Acta Cybernetica, Vol. 12, No. 1, Szeged, 1995

Parallel asynchronous computation of the values of an associative function *

Zoltán Horváth †

Abstract

This paper shows an application of a formal approach to parallel program design. The basic model is related to temporal logics. We summarize the concepts of a relational model of parallelism in the introduction. The main part is devoted to the problem of synthesizing a solution for the problem of parallel asynchronous computation of the values of an associative function. The result is a programming theorem, which is wide applicable for different problems. The abstract program is easy to implement effectively on several architectures.

The applicability of results is investigated for parallel architectures such as for hypercubes and transputer networks.

1 Introduction

We summarize the basic concepts of a relational model of parallelism [11,13,12]. Our model is an extension of a powerful and well-developed relational model of programming, which formalizes the notion of state space, problem, sequential program, solution, weakest precondition, specification, programming theorem, etc. [8,9,16].

1.1 A relational model of parallel programs

We take the specification as the starting point for program design. We use a model of programming which supports the top-down refinement of specifications [19,8,10,9,2,11]. The proof of the correctness of the solution is developed parallel to the refinement of the specification of the problem. We formalize the main concepts of UNITY [2] in an alternative way. We use a relatively simple mathematical machinery [8,11]. The result is an expressive model, which is related to branching time temporal logics.

We give a brief survey of the main concepts and apply the methodology to solve the problem of parallel asynchronous computation of the values of an associative function in the main part.

*Supported by the Hungarian National Science Research Grant (OTKA), Grant Nr. 2045

†Dept. of General Computer Science, Eötvös Loránd University, Budapest, Hungary, 1088 Budapest, Múzeum krt. 6-8., E-mail: hz@ludens.elte.hu

1.1.1 Preliminary notions

In the following we use the terminology used also in [17,8,10,9,11]. Notations are defined often by the help of the special equality sign ::=.

The binary relation $R \subseteq A \times B$ is a *function*, if $\forall a \in A : |R(a)| = 1$. We define the domain of a relation R as $\mathcal{D}_R$::=$\{a \in A | R(a) \neq \emptyset\}$. We use the notation $f : A \longmapsto B$ for functions.

The set of the logical values is denoted by $\mathcal{L}$, i.e., $\mathcal{L}$::=$\{\uparrow, \downarrow\}$. A relation $f \subseteq A \times \mathcal{L}$ is called logical function, if it is a function. We use the words predicate and condition as synonyms for logical function. $\lceil f \rceil$::=$\{a \in A | f(a) = \{\uparrow\}\}$ is called the truth-set of the logical function f. $[f]$ abbreviates the theorem $(\lceil f \rceil = A)$ [4]. The operations $\cup, \cap, A\backslash$ correspond to the function compositions $\wedge, \vee, \neg$. $\Rightarrow$ corresponds to $\subseteq$, $P \rightarrow Q$ is an abbreviation of $\neg P \vee Q$.

The set of the subsets of a set A is called the powerset of A and denoted by $\mathcal{P}(A)$.

Let $I \subset \mathcal{N}$. $\forall i \in I : A_i$ is a finite or numerable set. The set A::=$\times_{i \in I} A_i$ is called state space, the sets A_{i_j} are called *type value sets* . The projections $v_i : A \longmapsto A_i$ are called variables. A^* is the set of the finite sequences of the points of the state space and A^∞ the set of the infinite sequences. Let $A^{**} = A^* \cup A^\infty$.

We can imagine a statement (a sequential program) as a relation, which associates a sequence of points of the state space to some points of the state space, i.e., a statement is a subset of the direct product $A \times A^{**}$. The full formal definition of statement is given in [8].

The **effect relation** of a statement s is denoted by $p(s)$. The effect relation expresses the functionality of the statement. $p(s) \subseteq A \times A$, $\mathcal{D}_{p(s)}$::=$\{a \in A \mid s(a) \subseteq A^*\}$, and
$\forall a \in \mathcal{D}_{p(s)} : p(s)(a)$::=$\{b \in A \mid \exists \alpha \in s(a) : \tau(\alpha) = b\}$, where $\tau : A^* \rightarrow A$ is a function, which associates its last element to the sequence $\alpha = (\alpha_1, ..., \alpha_n)$, i.e., $\tau(\alpha) = \alpha_n$.

The logical function $wp(s, R)$ is called the **weakest precondition** of the postcondition R in respect of the statement s. We define $\lceil wp(s, R) \rceil$::=$\{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$. The logical function $sp(s, Q)$ is called the strongest postcondition of Q in respect of s. $\lceil sp(s, Q) \rceil$::=$p(s)(\lceil Q \rceil)$.

$A = A_1 \times \ldots \times A_n$, $F = (F_1, \ldots, F_n)$, where $F_i \subseteq A \times A_i$. Let $\lceil \pi_i \rceil$::=$\mathcal{D}_{F_i}$. The relation $F_i|_{\lceil \uparrow \rceil}$ is the extension of F_i for the truth set of condition $\uparrow$ [6], i.e., $F_i|_{\lceil \uparrow \rceil}(a)$::=$F_i(a)$, if $a \in \lceil \pi_i \rceil$ and $F_i|_{\lceil \uparrow \rceil}(a)$::=$a_i$, otherwise. $F||_{\lceil \uparrow \rceil}$::=$(F_1|_{\lceil \uparrow \rceil}, \ldots, F_n|_{\lceil \uparrow \rceil})$.

Let us use the notation $(\|_{i \in [1,n]} (v_i :\in F_{j_i}(v_1, .., v_n), \text{ if } \pi_{j_i}))$ for the statement s_j, for which $((\mathcal{D}_{s_j} = A) \wedge (\forall a \in A : p(s_j)(a) = F||_{\lceil \uparrow \rceil}(a)))$. This kind of (simultaneous, nondeterministic) assignment is called *conditional*, if $\forall a \in A : |p(s_j)(a)| < \omega$.

Let us denote the set of n-ary relations over A by $R_n(A)$. A function $F : R_n(A) \longmapsto R_n(A)$ is monotone if $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$. As it is well known every monotone function over a complete lattice has a minimal (least) and a maximal (greatest) fixpoint. The minimal fixpoint of the monotone function F is $\mu X : F(X) = \bigcap\{X | F(X) \subseteq X\}$, and the maximal fixpoint of F is $\eta X : F(X) = \bigcup\{X | X \subseteq F(X)\}$ [17].

1.1.2 The concepts of problem, parallel program and solution

The specification of a problem and its solution, the abstract program, is independent of architecture, scheduling and programming language. The abstract program is regarded as a relation generated by a set of deterministic (simultaneous) conditional assignments similar to the concept of abstract program in UNITY [2]. The conditions of the assignments encode the necessary synchronization restrictions explicitly. Some assignments are selected nondeterministically and executed in each step of the execution of the abstract program. Every statement is executed infinitely often, i.e., an unconditionally fair scheduling is postulated. The concept of fairness is used in the same sense as by Morris in [15] (Section 5.1), i.e., stricter than usually [2]. If more than one processor selects statements for execution, then the executions of different processors are fairly interleaved. A fixed point is said to be reached in a state, if none of the statements changes that state [2].

1.1.3 The specification of a problem

The problem is defined as a set of properties. Every property is a relation over the powerset of the sate space. Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions. We define $\rhd, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and $\mathrm{FP}, \mathrm{INIT}, \mathrm{inv}, \mathrm{TERM} \subseteq \mathcal{P}(A)$.

We introduce the following infix notations:

$P \rhd Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \rhd$, $\quad P \mapsto Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \mapsto$,
$P \hookrightarrow Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow$, $\quad \mathrm{FP} \Rightarrow R ::= \lceil R \rceil \in \mathrm{FP}$,
$Q \hookrightarrow \mathrm{FP} ::= \lceil Q \rceil \in \mathrm{TERM}$, $\quad Q \in \mathrm{INIT} ::= \lceil Q \rceil \in \mathrm{INIT}$,
$\mathrm{inv}\ P ::= \lceil P \rceil \in \mathrm{inv}$.

The $P \rhd Q, P \mapsto Q$, etc. formulas are called specification properties or shortly properties. The $\rhd, \mapsto, \hookrightarrow, \mathrm{inv}, \mathrm{TERM}$ relations define transition properties, the FP, INIT relations define boundary properties. The transition relations $\rhd$ and inv express so called safety properties, while the relations $\mapsto, \hookrightarrow, \mathrm{TERM}$ express progress properties. The definition of a solution gives an interpretation for the introduced concepts.

Definition 1.1 *Let A be a state space and let B be a finite or numerable set. Two relations expressing boundary properties and four relations expressing transition properties are associated to every point of the set B. The relation $F \subseteq B \times (\times_{i\in[1..3]} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \times_{i\in[1..4]} \mathcal{P}(\mathcal{P}(A)))$ is called a problem defined over the state space A. B is called the parameter space of the problem. The components of the elements of the direct products $\times_{i\in[1..3]} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$ and $\times_{i\in[1..4]} \mathcal{P}(\mathcal{P}(A))$ are denoted by $\rhd_b, \mapsto_b, \hookrightarrow_b$ and by $\mathrm{INIT}_b, \mathrm{FP}_b, \mathrm{inv}_b, \mathrm{TERM}_b$ respectively.*

A program satisfies the safety property $P \rhd Q$, if and only if there is no direct transition from $P \wedge \neg Q$ to $\neg P \wedge \neg Q$ only through Q if any. A program satisfies the progress properties $P \mapsto Q$ or $P \hookrightarrow Q$ if the program starting from P inevitably reaches a state, in which Q holds. $P \mapsto Q$ defines further restriction for the direction of progress. The fixed point property $\mathrm{FP} \Rightarrow R$ defines necessary conditions for the case when the program is in one of its fixed point. The $Q \in \mathrm{INIT}$ property defines sufficient condition for the initial states of the program. $Q \hookrightarrow \mathrm{FP}$ expresses that the program starting from Q inevitably reaches one of its fixed points. P is said to be *stable* if and only if $P \rhd \downarrow$. If P holds initially and P is stable, then P is an **invariant**, denoted by inv P.

1.1.4 The definition of a parallel program

Let S be an ordered pair of a conditional assignment and a nonempty, finite set of conditional assignments, such that $S = (s_0, \{s_j \mid j \in J \ \wedge \ \mathcal{D}_{p(s_j)} = A \ \wedge \forall a \in A : (|s_j(a)| < \omega)\}), J = \{1..m\}, m \geq 1$.

The program $UPG(S)$ is a binary relation which associates equivalence classes of graphs generated by the effect relation of s_0 and by disjoint union of the effect relations of conditional assignments $\{s_1, ..., s_m\}$ to the points of the state space. The formal definition of a parallel program is given in [13]. The program $UPG(S)$ generated by the ordered pair $S = (s_0, \{s_1, ...s_m\})$ is denoted shortly by S. The conditional assignment s_0 is called the initialization in S and $s_j : j \in [1..m]$ is said to be an element of the program S.

1.1.5 The formal definition of a solution

The program S solves the problem F, if S satisfies all (subset of) the properties given in F. The justification of the following definitions and the proofs of the theorems is given in [11,13].

Definition 1.2 *Let S be an abstract program, $S = (s_0, \{s_1, ...s_m\})$. Let us denote the set of the indices of the deterministic assignments of abstract program S by J_d and the the set of the indices of the nondeterministic assignments by J_{nd}. $fixpoint_S ::= (\bigwedge_{j \in J_d, i \in [1..n]} (\pi_{j_i} \rightarrow a_i = F_{j_i}(a)) \wedge (\bigwedge_{j \in J_{nd}, i \in [1..n]} (\neg \pi_{j_i})))$.*

Definition 1.3 *Let S be an abstract program. S satisfies (FP $\Rightarrow$ R) if $fixpoint_S \Rightarrow R$.*

Definition 1.4 *Let S be an abstract program, $S = (s_0, \{s_1, ...s_m\})$. $wp(S, R) ::= \forall s \in S : wp(s, R)$. $wpa(S, R) ::= \exists s \in S : wp(s, R)$.*

($wpa(S, R)$ is called the "angelic" weakest precondition [15]).

Definition 1.5 *The program S satisfies the property $P \rhd Q$ if and only if $(P \wedge \neg Q \Rightarrow wp(S, P \vee Q))$.*

Definition 1.6 *The program S satisfies the pair of properties $Q \in INIT$ and* inv P *if and only if $sp(s_0, Q) \Rightarrow P$ and P is stable.*

Definition 1.7
$G(P, Y, X) ::= P \vee (wpa(S, Y) \wedge wp(S, X \vee Y))$,
$F(P, Y) ::= \eta X : G(P, Y, X)$, and
$\leadsto P ::= \mu Y : F(P, Y)$.

Remark: Since G is monotone in P, Y, X, $\forall P, Y : \eta X : G(P, Y, X)$ exists, moreover $F(P, Y)$ is monotone in P, Y and $\leadsto P$ is monotone in P.

Definition 1.8 *(ensures) The program S satisfies the specification $(Q \mapsto P)$ if and only if $(Q \Rightarrow (P \vee (wpa(S, P) \wedge wp(S, Q \vee P))))$, i.e., $(Q \Rightarrow G(P, P, Q))$.*

Definition 1.9 *(leads-to, inevitable) The program S satisfies the specification $(Q \hookrightarrow P)$ if and only if $(Q \Rightarrow (\leadsto P))$.*

Theorem 1.1 *If $(\leadsto P)$ holds for $a \in A$, the scheduling is unconditionally fair and the program S is in the state a, then S inevitable reaches a state, for which P holds.*

We can prove the following theorems corresponding to the properties used in the definition of leads-to in UNITY [2]. The proof of progress properties is supported by the introduction of so called variant functions [6,2].

Theorem 1.2 *For an arbitrary program S,*
- *if $P \mapsto Q$ then $P \hookrightarrow Q$, and*
- *if $P \hookrightarrow Q$ and $Q \hookrightarrow R$, then $P \hookrightarrow R$.*
- *Let I be an arbitrary finite set. If $\forall i \in I : (P_i \hookrightarrow Q)$ then $(\exists i : P_i) \hookrightarrow Q$.*
- *Let W be a well-founded set in respect of the relation $<$.*
  If $\forall m \in W :: (P \wedge v = m) \hookrightarrow ((P \wedge v < m) \vee Q)$, then $P \hookrightarrow Q$.

Consequence 1.1 *If the program S satisfies the property: $(\neg\ fixpoint_S\ \wedge v = v') \mapsto ((\neg\ fixpoint_S\ \wedge v \leq v' - 1) \vee\ fixpoint_S)$, then S satisfies the property $(\uparrow\hookrightarrow\ fixpoint_S)$, i.e., $(\uparrow\hookrightarrow \mathrm{FP})$.*

A new specification is called a refinement of a previous one, if any solution for the new specification is a solution for the problem specified originally.

2 Computation of the values of an associative function

Let H be a set. Let $\circ : H \times H \longmapsto H$ denote an arbitrary associative binary operator over H.

$f : H^* \longmapsto H$ is a function describing the single or multiple application of the operator $\circ$. Since $\circ$ is associative, for any arbitrary sequence $x \in H^*$ of length at least three
$f(\ll x_1, ..., x_{|x|} \gg) = f(\ll f(\ll x_1, .., x_{|x|-1} \gg), x_{|x|} \gg) = f(\ll x_1, f(\ll x_2, .., x_{|x|} \gg) \gg)$. We write $f(\ll h_1, h_2 \gg)$ instead of the infix notation $(h_1 \circ h_2)$ in the following. We extend f for sequences of length one: $f(\ll h \gg) = h$.

Let a finite sequence $a \in H^*$ of the elements of H be given. The indices are associated to the elements of the sequence a in the reverse order, i.e., the last element is denoted by a_1. If the length of the sequence is n, then the first element is denoted by a_n. $a = \ll a_n, ..., a_1 \gg$, $(n \geq 1)$. Let us compute the value of the function $\mathcal{G} : [1..n] \longmapsto H$ for all $i \in [1..n]$, where $n \geq 1$ and

$$\mathcal{G}(i) = f(\ll a_i, ..., a_1 \gg).$$

To solve the problem we use a similar train of thought to those presented in the cases of parallel synchronous computation of the sum of binary numbers and of the asynchronous computation of the shortest path [2].

2.1 The formal specification of the problem

We specify that the program inevitably reaches a fixed point and the array g contains the values of f in any fixed point.

$A = G$, where $G = vector([1..n], H)$, $n \geq 1$; $g : G$

$$\uparrow\hookrightarrow \mathrm{FP} \tag{1}$$

$$\mathrm{FP} \Rightarrow (\forall i \in [1..n] : g(i) = f(\ll a_i, ..., a_1 \gg)) \tag{2}$$

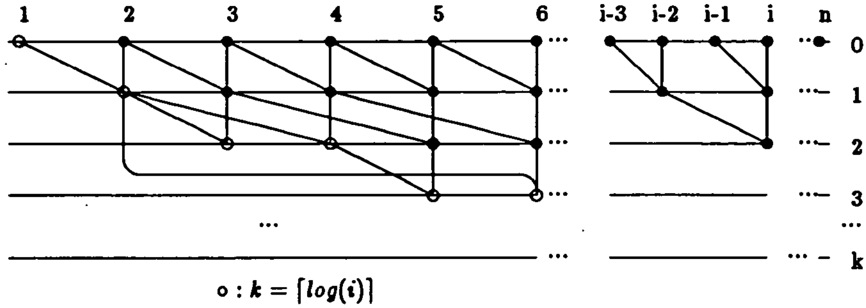

Figure 1: $gs(i,k) = h(i,k)$, if $k \le \lceil k(i)$.

Let us observe that the computation of the values of $\mathcal{G}$ at place i is made easier with the knowledge of the value of f for subsequences $f(\ll a_u, ..., a_v \gg)$ indexed by the elements of an arbitrary $[u..v] \subseteq [i..1]$ interval. Moreover the result computed for a subsequence is useful in the computation of the value of f for any sequence which includes the subsequence.

From the above line of reasoning, we extend the state space and refine the specification of the problem. Let us introduce the auxiliary function h. Let $h(i,k)$ denote the value of f for the sequence of which the first element is a_i and its length is 2^k or the last element is a_1, if $i < 2^k$. The two-dimensional array gs is introduced to store the known values of h. This method is called the substitution of a function by a variable [7]. The connection between the variables gs, k, t and the function h is given by the invariants (4)-(6). The lines on the Figure 1 illustrate the connections among the elements of the matrix gs according to lemma 2.1 and to invariants (4)-(6).

$$
\begin{array}{llllllll}
A' = & G\times & GS\times & K\times & T & G & = vector([1..n], H), & \\
 & g & gs & k & t & GS & = vector([1..n, 0..(\lceil \log(n) \rceil)], H) & \\
 & & & & & K & = vector([1..n], \mathcal{N}_0), & \\
 & & & & & T & = vector([1..n], \mathcal{N}_0), & n \ge 1
\end{array}
$$

The precise definition of the partial function $h : [1..n] \times \mathcal{N}_0 \longrightarrow H$ is:

$$
h(i,k) ::= \begin{cases} f(\ll a_i, ..., a_1 \gg), & \text{if } i - 2^k + 1 \le 1 \\ f(\ll a_i, ..., a_{(i-2^k+1)} \gg), & \text{if } i - 2^k + 1 \ge 1 \end{cases}
$$

Lemma 2.1
If $(i - 2^k \ge 1)$, *then* $f(\ll h(i,k), h(i-2^k,k) \gg) = h(i,k+1)$.

Proof: Since $i - 2^k \ge 1$, $h(i,k) = f(\ll a_i, ..., a_{(i-2^k+1)} \gg)$. If $(i-2^k) - 2^k + 1 \ge 1$, then $h(i-2^k,k) = f(\ll a_{(i-2^k)}, ..., a_{(i-2^k-2^k+1)} \gg)$. Since f is associative: $f(\ll h(i,k), h(i-2^k,k) \gg) = f(\ll a_i, ..., a_{(i-2^k+1)}, a_{(i-2^k)}, ..., a_{(i-2^k-2^k+1)} \gg) = h(i,k+1)$. If $(i-2^k) - 2^k + 1 < 1$, then $h(i-2^k,k) = f(\ll a_{(i-2^k)}, ..., a_1 \gg)$. Using the associativity of f: $f(\ll h(i,k), h(i-2^k,k) \gg) = f(\ll a_i, ..., a_{(i-2^k+1)}, a_{(i-2^k)}, ..., a_1 \gg) = h(i,k+1)$.

Let us choose the variant function $v : A \longmapsto \mathcal{N}_0$ in the following way:

$$v::=4*n*n-\sum_{i=1}^{n}(k(i)+\chi(k(i)=\lceil\log(i)\rceil\wedge g(i)=gs(i,k(i))))$$

The variant function depends on the number of elements of the matrix gs which elements are different from the value of function h at the corresponding place and on the number of places where the value of the array g is different from the value of function $\mathcal{G}$.

Lemma 2.2 *The specification below is a refinement of the specification (1)-(2).*

$$\uparrow\hookrightarrow \mathrm{FP} \tag{3}$$

$$\mathrm{FP}\Rightarrow \forall i\in[1..n]:(k(i)=\lceil\log(i)\rceil)\ \wedge(g(i)=gs(i,\lceil\log(i)\rceil)) \tag{4}$$

$$\mathrm{inv}\ (\forall i\in[1..n]:k(i)\le\lceil\log(i)\rceil\wedge\forall k:k\le k(i):gs(i,k)=h(i,k)) \tag{5}$$

$$\mathrm{inv}\ (\forall i\in[1..n]:t(i)=2^{k(i)}) \tag{6}$$

Proof:
$k(i)=\lceil\log(i)\rceil$ and $g(i)=gs(i,\lceil\log(i)\rceil)$ in fixed point according to (4). Using (5) it follows that the equation $g(i)=gs(i,\lceil\log(i)\rceil)=h(i,\lceil\log(i)\rceil)$ holds in fixed point. Since $2^{\lceil\log(i)\rceil}\ge i$, after the application of the definition of h we get $h(i,\lceil\log(i)\rceil)=f(\ll a_i,\dots,a_1\gg)$, which is the same as property (2).

Remark 2.1 The property (1) is not refined. The proof of the correctness of any program in respect of (1)=(3) is based on Consequence 1.1. This means the choose of a variant function may be regarded as an implicit refinement step in respect of property (1). Since the property (6) defines restrictions over the new components of the state space only, we need not to use it in the proof of the refinement.

2.2 A solution

Theorem 2.1 *The abstract program below is a solution for the problem specified by (3)-(6), i.e., a solution for the problem of the computation of the values of an associative function.*

$$s_0:\quad \underset{i=[1..n]}{\Box}\, gs(i,0),t(i),k(i):=f(\ll a_i\gg),1,0$$

$$S:\{\quad \underset{i=[1..n]}{\Box}\, gs(i,k(i)+1),t(i),k(i):=$$

$$\begin{cases} f(\ll gs(i,k(i)), & gs((i-t(i)),k(i))\gg),2*t(i),k(i)+1,\\ & \textit{if } (i-2*t(i)+1\ge 1)\wedge(k(i-t(i))\ge k(i))\\ f(\ll gs(i,k(i)), & gs(i-t(i),k(i-t(i)))\gg),\\ & \quad 2*t(i),k(i)+1,\\ & \textit{if } (i-t(i)\ge 1)\wedge(i-2*t(i)+1<1)\\ & \quad \wedge(k(i-t(i))=\lceil\log(i-t(i))\rceil) \end{cases}$$

$$\underset{i=[1..n]}{\Box}\, g(i):=gs(i,k(i))\,\textit{if } (k(i)=\lceil\log(i)\rceil)$$

$$\}$$

where $\underset{i=[1..n]}{\Box}$ *is used for the abbreviation of n statements. Each statement is instantiated from the general form by substituting the dummy variable i by a concrete value.*

Proof:

(3): Every statement of the program decreases the variant function by 1 or does not cause state transition. If the program is not in one of its fixed points, then there exists an $i \in [1..n]$ and a corresponding conditional assignment, which assignment increases the value of $k(i)$, or there exists an i for which $k(i) = \lceil \log(i) \rceil$ and the value of $g(i)$ is different from the value of $gs(i, (\lceil \log(i) \rceil))$.

(4): using the definition of the $fixpoint_S$:

$$\forall i \in [1..n] \qquad (k(i) = \lceil \log(i) \rceil) \rightarrow g(i) = gs(i, k(i)) \ \wedge \tag{7}$$

$$((i - 2 * t(i) + 1 < 1) \vee (k(i - t(i)) < k(i))) \ \wedge \tag{8}$$

$$(i - t(i) < 1) \vee \quad (i - 2 * t(i) + 1 \geq 1) \vee (k(i - t(i)) \neq \lceil \log(i - (t(i))) \rceil)) \tag{9}$$

We apply mathematical induction on i to prove: $\forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil)$. Base case: $i = 1$. From (5) and $sp(s_0, \uparrow)$ it follows that $(k(1) = \lceil \log(1) \rceil)$. Inductive hypothesis: $\forall j < i : (k(j) = \lceil \log(j) \rceil)$. Since $t(i) \geq 1$, $(k(i - t(i)) \neq \lceil \log(i - (t(i))) \rceil)$ contradicts the hypothesis. This means (9) can be simplified to $(i - t(i) < 1) \vee (i - 2 * t(i) + 1 \geq 1)$. If $(i - 2 * t(i) + 1 \geq 1)$, then $k(i - t(i)) < k(i)$ else (8) does not hold. Using the inductive hypothesis and $t(i) \geq 1$ we get $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$, i.e., $\lceil \log(i - t(i)) \rceil < k(i)$. The last statement contradicts the initial condition: $(i - 2 * t(i) + 1 \geq 1)) \Rightarrow (i - t(i) - t(i) + 1 \geq 1) \Rightarrow \lceil \log(i - t(i)) \rceil \geq k(i)$. This means $(i - 2 * t(i) + 1 < 1)$.

$(i - 2 * t(i) + 1 < 1) \Rightarrow (i - t(i) < 1)$, otherwise (9) does not hold. $(i - t(i) < 1) \Rightarrow k(i) \geq \lceil \log(i) \rceil$. Using the invariant (5) we get: $k(i) = \lceil \log(i) \rceil$. Based on (7) : $g(i) = gs(i, k(i)) = gs(i, \lceil \log(i) \rceil)$.

(6): Since $sp(s_0, \uparrow)$ implies $t(i) = 1$ and $k(i) = 0$, the $t(i) = 2^{k(i)}$ equality holds initially. All the assignments change the value of $k(i)$ and $t(i)$ simultaneously.

(5): Since $h(i, 0) = f(\ll a(i) \gg)$, $sp(s_0, \uparrow) \Rightarrow gs(i, k(i)) = h(i, k(i))$. Since $k(i)$ is initially 0, $sp(s_0, \uparrow) \Rightarrow (k(i) \leq \lceil \log(i) \rceil)$.

After calculating the weakest preconditions of the assingments it is sufficient to show that

- $(i - 2 * t(i) + 1 \geq 1) \wedge (k(i - t(i)) \geq k(i))$ and $\forall k : k \leq k(i) : gs(i, k) = h(i, k)$ implies the equality for $k(i) + 1$, i.e., $f(\ll gs(i, k(i)), gs(i - t(i), k(i)) \gg) = h(i, k(i) + 1)$ and $k(i) + 1 \leq \lceil \log(i) \rceil$,
- $(i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge (k(i - t(i)) = \lceil \log(i) \rceil$ and $\forall k : k \leq k(i) : gs(i, k) = h(i, k)$ implies the equality for $k(i) + 1$, i.e., $f(\ll gs(i, k(i)), gs(i - t(i), (\lceil \log(i - t(i)) \rceil)) \gg) = h(i, k(i) + 1)$ and $k(i) + 1 \leq \lceil \log(i) \rceil$.

$(i - 2 * t(i) + 1 \geq 1) \wedge (t(i) \geq 1) \Rightarrow (i - t(i) \geq 1) \Rightarrow k \leq \log(i - 1) < \log(i) \leq \lceil \log(i) \rceil)$.

In the first case $k(i) \leq k(i)$ implies $gs(i, k(i)) = h(i, k(i))$ and $(k(i - t(i)) \geq k(i))$ implies $gs(i - t(i), k(i)) = h(i - t(i), k(i))$. In the second case $k(i) \leq k(i)$ implies $gs(i, k(i)) = h(i, k(i))$ and $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$ implies $gs(i - t(i), (\lceil \log(i) \rceil)) = h(i - t(i), (\lceil \log(i) \rceil))$. In both of the cases the application of the Lemma 2.1 leads to the statement.

(end of proof.)

Let us suppose the abstract program is implemented on a parallel computer containing $O(n)$ processors. If the left side of an assignment refers to an array component indexed by i, then the assignment is mapped to the ith (logical) processor. Easy to see, that the program reaches one of its fixed point in at most $O\lceil\log(n)\rceil$ state transforming steps. The logical processors may work asynchronously.

2.3 Transformation of the program

The program corresponds neither to the rule of fine-grain atomicity [1](2.4) nor to the shared variable schema [2]. To ensure effective asynchronous computation we have to transform the program by introducing new variables and using the method of substitution of a function by a variable for the function log [7].

Let us use the auxiliary arrays $gst(i) = gs(i-t(i), k(i))$, $kt(i) = k(i-t(i))$, $gstk(i) = gs(i-t(i), kt(i))$, if the values are necessary and known by the ith logical processor and the value of $kt(i)$ is big enough to determine the next (i.e. the $(k(i)+1)$th) value of the ith column of the matrix gs (10). Let us introduce the auxiliary boolean variables $ktf(i), gstf(i), gstkf(i)$ to administrate the usage of the auxiliary arrays. The ith component of the auxiliary arrays is local in respect of the ith processor.

Every assignment of the transformed program will refer to at most one nonlocal variable.

2.3.1 The refinement of the specification

We extend the specification (3)-(6) with the following invariants:

$$\text{inv } \forall i \in [1..n]: \quad (kt(i) \le k(i-t(i)) \ \wedge \ ktf(i) \rightarrow (kt(i) \ge k(i) \vee kt(i) = l(i-t(i)))) \tag{10}$$

$$\text{inv } \forall i \in [1..n]: \quad (gstf(i) \rightarrow ktf(i) \wedge (i - 2 * t(i) + 1 \ge 1) \ \wedge gst(i) = gs(i-t(i), k(i))) \tag{11}$$

$$\text{inv } \forall i \in [1..n]: \quad (gstkf(i) \rightarrow ktf(i) \wedge (i - t(i) \ge 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge gstk(i) = gs(i-t(i), kt(i)) = gs(i-t(i), k(i-t(i)))) \tag{12}$$

$$\text{inv } \forall i \in [1..n]: \quad \lceil\log(i)\rceil = l(i) \tag{13}$$

2.3.2 The transformed program

$$s_0: \quad \underset{i=[1..n]}{\square} gs(i,0), t(i), k(i), l(i), ktf(i), gstkf(i), gstf(i), kt(i) := f(\ll a_i \gg), 1, 0, \lceil \log(i) \rceil, \downarrow, \downarrow, \downarrow, 0$$

$$S: \{ \quad \underset{i=[1..n]}{\square} kt(i) := k(i - t(i)), \text{if } \neg ktf(i) \wedge (i - t(i)) \geq 1$$

$$\underset{i=[1..n]}{\square} ktf(i) := \uparrow, \text{if } \neg ktf(i) \wedge (i - t(i)) \geq 1 \wedge (kt(i) \geq k(i) \vee kt(i) = l(i - t(i)))$$

$$\underset{i=[1..n]}{\square} gst(i), gstf(i) := gs(i - t(i), k(i)), \uparrow, \text{ if } ktf(i) \wedge (i - 2 * t(i) + 1 \geq 1) \wedge (kt(i) \geq k(i)) \wedge \neg gstf(i)$$

$$\underset{i=[1..n]}{\square} gstk(i), gstkf(i) := gs(i - t(i), kt(i)), \uparrow, \text{ if } ktf(i) \wedge (i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge (kt(i) = l(i - t(i))) \wedge \neg gstkf(i)$$

$$\underset{i=[1..n]}{\square} gs(i, k(i) + 1), t(i), k(i), ktf(i), gstf(i), gstkf(i), kt(i) :=$$

$$\begin{cases} f(\ll gs(i, k(i)), & gst(i) \gg), 2 * t(i), k(i) + 1, \downarrow, \downarrow, \downarrow, 0 \\ & \text{if } gstf(i) \\ f(\ll gs(i, k(i)), & gstk(i) \gg), 2 * t(i), k(i) + 1, \downarrow, \downarrow, \downarrow, 0 \\ & \text{if } gstkf(i) \end{cases}$$

$$\underset{i=[1..n]}{\square} g(i) := gs(i, k(i)), \quad \text{if } k(i) = l(i)$$

$$\}$$

Proof: The invariants (10)-(13) are easy to prove by the calculation of the weakest preconditions and $sp(s_0, \uparrow)$. Using the invariants (10)-(13) we can state that the assignments changing the variables mentioned in (3),(5)-(6) are equivalent of the original assignments. This means the specification properties (3),(5)-(6) remain valid for the transformed program too. To prove the fixpont property (4) it will be sufficient to show: if the transformed program reaches one of its fixed points then the original program is in one of its fixed points too and the conditions (7)-(9) hold.□

3 Discussion

The program is easy to implement on synchronous, asynchronous and on distributed architectures, such as for hypercubes [18] or T9000 transputer networks, where implementation of $O(\lceil \log(n) \rceil)$ communication channels is supported by the concepts of logical links.

A solution is developed in [14] for pipeline architectures.

The introduced relational model provides effective tools for the stepwise development of a parallel solution as illustrated by the chosen example. The theorem 2.1

may be called a programming theorem [6]. With its help we can solve a class of classical problems. For example parallel addition, comparison of ascending sequences [2], etc. are easy to formalize by the help of associative functions.

References

[1] Andrews, G.R.: *Concurrent Programming, Principles Practice*, Benjamin/Cummings, 1991.

[2] Chandy, K.M., Misra, J.: *Parallel program design: a foundation*, Addison-Wesley, 1988, (1989).

[3] Dijkstra, E.W.: *A Discipline of Programming*, Prentice-Hall, 1976.

[4] Dijksta, E.W., Scholten, C.S.: *Predicate Calculus and Program Semantics*, Springer-Verlag, 1989.

[5] Emerson, E.A., Srinivasan, J.: Branching Time Temporal Logic, in *Linear Time, Branching Time and Partial Order in Logics an Models for Concurrency, LNCS 354*. Springer-Verlag 1989, 123-172.

[6] Fóthi Á.: *Introduction into Programming (Bevezetés a programozáshoz)*, in Hungarian, ELTE TTK, Budapest, 1983.

[7] Fóthi Á.: verbal communications

[8] Fóthi Á.: A Mathematical Approach to Programming, *Annales Uni. Sci. Budapest. de R. Eötvös Nom. Sectio Computatorica*, Tom. IX. (1988), 105-114.

[9] Fóthi Á., Horváth Z.: The Weakest Precondition and the Theorem of the Specification, in *Proceedings of the Second Symposium on Programming Languages and Software Tools*, Pirkkala, Finland, August 21-23, 1991, Eds.: Kai Koskimies and Kari-Jouko Räihä, Uni. of Tampere, Dep. of Comp. Sci. Report A-1991-5, August, 1991, 39-47.

[10] Horváth Z.: Fundamental relation operations in the mathematical models of programming, *Annales Uni. Sci. Budapest. de R. Eötvös Nom. Sectio Computatorica*, Tom. X. (1990), 277-298.

[11] Horváth Z.: The Weakest Precondition and the the Specification of Parallel Programs, in *Proceedings of the Third Symposium on Programming Languages and Software Tools*, Käariku, Estonia, August 21-23, 1993, 24-33.

[12] Horváth Z., Kozma L.: Parallel Programming Methodology, to appear in *Proceedings of the Workshop on Parallel Processing, Technology and Applications*, Technical University Budapest, February 10-11, 1994.

[13] Horváth Z.: The Formal Specification of a Problem Solved by a Parallel Program - a Relational Model, in Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 9-10, 1995, 165-189.

[14] Loyens, L.D.J.C., van de Vorst, J.G.G.: Two Small Parallel Programming Exercises, *Science of Computer Programming* Vol. 15(1990), 159-169.

[15] Morris, J., M.: Temporal Predicate Transformers and Fair Termination, *Acta Informatica*, Vol. 26, 287-313, 1990.

[16] Nyéky-Gaizler J., Konczné-Nagy M., Fóthi Á., Harangozó É.: Demonstration of a problem solving method, *Acta Cybernetica* 12 (1995) 71-82.

[17] Park, D.: On the semantics of fair parallelism, in *LNCS 86*, 504-526, Springer 1980.

[18] Quinn, M., J.: *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, Inc., 1987.

[19] Varga L.: *Programok analízise és szintézise*, Akadémiai Kiadó, Budapest, 1981.

Received, July, 1994