WAVE COMPUTATION: LANGUAGE AND VERIFICATION

Technical Report CSE-92-03

M.D. Rice
Mathematics Department
Wesleyan University
Middletown, CT 06459

S.B. Seidman
Department of Computer Science and Engineering
Auburn University
Auburn, AL 36849-5347

July 29, 1992

Wavefront Computation: Language and Verification

M. D. Rice

Mathematics Department
Wesleyan University
Middletown, CT 06459

S.B. Seidman

Department of Computer Science
and Engineering
Auburn University, Auburn, AL 36849

ABSTRACT

Wavefront computation was first proposed by S. Y. Kung as a means of abstracting the computational nature of systolic hardware arrays. The wavefront model combines data-driven computation with regular geometry. Despite the model's attractiveness, it has neither received serious study as a model of parallel computation, nor has it been used as the basis for practical parallel programming. In our recent work, we have been investigating the wavefront paradigm using a computational model based on a general class of interconnection networks that support precisely defined wavefront operations and a formal notion of boundary. In this paper, the semantics of a specification language is described for the special case of n-dimensional grids and used to verify rectangular wavefront algorithms.

0. Introduction

In the past decade, both systolic arrays and wavefront arrays ([K1], [K2]) have been suggested as models of synchronized parallel computation. Recently, significant efforts in the areas of software, hardware, and algorithms have been directed at these computational models. In particular, there has been considerable work on the synthesis of a systolic array from an algorithmic description ([HL], [Li], [Q]), the mapping of algorithms onto systolic models ([BL], [CCL]), and proofs of correctness for systolic algorithms ([He], [P]). A limited amount of work has also been done on designing suitable high-level languages and environments for expressing systolic and wavefront algorithms ([A], [EC], [K3], [Le], [S1,2]).

Our research efforts during the past few years have been directed toward developing a unified treatment of language, verification, computational model, and implementation in a wavefront context. The basis of our approach is a precise identification of the elements needed to describe a generalized type of wavefront computation. These ideas have been used to develop the wavefront specification language whose syntax and semantics are presented in Section 1. Wavefront algorithms specified using this language can be verified using the framework provided by the semantics. Specifications for two typical wavefront algorithms are presented in Section 2 and the verification of a non-trivial Gaussian elimination algorithm is presented in Section 3.

An operational semantics for a two-dimensional version of the specification language has been described using CSP and additional segments of occam code. This description has been used as

the basis for a prototype multiprocessor implementation of the two-dimensional wavefront model on Transputer networks. A complete discussion of the operational semantics and implementation is found in [RS].

1. Wavefront Specification Language

The wavefront computational model used in this paper generalizes the basic wavefront array model described in [K2] and [K3]. The basic model consists of a two-dimensional grid of computational cells with a north-south-east-west interconnection network, where the computation in each cell is data-driven. The term *wavefront* refers to the fact that computational activity starts at the upper left corner of the grid and proceeds downward diagonally to the lower right corner. Each cell waits for the arrival of the data in the wavefront, performs a computation that is the same for all cells in the network, and then generates the data needed to propagate the wavefront.

The generalized wavefront model extends the basic model in several ways. First, the generalized model allows the specification of an n-dimensional grid of virtual cells. The wavefront computation starts on a set of *input cells* which is the intersection of (n - 1)-dimensional boundary sets. These cells receive wavefront data from the environment. As in the basic model, the wavefront data propagates through the network. Finally, a corresponding set of *output cells* sends data back to the environment. Secondly, each cell can perform an initial pure cell computation during which it does not communicate with other cells. Third, each cell may perform a different computation based on either its geometric position or the number of wavefronts that it has processed. All computations within individual cells are performed sequentially. Finally, cell variables can be initialized by values from the environment and the final values of cell variables can be stored in the environment at the conclusion of the wavefront computation.

The specification language reflects the generalized wavefront model. The specification of a wavefront algorithm consists of two portions: an *instantiation* and a *cell declaration.*

<Specification> ::= <Instantiation> <Cell_decl>

An instantiation uses <Coordinate_list> to specify the dimension and size of the network and <Boundary_ports> to specify ports which handle the movement of wavefront data to and from the network.

```
<Instantiation> ::= <GConst_defn>
                    USING <Boundary_ports>
                      EXECUTE NETWORK(<Coordinate_list>)
                        OF CELL (<External_ports>)
                      FOR NUMWAVES = <GExp>
```

The cell declaration specifies an individual cell computation. The <Parameter_list> specifies the cell variables whose values will be either initialized by, or stored in, the environment. The corresponding locations in the environment are specified in the instantiation by <External_ports>.

<Cell_decl> ::= CELL (<Parameter_list>) =
<Local_defns_and_decls>
{ [<Cell_comp>] [<Systolic_comp>] }

The complete syntax of the language is presented in the Appendix.

The semantics of the language is based on semantic rules given by a set of equations which describe the values of expressions and standard axiomatic semantics for sequential statements in Dijkstra's language of guarded commands. The intention is to support the verification of algorithms by providing precise meanings for the language constructs. We will ignore many issues that are usually included in a formal semantic description.

Notation: $\mathbf{N}$ and $\mathbf{Z}$ will denote the set of natural numbers and integers, respectively. Given $m, n \in \mathbf{Z}$, $[m, n] = \{x \in \mathbf{Z} : m \le x \le n\}$. Assume $b, i \in \mathbf{Z}$, and $I \subseteq \mathbf{Z}$. If $p^* \in \mathbf{Z}^I$, then $p = p^* \oplus \{i \to b\} \in \mathbf{Z}^{I \cup \{i\}}$ is defined by $p_j =$ **if** $(j = i)$ **then** b **else** p^*_j, $j \in I \cup \{i\}$.

The semantic rules are based on two expression evaluators which specify the values of global and local expressions. The first of these is

$$E : \mathbf{GlobalExp} \to \mathbf{Values}_\perp,$$

where **GlobalExp** denotes the expressions used in the instantiation and the flat domain $\mathbf{Values}_\perp$ denotes the set of storable values (including integers and reals) augmented with the undefined element $\perp$.

Given the following portion of an instantiation

NETWORK($e_1 : f_1, e_2 : f_2, ..., e_d : f_d$) OF
CELL (<External_ports>)
FOR NUMWAVES = g,

define $NumWaves = E(g)$, $m_i = E(e_i)$, and $n_i = E(f_i)$, $1 \le i \le d$.

We assume that

- $\{m_i\} \cup \{n_i\} \cup \{NumWaves\} \subseteq \mathbf{Z}$
- $\forall i \bullet m_i \le n_i$
- $NumWaves \ge 0$.

NumWaves represents the number of waves of data processed by each cell during the wavefront computation. If *NumWaves* = 0, no wavefront computation is performed. The integers $\{m_i\}$ and $\{n_i\}$ represent the lower and upper bounds, respectively, of the cell coordinates.

An alternative form of the network specification is

$$\textbf{NETWORK}(e : f : h)\ \textbf{OF}$$

If this form is used, we assume that $d = E(h)$ is a positive integer, and for each $1 \le i \le d$, $m_i = E(e)$ and $n_i = E(f)$ satisfy the restrictions stated above.

The second expression evaluator is

$$[\ ] : \textbf{LocalExp} \to \textbf{Indices} \multimap \textbf{Stages} \multimap \textbf{Values}_{\perp},$$

where **LocalExp** denotes the expressions used in cell declarations, **Stages** = **N** denotes the possible stages of a computation, and **Indices** = $\mathbf{Z}^d$ denotes the possible cell coordinates.

Define $$\mathbf{P} = \prod\{[m_i, n_i] : 1 \le i \le d\}.$$

Given $e \in$ LocalExp, the domain of [e] is P and for each $p \in$ P, the domain of [e]p is [0, *NumWaves*]. In addition,

- [e]ps denotes the *initial value* of e in cell p at the beginning of wavefront s
- [e']ps denotes the *final value* of e in cell p after wavefront s has been processed.

If $s = 0$, the preceding values refer to the pure cell computation, while if $s > 0$, the values refer to the wavefront computation. In general, we assume that [ ] commutes with the standard unary and binary arithmetic operators, preserves the values of constants, and respects the usual precedence rules for expressions.

The movement of data between the cells and the environment involves both *external* and *boundary* ports. These ports correspond to d-dimensional *cell* data and (d-1)-dimensional *boundary* data, respectively.

An *external* port is a partial mapping

$$\textbf{Indices} \multimap \textbf{Values}$$

which has domain P and thus corresponds to a variable that has a value in each cell. The variable corresponding to an external port will either be initialized prior to any computation or will have its final value stored in the environment after the completion of the wavefront computation.

A *boundary* port is a partial mapping

$$Z^{d-1} \rightarrow\!\!\!|\ \rightarrow N \rightarrow\!\!\!|\ \rightarrow \text{Values}$$

which specifies a stream of data values for each cell in some boundary set of P. Boundary ports can be distinguished as *input ports* and *output ports*. There is one input port and one output port for each $1 \le i \le d$. These ports have domain

$$P_i = \prod\{[m_j, n_j]: 1 \le j \le d \wedge j \ne i\},$$

and correspond to the following boundary subsets of P:

$$IN_i = \{m_i\} \times P_i \quad (\textit{input}) \qquad\qquad OUT_i = \{n_i\} \times P_i \quad (\textit{output}).$$

For example, if b_3: $P_3 \rightarrow [0, NumWaves] \rightarrow$**Values** is an input port and $p^* \in P_3$, then $b_3(p^*)s$ is the value received by $p = p^* \oplus \{3 \rightarrow m_3\} \in IN_3$ during wavefront s.

Semantic Rules

For convenience, we will assume that **Indices** = P and **Stages** = $[0, NumWaves]$. The semantics of the read-only variables *self* and *stage* are specified as follows. For every $p \in$ **Indices** and $s \in$ **Stages**,

$(0)_i$ $[\![$ **self**$[i]$ $]\!]ps = p_i \qquad 1 \le i \le d$
$(0)_{ii}$ $[\![$ **stage** $]\!]ps = s.$

The variables **self** and **stage** can be used in conditional statements to express space and time heterogeneity of a cell computation. For example, the statement

COND (*self*[1] < *self*[2]) -> { $c := c + 2$ }, (*self*[1] >= *self*[2]) -> **SKIP**

specifies *space heterogeneity*; when it is executed, the value of c is incremented by 2 in each cell whose first coordinate is less than its second coordinate. In a similar manner,

COND (*stage* = 1) -> { $z := w$}, (*stage* > 1) -> { $z, w := w, z$ }

specifies *time heterogeneity*; the value of w is assigned to z when the first wavefront is processed and the values of z and w are interchanged when the second and succeeding wavefronts are processed.

Based on a specification

CELL(IMPORT $i_1, i_2, ..., i_u$ EXPORT $e_1, e_2, ..., e_v$)

found in the instantiation and a specification

$$\textbf{CELL(IMPORT } g_1, g_2, ..., g_r \textbf{ EXPORT } h_1, h_2, ..., h_s),$$

found in the cell declaration, we assume that $r = u$, $s = v$, and for every $p \in$ Indices,

$(1)_i$ $\quad [\, g_m \,]p\ 0 = i_m(p) \qquad\qquad 1 \le m \le r$

$(1)_{ii}$ $\quad e_n(p) = [\, h_n' \,](p)NumWaves \qquad 1 \le n \le s.$

Equation $(1)_i$ says that when the pure cell computation is initiated, the variable g_m has the values of the external port i_m. Similarly, equation $(1)_{ii}$ says that at the completion of the wavefront computation, the external port e_n has the final values of the variable h_n.

The communication specification

$$?\ \textit{in-list}\ \text{<Seq\_comp>}\ !\ \textit{out-list}$$

declares two array identifiers *in-list* and *out-list* of size d. The references $\{in\text{-}list[i]\}$ and $\{out\text{-}list[i]\}$, $1 \le i \le d$, may be used in <Seq_comp>.

Assume that $exp \in$ **LocalExp** does not use any of the references $\{in\text{-}list[i]\}$. For every $p \in$ **Indices** and $s \in [1, NumWaves]$,

(2) $\quad [\, exp \,]ps = [\, exp' \,]p(s\ -1).$

Equation (2) states that the wavefront computation in each cell is performend in a sequential manner. The initial value of an identifier that does not refer to *in-list* in a cell at a particular wavefront is the final value of the identifier in the same cell with respect to the preceding wavefront.

For each $1 \le i \le d$, $p \in \mathbf{P}^{(i)} = [m_i + 1, n_i] \times \mathbf{P}_i$, and $s \in [1, NumWaves]$,

(3) $\quad [\, in\text{-}list[i] \,]ps = [\, out\text{-}list[i]' \,]p^{(i)}s,$

where $p^{(i)} = p \oplus \{i \rightarrow p_i - 1\}$.

Equation (3) states that the initial value of $in\text{-}list[i]$ in a non-input boundary cell is the final value of $out\text{-}list[i]$ in the adjacent cell in dimension i.

Assume that the corresponding instantiation involves the syntactic construct

$$\text{USING } b, c$$

This construct specifies two lists of boundary ports: $\{b_i\}$ and$\{c_i\}$. Data movement for the boundary cells is then specified as follows: for each $1 \le i \le d$,

$$(4)_{\text{i}} \quad [\, \textit{in-list}[i] \,]p = b_i(p^i) \qquad (p \in \mathbf{IN}_i)$$
$$(4)_{\text{ii}} \quad c_i(q^i) = [\, \textit{out-list}[i]' \,]q \qquad (q \in \mathbf{OUT}_i),$$

where $\{p^i, q^i\} \subseteq \mathbf{P}_i$ are defined by $p = p^i \oplus \{i \rightarrow m_i\}$ and $q = q^i \oplus \{i \rightarrow n_i\}$, respectively.

Equations $(4)_{\text{i-ii}}$ state that the values of *in-list*[i] are read from the input boundary port b_i and the output boundary port c_i reads its values from *out-list*[i].

It is also possible to communicate on only a subset of the grid dimensions. To do this, the syntactic form

$$?\ [X_1, X_2, ..., X_k]\ \ \text{<Seq\_comp>}\ \ !\ [A_1, A_2, ..., A_k]$$

must be used, along with the instantiation construct

USING {(IN_PORT | OUT_PORT) (<Dim>, <Id>)}.

Each IN_PORT specification for a given dimension must be matched by an OUT_PORT specification for the same dimension. Each dimension can be used in at most one pair of port specifications. The number of port specifications must be the same as the length of the input and output lists in the communication specification, and the association of ports with identifiers is positional. In this situation, each cell will receive the values of the variables $\{X_i\}$ and will send the values of the variables $\{A_i\}$ along the specified dimensions. There will be no data movement with respect to dimension other than the ones specified.

For example, if the syntax is

USING IN_PORT(1, b_1) IN_PORT(3, b_3) OUT_PORT(1, c_1) OUT_PORT(3, c_3)

$$?\ [X_1, X_3]\ \ \text{<Seq\_comp>}\ \ !\ [A_1, A_3],$$

then $k = 2$. The corresponding semantic equations for $m \in \{1, 3\}$ are

$$(3) \quad [\, X_m \,]ps = [\, A_m' \,]p^{(m)}s \qquad (p \in \mathbf{P}^{(m)})$$

$$(4) \quad [\, X_m \,]p = b_m(p^m) \qquad (p \in \mathbf{IN}_m)$$
$$c_m(q^m) = [\, A_m' \,]q \qquad (q \in \mathbf{OUT}_m).$$

2. Wavefront Algorithms

The following two examples illustrate the specification of wavefront algorithms.

Example 1: The following wavefront algorithm for Gaussian elimination is originally due to Ahmed, Delosme, and Morf [ADM]. The instantiation portion of the specification is

```
N = 100; North = 1; South = 1; East= 2; = West = 2;
USING  IN_PORT (North, InputData), IN_PORT (West, JunkData),
       OUT_PORT (South, JunkOutput), OUT_PORT (East, OutputData)
EXECUTE NETWORK (0 : N-1, 0 : N-1) OF CELL() FOR NUMWAVES = N
```

The corresponding cell declaration is

```
CELL() =
   CONSTANT  xself := self[1]; yself := self[2];
   VARIABLE  r : REAL;
   {  r := 0.0                                                          -- 1
      ? [top, left]
      COND  (xself ≥ yself)      -> { right, bottom := top, left},      -- 2
            (stage < xself + 1)  -> { right, bottom := left, top},      -- 3
            (stage = xself + 1)  -> { r := top/left;                    -- 4
                                      bottom, right := r, left},        -- 5
            (stage > xself + 1)  -> { bottom := top - r * left;         -- 6
                                      right := left }                   -- 7
      ! [bottom, right] }
```

The specification has been annotated with line numbers that will be used in the verification presented below in Section 2.

Example 2: The following wavefront algorithm for an IIR filter repeatedly computes the values

$$y_n = \Sigma\{a_k x_{n-k} : 0 \le k \le n\} - \Sigma\{b_k y_{n-k} : 1 \le k \le n\},\ 0 \le n \le N,$$

based on the constants $\{a_k : 0 \le k \le N\}$ and $\{b_k : 1 \le k \le N\}$ and the wavefront input data $\{x_n : 0 \le n \le N\}$. The algorithm consists of $2K$ stages using an $(N + 1) \times (N + 1)$ network of cells. The file *ZeroData* consists of all zeros, the West file *InputData* will consist of alternate rows of data $[x_0\ x_1 \ldots x_N]$ and zeros $[0\ 0 \ldots 0]$, and the East file *OutputData* will consist of alternate rows of the input data $[x_0\ x_1 \ldots x_N]$ and the output data $[y_0\ y_1 \ldots y_N]$.

The instantiation portion of the IIR algorithm specification is

```
N = 100; North = 1; South = 1; East = 2; West = 2;
USING IN_PORT (North, ZeroData), IN_PORT (West, InputData),
      OUT_PORT (East, OutputData), OUT_PORT (South, JunkOutput)
EXECUTE NETWORK(0 : N, 0 : N) OF CELL(IMPORT AData, BData)
FOR NUMWAVES = 2*K
```

The file *AData* satisfies the following property: $AData[i, j] =$ **if** $(j \geq i)$ **then** a_{j-i} **else** 0. *BData* satisfies the analogous property with the additional assumption that $b_0 = 0$.

The corresponding cell declaration is

```
CELL (IMPORT a, b) =
    CONSTANT  xself := self[1]; yself := self[2];
    VARIABLE  y : REAL;
    {   ? [top, left]
        right, bottom := left, top;
        COND
            (xself > yself)        ->  SKIP,
            (stage MOD 2 = 1)  ->  {  y := a*left },
            (stage MOD 2 = 0)  ->  {  y := y + top - b*left;
                                      bottom := y;
                                      COND  (xself = yself)  -> { right := y },
                                                TRUE         -> SKIP  }
        ! [bottom, right] }
```

3. Verification of Wavefront Algorithms

Since the semantic equations reflect the generalized wavefront model, they can be used to prove the correctness of algorithms. The procedure for verifying an algorithm generally consists of an induction proof that uses these equations along with additional equations derived from the standard axioms for sequential computation. This technique will be illustrated by verifying the Gaussian elimination algorithm specified in Example 1.

Define $\mathfrak{M}_N$ to be the subset of nonsingular real $N \times N$ matrices for which Gaussian elimination without pivoting can be performed. For $1 \leq i \leq N$, define $E_i : \mathfrak{M}_N \to \mathfrak{M}_N$ as follows:

$$(3.1) \qquad (E_iA)_{i'j'} = \begin{cases} a_{i'j'} & (i' \leq i) \vee (j' < i) \\ a_{i'j'} / a_{ii} & (i' > i) \wedge (j' = i) \\ a_{i'j'} - (a_{i'i}/a_{ii}) * a_{ij'} & (i' > i) \wedge (j' > i) \end{cases}$$

(3.2) For each matrix $A \in \mathfrak{M}_N$, define $A^{(0)} = A$ and $A^{(m)} = E_m A^{(m-1)}$ $(1 \leq m \leq N\text{-}1)$.

The North input to the grid consists of the columns of matrix A, and the following equations describe the input on the boundaries:

(3.3) InputData(i)(j) = A[i+1,j] $\qquad 0 \leq i \leq N\text{-}1,\ 1 \leq j \leq N$

(3.4) JunkData(i)(j) = 0.0 $\qquad 0 \leq i \leq N\text{-}1,\ 1 \leq j \leq N$

Based on **(4$_i$)**, OutputData(i)(k) = [right'] (i,N-1)(k). The following result therefore proves that the columns of the matrix $A^{(N-1)}$ will be output on the East boundary of the grid. This establishes the correctness of the Gaussian elimination algorithm.

Theorem: For $0 \le i, j \le N\text{-}1$ and $1 \le k \le N$, the following equations hold:

$$[\text{right'}]\ (i,j)(k) = \begin{cases} \text{JunkData}\ (i\text{-}j\text{-}1)(k) & j < i \\ A^{(i)}_{i+1,k} & j \ge i \end{cases}$$

$$[\text{bottom'}]\ (i,j)(k) = \begin{cases} \text{JunkData}\ (i\text{-}j)(k) & j \le i \\ A^{(i+1)}_{j+1,k} & j > i \end{cases}$$

$$[\text{r'}]\ (i,j)(k) = \begin{cases} 0.0 & (i \ge j) \vee (k \le i) \\ A^{(i+1)}_{j+1,i+1} & (i < j) \wedge (k \ge i+1) \end{cases}$$

Proof: The proof uses induction on the sum of the row and column indices i and j, and on the wavefront number. Each step of the proof is supported by annotation; the annotations {Ax}, {Sy}, or {z} refer to line x of the algorithm, to semantic equation y, and to equation z in this section, respectively. The result is first established for the cases $i = 0$ and $j = 0$. It follows immediately from the semantic equations that the desired conclusions hold for the cells in the left column of the grid. Induction on j is used to verify the conclusions for the cells in the top row, as follows.

(a)	[right'] (0,0)(k)	=	[top] (0,0)(k)	{A2}
		=	InputData(0)(k)	{S4$_i$}
		=	A[1,k]	{3.3}
		=	$A^{(0)}_{1,k}$	{3.2}
(j > 0)	[right'] (0,j)(k)	=	[left] (0,j)(k)	{A5, A7}
		=	[right'] (0,j-1)(k)	{S3}
		=	$A^{(0)}_{1,k}$	(induction on j)
(b)	[r'] (0,0)(k)	=	0.0	{A1}

$(j > 0)$ $[r'](0,j)(1) = [top]\ (0,j)(1) / [left]\ (0,j)(1)$ {A4}

$= InputData(j)(1) / [right']\ (0,j-1)(1)$ {$S4_i$,S3}

$= A[j+1,1]/A[1,1]$ {3.3}; (part (a) of proof)

$= A^{(1)}_{j+1,1}$ {3.2}

Note: $S4_i$ implies that $[top]\ p\ (k) = b_1(p^1)(k) = InputData(p^1)(k)$.

$(k > 1)$ $[r']\ (0,j)(k) = [r]\ (0,j)(k)$ {A6, A7}

$= [r']\ (0,j)(k-1)$ {S2}

$= A^{(1)}_{j+1,1}$ (induction on wavefront number)

(c) $[bottom']\ (0,0)(k) = [left]\ (0,0)(k)$ {A2}

$= JunkData(0)(k)$ {$S4_i$}

If $j > 0$, there are two cases to consider:

$(k = 1)$ $[bottom']\ (0,j)(1) = [top]\ (0,j)(1) / [left]\ (0,j)(1)$ {A4, A5}

$= A^{(1)}_{j+1,1}$ (as in part (b))

$(k \geq 2)$ $[bottom']\ (0,j)(k) = [top]\ (0,j)(k) - ([r]\ (0,j)(k) * [left]\ (0,j)(k))$ {A6}

$= [top]\ (0,j)(k) - (A^{(1)}_{j+1,1} * [left]\ (0,j)(k))$ (part (b) of proof)

$= InputData(j)(k) - (A^{(1)}_{j+1,1} * [right']\ (0,j-1)(k))$ {$S4_i$,S3}

$= A[j+1,k] - (A^{(1)}_{j+1,1} * A^{(0)}_{1,k})$ {3.3}; (part(a) of proof)

$= A^{(1)}_{j+1,k}$ {3.2}

The argument in the general case ($i > 0, j > 0$) proceeds differently for *r*, *right*, and *bottom*. For *r*, it is clear that if $j \leq i$ or $k \leq i$, then $[r']\ (i,j)(k) = 0.0$, while if $k > i+1$, $[r']\ (i,j)(k) = [r']\ (i,j)(k-1)$. If $j > i$ and $k = i+1$, we have

$[r']\ (i,j)(i+1) = [top]\ (i,j)(i+1) / [left]\ (i,j)(i+1)$ {A4}

$= [bottom']\ (i-1,j)(i+1) / [right']\ (i,j-1)(i+1)$ {S3}

$= A^{(i)}_{j+1,i+1} / A^{(i)}_{i+1,i+1}$ (induction on $i+j$)

$= A^{(i+1)}_{j+1,i+1}$ {3.2}

For *right*, there are three distinct cases:

($j < i$)	⟦right'⟧ (i,j)(k)	=	⟦top⟧ (i,j)(k)	{A2}
		=	⟦bottom'⟧ (i-1,j)(k)	{S3}
		=	JunkData(i-j-1)(k)	(induction on $i+j$)
($j = i$)	⟦right'⟧ (i,i)(k)	=	⟦top⟧ (i,i)(k)	{A2}
		=	⟦bottom'⟧ (i-1,i)(k)	{S3}
		=	$A^{(i)}_{i+1,k}$	(induction on $i+j$)
($j > i$)	⟦right'⟧ (i,j)(k)	=	⟦left⟧ (i,j)(k)	{A3, A5, A7}
		=	⟦right'⟧ (i,j-1)(k)	{S3}
		=	$A^{(i)}_{i+1,k}$	(induction on $i+j$)

For *bottom*, there are two distinct cases:

($j \leq i$)	⟦bottom'⟧ (i,j)(k)	=	⟦left⟧ (i,j)(k)	{A2}
		=	⟦right'⟧ (i,j-1)(k)	{S3}
		=	JunkData(i-j)(k),	(induction on $i+j$)

while

	⟦bottom'⟧ (i,0)(k)	=	⟦left⟧ (i,0)(k)	{A2}
		=	JunkData(i)(k)	{$S4_i$}

($j > i$) In this situation, there are three subcases, depending on the wavefront number:

($k < i+1$)	⟦bottom'⟧ (i,j)(k)	=	⟦top⟧(i,j)(k)	{A3}
		=	⟦bottom'⟧ (i-1,j)(k)	{S3}
		=	$A^{(i)}_{j+1,k}$	(induction on $i+j$)
		=	$A^{(i+1)}_{j+1,k}$	(by {3.1, 3.2}, since $k < i+1$)
($k = i+1$)	⟦bottom'⟧ (i,j)(k)	=	⟦top⟧ (i,j)(k) / ⟦left⟧ (i,j)(k)	{A4, A5}
		=	⟦bottom'⟧ (i-1,j)(k) / ⟦right'⟧ (i,j-1)(k)	{S3}
		=	$A^{(i)}_{j+1,k}$ / $A^{(i)}_{i+1,k}$	(induction on $i+j$)
		=	$A^{(i+1)}_{j+1,i+1}$	{3.1}

$$(k > i+1) \quad [\![\text{bottom}']\!](i,j)(k) = [\![\text{top}]\!](i,j)(k) - ([\![\text{r}]\!](i,j)(k) * [\![\text{left}]\!](i,j)(k)) \quad \{A6\}$$

$$= [\![\text{bottom}']\!](i-1,j)(k) - ([\![\text{r}]\!](i,j)(k) * [\![\text{right}']\!](i,j-1)(k)) \quad \{S3\}$$

By {S2}, $[\![\text{r}]\!](i,j)(k) = [\![\text{r}']\!](i,j)(k-1)$, which in turn equals $A^{(i+1)}_{j+1,i+1}$, by induction on the wavefront number. We conclude that

$$[\![\text{bottom}']\!](i,j)(k) = A^{(i)}_{j+1,k} - (A^{(i+1)}_{j+1,i+1} * A^{(i)}_{i+1,k}) \quad \text{(induction on } i+j\text{)}$$

$$= (E_{i+1}A^{(i)})_{j+1,k} \quad \{3.1\}$$

$$= A^{(i+1)}_{j+1,k} \quad \{3.2\}$$

4. Conclusions

Our approach to wavefront computing has several definite advantages. First, the generalized wavefront model provides a context for the development of wavefront algorithms that allows the designer of algorithms to ignore potential deadlock problems and other difficult issues involving the coordination of inter-processor communication and external input and output operations. The model's implicit parallelism allows an algorithm to be described by specifying the behavior of individual cells. Second, the language provides a clear and concise means of specifying a wide variety of wavefront algorithms. Finally, the semantic equations permit the verification of wavefront algorithms.

Although *n*-dimensional grids support the specification of many wavefront algorithms, exploration of the potential of the wavefront paradigm requires its extension to other regular interconnection networks. In particular, this extension involves the formulation of general notions of directional movement and boundary in an interconnection network. Research along these lines is currently underway.

References

[A] M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu, and J. A. Webb, The Warp computer: architecture, implementation, and performance, *IEEE Transactions on Computers*, C-36 (1987), 1523-1538.

[ADM] H. M. Ahmed, J. M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing, *Computer* **15** (1982), 65-82.

[BL] M. Barnett and C. Lengauer, "A systolizing compilation scheme", Department of Computer Sciences, University of Texas, Technical Report TR-91-03, 1991.

[CCL] M. Chen, Y.-I. Choo, and J. Li, "Compiling parallel programs by optimizing performance", *Journal of Supercomputing* 2 (1988), 171-207.

[EC] B. R. Engstrom and P. R. Cappello, "The SDEF systolic programming system", *Journal of Parallel and Distributed Computing* **18** (1989), 201-231.

[He] M. Hennessy, "Proving systolic systems correct", *ACM Transactions on Programming Languages and Systems* **8** (1986), 344-387.

[HL] C.-H. Huang and C. Lengauer, "The Derivation of Systolic Implementations of Programs", *Acta Informatica* **24** (1987), 595-632.

[K1] H. T. Kung, "Why systolic architectures", *IEEE Computer* **15** (1982), 37-46.

[K2] S. Y. Kung, *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[K3] S. Y. Kung, K.S. Arun, Ron J Gal-ezer, and D. V. B. Rao, "Wavefront array processor: language, architecture, and applications", *IEEE Transactions on Computers* **C-31** (1982), 1054-1065.

[Le] C. Lengauer, "Code generation for a systolic computer", *Software - Practice and Experience* **20** (1990), 262-282.

[Li] B. Lisper, *Synthesizing Synchronous Systems by Static Scheduling in Space-Time*, Lecture Notes in Computer Science **362**, Springer-Verlag, Berlin, 1989.

[P] S. Purushothaman, "Reasoning about modular systolic algorithms", *Proceedings of the 1987 International Conference on Parallel Processing* , pp. 841-843.

[Q] P. Quinton, "The systematic design of systolic arrays", in *Automata Networks in Computer Science - Theory and Applications*, eds. F. F. Soulié, Y. Robert, M. Tchuente, Manchester University Press, 1987, pp. 229-260.

[RS] M.D. Rice and S.B. Seidman, "A multiprocessor testbed for generalized systolic computation", *Proceedings of Transputing '91*, IOS Press, Amsterdam, 1991, vol. 1, pp. 281-295.

[S1] L. Snyder, "A dialect of the Poker programming environment specialized for systolic computation", *Proceedings of the International Workshop on Systolic Arrays*, Oxford, July, 1986.

[S2] L. Snyder, "The XYZ abstraction levels of Poker-like languages", in *Languages and Compilers for Parallel Computing*, Research Monographs in Parallel and Distributed Computing, MIT Press,Cambridge, MA, 1989, pp. 470-489.

Appendix: Syntax for Wavefront Language

<Specification> ::= <Instantiation> <Cell_decl>

<Instantiation> ::= <GConst_defn>
USING <Boundary_ports>
EXECUTE NETWORK(<Coordinate_list>)
OF CELL (<External_ports>)
FOR NUMWAVES = <GExp>

<Cell_decl> ::= CELL (<Parameter_list>) =
<Local_defns_and_decls>
{ [<Cell_comp>] [<Systolic_comp>] }

<Boundary_ports> ::= <Id>, <Id> | {(IN_PORT | OUT_PORT) (<Dim>,<Id>)}
<External_ports> ::= [IMPORT <Id_list>] [EXPORT <Id_list>]

<Coordinate_list> ::= <Pair_list> | <Pair> : <GExp>
<Pair> ::= <GExp> : <GExp>

<Parameter_list> ::= [IMPORT <Id_list>] [EXPORT <Id_list>]
<Local_defns_and_decls> ::= {<LConst_defn>} {<Var_decl>}

<Systolic_comp> ::= ? <Data_list> <Seq_comp> ! <Data_list>
<Data_list> ::= <Id> | [<Id_list>]

<Seq_comp> ::= <Cell_comp> (except that <While_cmd> cannot be used)
<Cell_comp> ::= <Cmd> | {<Cell_comp>} | <Cell_comp> ; <Cell_comp>

<Cmd> ::= <Assign_cmd> | <Cond_cmd> | <While_cmd> | SKIP
<Assign_cmd> ::= <Id_list> := <LExp_list>
<Cond_cmd> ::= COND <LExp> -> {<Cell_comp>} {, <LExp> -> {<Cell_comp>}}
<While_cmd> ::= WHILE <LExp> -> <Cell_comp>

<LConst_defn> ::= CONSTANT { <Id> = <Exp> ; }
<Var_decl> ::= VARIABLE { <Id_list> : (INTEGER | REAL) ; }

<LExp> ::= TRUE | expressions based on <Num>, the identifiers {self[]} and stage, and the identifiers defined in <LConst_defn> or declared in <Var_decl> or <Parameter_list>

<Exp> ::= expressions based on <Num>, the identifiers {self[]}, and the identifiers previously defined in <LConst_defn>

<GConst_defn> ::= {<Id> = <Num>;}
<Dim> ::= <GExp>
<GExp> ::= arithmetic expressions based on <Num> and identifiers in <GConst_defn>
<Num> ::= integer or real literal