma(O_i) \) for both systems is the same.

To show that \( \gamma(O) \) is the same for \( \Gamma \) and \( \Gamma' \) in the above two cases, we can use the same proof process again. The recursive procedure will also converge to the situation where \( V \) is an input vertex; then the same argument as from Proof 4.1(1) can be applied. Therefore, the same sequence of external events will be observed in \( A \) of both \( \Gamma \) and \( \Gamma' \).

This result contradicts the assumption that some corresponding events of \( \Gamma \) and \( \Gamma' \) are different. That is, the assumption must be false. Therefore, \( \Gamma \equiv \Gamma' \).

References


Semantics of a Parallel Computation Model and its Applications in Digital Hardware Design

Zebo Peng

Abstract: This paper describes a parallel computation model based on a data/control flow notation which consists of separate but related sub-models of data path and control. The data path is formulated as a directed graph. The control structure, on the other hand, is modelled as a Petri net. This model is used for specification and synthesis of digital hardware with a high degree of concurrency and parallelism. The semantics of the proposed model is defined in terms of its interactions with the environment. That is, two pieces of hardware are considered to be semantically equivalent if they interact with an environment in the same way. This allows manipulation of the internal structure of the hardwares to improve performance as well as reduce cost. A set of transformations for the model which preserve its semantics is presented. A sequence of such transformations can be used to move a design from an abstract description to a final implementation.
A Selection of Previous Research Reports.


LiTH-IDA-R-88-18  Nils Dahlbäck: Mental Models and Text Understanding - a Commented Review.


LiTH-IDA-R-88-16  Lin Padgham: A Model and Representation for Type Information and Its Use in Reasoning with Defaults. Also in Proc. of AAAI'88, American Association for Artificial Intelligence, 1988.


LiTH-IDA-R-88-13  Erik Tengvall: Ett kartorienterat planeringsystem för autonom farkoster, en design diskussion.


LiTH-IDA-R-88-11  Mats Wirén: An Incremental Chart Parser for PATR.


LiTH-IDA-R-88-09  Peter Fritzson: Incremental Symbol Processing.


LiTH-IDA-R-88-06  Christer Bäckström: A Representation of Coordinated Actions Characterized by Interval Valued Conditions.

LiTH-IDA-R-88-05  Christer Bäckström: Keeping and Forcing: How to Represent Cooperating Actions.

organizes undergraduate and graduate studies in Computer Science, Telecommunication and Computer Systems, and Administrative Data Processing. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- **ACTLAB - Laboratory for Complexity of Algorithms**, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.

- **ASLAB - Application Systems Laboratory**, which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.

- **CADLAB - Laboratory for Computer-Aided Design of Electronics**, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.

- **LIBLAB - Laboratory for Library and Information Science**, which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.

- **LOGPRO - Laboratory for Logic Programming**, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.

- **NLPLAB - Natural Language Processing Laboratory**, which conducts research related to the development and use of natural language interfaces to computer software.

- **PELAB - Programming Environments Laboratory**, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.

- **RKLLAB - Laboratory for Representation of Knowledge in Logic**, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

**Research Reports 1988**

**LiTH-IDa-R-88-36**  

**LiTH-IDa-R-88-35**  

**LiTH-IDa-R-88-34**  
Arne Jönsson, Nils Dahlbäck: Talking to a computer is not like talking to your best friend. Also in *Proc. of the First Scandinavian Conference on Artificial Intelligence*, March 9-10, 1988, Tromsø, Norway.

**LiTH-IDa-R-88-32**  