Distributed Computing (1986) 1:177–183

DISTRIBUTED COMPUTING

# Systolic algorithms as programs

**K. Mani Chandy and J. Misra**
Department of Computer Sciences, University of Texas, Austin, TX 78712, USA

**Abstract.** We represent a systolic algorithm by a program consisting of one multiple assignment statement that captures its operation and data flow. We use invariants to develop such programs systematically. We present two examples, matrix multiplication and LU-decomposition of a matrix.

**Key words:** Systolic algorithm – Multiple assignment – Invariant – Program development – Proofs of programs

## 1 Introduction

Systolic algorithms [1] are synchronous parallel programs executing on a number of nodes (machines) interconnected by a set of lines. Systolic algorithms are often described by pictures of nodes and lines, descriptions of processing at each node in the picture and data movement between nodes. A pictorial representation of an algorithm suggests that it can be implemented on a VLSI chip; however, pictures do not lend themselves readily to proofs of correctness.

We view systolic algorithms as programs and apply traditional program development techniques, based on invariants, in their design. In this paper we carry out the development of algorithms for matrix multiplication of band matrices and L-U decomposition of a band matrices. Both algorithms are from Kung and Leiserson [1].

We are far from proposing a VLSI design methodology: we do not consider many of the limitations in a physical realization; these are concerns for a later stage in the design. However, our use of traditional program development techniques seems to yield designs for which data flow rates, initial and boundary conditions – the tedious details – are derived mechanically.

A great deal of work has been done on systematic methods for developing systolic algorithms [5–8]. These methods are largely based on transforming sets of equations into forms suitable for implementation on systolic hardware. The primary contribution of this paper is to represent systolic algorithms by *programs derived from invariants.* Each program consists of *one* multiple assignment statement. Our goal is to apply traditional programming techniques in developing systolic algorithms.

This work was supported in part by a grant from the Office of Naval Research under grant N00014-85-K-0057

*Offprint requests to:* K.M. Chandy
For photographs and biographics see Distributed Computing (1986) 1:40–52

## 2 Programs and systolic algorithms

### 2.1 Programs

Our programs have *multiple assignment* statements. A multiple assignment statement of the form,

$$x, y := f(x, y), g(x, y)$$

assigns $f(x', y')$ and $g(x', y')$ to $x$, $y$ respectively where $x'$, $y'$ are the values of $x$, $y$ prior to the execution of the statement. We allow the right sides of assignments to be conditional expressions. For instance, we represent

$$x := \begin{cases} 0, & \text{if } a > 0 \\ 1, & \text{if } a \leq 0 \end{cases}$$

by,

$$x := 0 \text{ if } a > 0 \sim 1 \text{ if } a \leq 0$$

A program consists of *declarations* of its variables and their initial values and *one* multiple assignment statement. The program execution consists of executing this statement repeatedly forever. Non-terminating execution is convenient for reasoning; however, the program may be stopped when the left and right sides of the statement are equal in value, because no further change in variable values is then possible. Restricting a program to one multiple assignment may seem too restrictive. However, our experience suggests that such programs are adequate for representing systolic algorithms. A multiple assignment can be thought of as a synchronous computation – computing all expressions on the right side synchronously – and hence, captures the essence of systolic computations. Elsewhere [2–4], we have shown that a *set* of multiple assignment statements executed in a non-deterministic fashion represents different kinds of parallel and distributed computations; for this paper we do not require this generality.

## *2.2 Systolic algorithms*

A systolic algorithm is executed on a collection of nodes and directed lines connecting pairs of nodes. A *step* of the computation consists of some nodes (1) reading values from (some or all of) their input lines, (2) computing and (3) writing values to (some or all of) their output lines. A value written to a line is available at the *next step* at the node to which the line is directed. We may represent local data at a node by placing the data on lines directed from the node to itself.

Systolic algorithms display regular structures: there are only a few kinds of nodes, and interconnections among nodes are regular. Furthermore, in many cases, systolic hardware operates in a pipelined fashion.

## *2.3 Representing systolic algorithms by programs*

We represent each line in a systolic circuit by a variable; a variable value at any point in the computation is the value on the corresponding line. Each node in a systolic circuit is represented by an assignment (in the multiple assignment statement). A synchronous step in the systolic algorithm is simulated by executing a multiple assignment statement: it assigns new values to certain variables based on current values of some variables. A small example is given below.

**Example (Shift register).** A systolic algorithm for a shift register with $N$ nodes is shown pictorially

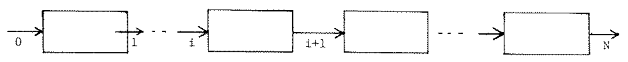

**Fig. 1.** Shift register

in Fig. 1. Every node transmits the value from its input line to its output line in every step. Lines are numbered as shown in the picture.

Let $x[i]$ be the variable associated with the ith line. The multiple assignment statement which represents the operation of this algorithm is, (informally)

for all $i$ in 0 to $N-1$: : {assign in parallel}
$x[i+1] := x[i]$.

We will write this as (in a notation to be introduced later):

```
⟨$i$ in 0..$N-1$::
  || $x[i+1] := x[i]$
⟩
```

Note that there is no explicit mention in the program about data movement. Data items move within the array by being assigned to different array elements, but our treatment *does not* trace the movement of individual data items.

A multiple assignment statement may represent an algorithm having no systolic realization. For instance, a line value is read by exactly one node in a systolic algorithm but a variable may appear in the right hand side of more than one assignment in our program. Similarly, computation at a node usually depends on only a few input line values due to physical constraints, but our programs allow expressions on the right hand side to have arbitrary numbers of variables. We constrain our programs to mirror these limitations of systolic hardware.

*Limited fan-in, fan-out:* Each expression on the right hand side of an assignment has a bounded number of variables. This bound is the maximum fan-in. Each variable appears at most once on the left hand side of an assignment and at most once on the right hand side of an assignment; this is because each line is directed from one node to one other node.

Systolic algorithms typically operate on arrays of data items. Systolic algorithms require that the speed at which data moves through the circuit be independent of the index of the data items (usually). Hence, we propose:

*Linearity:* The step number at which a computation is done is usually a simple (e.g., linear or piecewise-linear) function of data indices.

We have shown the correspondence between systolic algorithms and a special kind of program. Henceforth, we deal only with issues of developing such a program from a specification.

## 2.4 Program development

As in other areas of programming, an *invariant* is a central concept in our approach to systolic algorithms. In fact, it seems that the program design task is almost over once a suitable invariant is found. We introduce a variable $t$, denoting the step number ($t$ is initially 0 and is increased by 1 in each execution of the statement) and state an invariant relating various data items and $t$. We will be guided by the limited fan-in-fan-out and linearity requirements in postulating an invariant. The effect of statement execution is to preserve the invariant when $t$ is increased by 1.

The invariant is useful in deriving initial conditions and boundary conditions. Determining these conditions and the rate of data flow are the most tedious details one has to contend with; invariants seem to simplify the effort.

## 2.5 Notation

We use || to break up a multiple assignment statement into its component assignments for convenience in reading. For instance,

$x, y := y, x$

is equivalent to

$x := y \,||\, y := x.$

The following notation, where $S$ is a set and each $Q(i)$ is an assignment (or multiple assignment):

$\langle i \text{ in } S :: || Q(i)\rangle$

denotes a statement obtained by enumerating, for every element of $S$, $Q(i)$ with $i$ replaced by that element. For example $\langle i \text{ in } 0..1 : || X[i] := Y[i]\rangle$ is equivalent to $|| X[0] := Y[0] \,||\, X[1] := Y[1]$. We omit $S$ when it is clear from the context. The statement,

$x := e \text{ if } b$

is to be interpreted as

$x := e \text{ if } b \sim x \text{ if } \neg b.$

The scope of *if* will be shown explicitly, if needed, as in the following.

$x, y := (e_1, e_2) \text{ if } b$

is equivalent to,

$x, y := e_1 \text{ if } b,\ e_2 \text{ if } b$

and also equivalent to,

$(x, y := e_1, e_2) \text{ if } b$

## 3 Band matrix multiplication

The problem is to compute

$C = A \cdot B$

where $A$, $B$ are band matrices and "$\cdot$" denotes multiplication.

We have,

$C[i, k] = \sum_j A[i, j] \times B[j, k]$

This expression cannot be used directly for computing $C[i, k]$ since that would violate the limited fan-in-fan-out requirement. Therefore, we define as in [1]:

$$C^j[i, k] = \begin{cases} 0, & \text{if } j < 0 \\ C^{j-1}[i, k] + A[i, j] \times B[j, k], & \text{if } j \geq 0 \end{cases} \qquad (1)$$

Equation (1) suggests that $A[i, j]$ and $B[j, k]$ will be multiplied in some step. Using the linearity criterion, we may postulate that they will be multiplied in a step which is a linear function of $i, j, k$. If this linear function is independent of one of its arguments, say $i$, then for any fixed value of $j, k$, $A[i, j]$ and $B[j, k]$, will be multiplied in the same step for all $i$; that is $B[j, k]$ will appear in more than one computation in a step, thus violating the limited fan-in-fan-out requirement. Hence, we may assume that $A[i, j]$, $B[j, k]$ are multiplied in a step that is a nontrivial linear function of each of its arguments – we choose the simplest such function; $i+j+k$.

Since $A$, $B$ are band matrices, we postulate that each diagonal (main, subdiagonal or superdiagonal) is pipelined. Let one node be assigned for each pair of diagonals – one from $A$ and one from $B$ – to carry out computations on element pairs from these diagonals. Element $A[i, j]$ belongs to diagonal $(i-j)$ of $A$ and $B[j, k]$ to diagonal $(j-k)$ of $B$; hence index the node at which they are multiplied by $(i-j, j-k)$.

Equation (1) suggests that $A[i, j]$, $B[j, k]$, $C^{j-1}[i, k]$ be made available at the same time at some node and, from this discussion, that node is $(i-j, j-k)$. Therefore, each node $(v, w)$ has three input lines $X[v, w]$, $Y[v, w]$ and $Z[v, w]$, along which $A$, $B$, $C$ respectively are pumped into it. From this discussion, we have the following invariant.

**Invariant:**

$t=i+j+k \Rightarrow$
$[X[i-j, j-k]=A[i,j]$ and,
$Y[i-j, j-k]=B[j,k]$ and
$Z[i-j, j-k]=C^{j-1}[i,k]$.
]

The variables $i, j, k, t$ in the invariant are universally quantified over all integers; ignore the equations corresponding to undefined subscript values in the right side.

Our design task is nearly complete! We merely have to show how to establish the invariant initially, and how to preserve it when $t$ is increased by 1.

## *3.1 Initial Conditions*

Initially, let $t$ be 0. Then for any $i, j$ with $k=-(i+j)$, we are required to have,

$X[i-j, i+2j]=A[i,j]$.

Similarly,

$Y[-2j-k, j-k]=B[j,k]$.

Let $j=-(i+k)$, where $i \geq 0$, $k \geq 0$. Then, $j \leq 0$. Hence,

$C^{j-1}[i,k]=0$.

Substituting $-(i+k)$ for $j$ in the invariant,

$Z[2i+k, -i-2k]=0$.

Summarizing the initial conditions,

$X[i-j, i+2j]=A[i,j]$, for all $i, j$
$Y[-2j-k, j-k]=B[j, \mathrm{k}]$, for all $j, k$
$Z[2i+k, -i-2k]=0$, for $i \geq 0$, $k \geq 0$.
$t=0$

## *3.2 Preserving the invariant*

We show how to preserve the invariant when $t$ is increased by 1. First, we simplify the notation by introducing,

$v=i-j$ and $w=j-k$.

First consider the data item $A[i,j]$. From the invariant, it equals $X[v, w]$ at $t=i+j+k$. It must equal $X[v, w-1]$ after $t$ is increased by 1. This can be accomplished by the assignment,

$X[v, w-1] := X[v, w]$.

Similarly, we get the assignments,

$Y[v+1, w] := Y[v, w]$ and,
$Z[v-1, w+1] := Z[v, w]+X[v, w] \times Y[v, w]$.

Note that these steps need be carried out only for $t, i, j, k$ satisfying $t=i+j+k$, i.e., $t=(i-j)-(j-k)+3j$. We rewrite this condition – weakening it somewhat, to eliminate $i, j, k$ – as $t=(v-w) \bmod 3$. This results in the following program.

```
Program P1 {for multiplying band matrices}

 initially :

    ⟨ for all i,j :: X[i − j, i + 2j]      =  A[i,j] ⟩
    ⟨ for all j,k :: Y[ − 2j − k, j − k]   =  B[j,k] ⟩
    ⟨ for all i,k :: Z[2i + k, − i − 2k]   =      0 ⟩

 assign    : ⟨ for all v, w::
              ( ||    X[v,w − 1]  :=  X[v,w]
                ||     Y[v + 1,w] :=  Y[v,w]
                || Z[v − 1,w + 1] :=  Z[v,w] + X[v,w]×Y[v,w]  )
                                                          if t = (v − w) mod 3 ⟩
                ||              t :=  t + 1
end {P1}
```

This program represents a systolic array. We have finished a large part of the design. What remains to be done is to determine the size of the systolic array and the number of steps required to complete execution.

## *3.3 Determining array size and number of steps*

Program $P1$ does not specify the dimensions of $X$, $Y$, $Z$ nor the step number $t$, up to which program execution should continue. These parameters, and others, can be deduced from the invariant using the sizes of input matrices $A$, $B$ as parameters.

Let $BA$, $TA$ (bottom of $A$, top of $A$) have the following meaning: $A[i,j]$ is zero unless $BA \leq i-j \leq TA$. Likewise, define $BB$, $TB$ for matrix $B$. The multiplication in program $P1$ yields a zero if $X[v, w]=0$ or $Y]v, w]=0$. Therefore, we may restrict $v$, $w$ to the range $BA \leq v \leq TA$ and $BB \leq w \leq TB$ for computation of the product. Hence, $Z$ can be dimensioned $(BA-1 \,..\, TA, BB \,..\, TB+1)$. Other assignments merely move the elements of $A$ or $B$; this corresponds to feeding the systolic array appropriate elements of $A$ and $B$.

Next, we determine when and where $C[i,k]$, for any given $i, k$, becomes available. That is, we want to find $T$ and $v$, $w$ such that,

$(t=T) \Rightarrow (Z[v, w]=C[i,k])$.

First we determine $j$ such that:

$$C[i,k]=C^{j-1}[i,k]. \qquad (2)$$

This holds when $A[i,j]=0$ or $B[j,k]=0$. ($A[i,j]=0$ and $B[j,k]=0$ if $j$ exceeds the number of columns of $A$. To eliminate special case analysis, we assume that $A$, $B$ are augmented with a suitable number of zeros for larger values of $j$.)

$A[i,j]=0$, for $j\geq 0$, if $i-j<BA$,
$B[j,k]=0$, for $j\geq 0$, if $j-k>TB$.

Hence the minimum value of $j$ for which (2) holds is $j^*$ given by

$j^*=min(i-BA, k+TB)+1$

From the invariant, at $T=i+j^*+k$,

$Z[i-j^*, j^*-k]=C[i,k]$.

Note that in case $-BA=TB$, $j^*=min(i,k)+TB+1$. Hence the systolic array has a pleasing diamond structure as given in [1]. However, for arbitrary $BA$, $TB$, the structure is not as regular. We show the relevant portion (i.e., where multiplication is done) of an arbitrary systolic array in Fig. 2.

The invariant simplified the considerations of initial and boundary conditions and data flow rates. In this particular example, we imagined that all the elements of matrices $A$, $B$ are initially placed on certain lines, though the useful work (of multiplication) is performed in a limited region. Now we consider an example, L-U decomposition, where such an assumption cannot be made; in fact, the goal of the algorithm is to compute something akin to $A, B$ from $C$.

## 4 L-U Decomposition of a band matrix

Given a band matrix $A$, its L-U decomposition is a pair of matrices $L$ and $U$ where $L$ is a lower diagonal matrix with 1's on diagonals and $U$ is an upper diagonal matrix, satisfying the following equations. (These equations are from [1] with indices renamed and renumbered starting from 0.)

$$A^0[i,k]=A[i,k]$$
$$A^{j+1}[i,k]=A^j[i,k]-L[i,j]\times U[j,k], j\geq 0$$

$$L[i,j]=\begin{cases}0, & \text{if } i<j\\ 1, & \text{if } i=j\\ A^j[i,j]/U[j,j], & \text{if } i>j\end{cases}$$

$$U[j,k]=\begin{cases}0, & \text{if } j>k\\ A^j[j,k], & \text{if } j\leq k\end{cases}$$

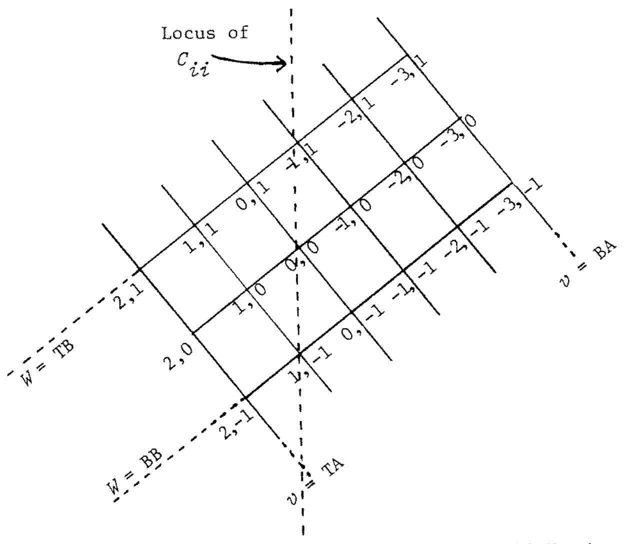

**Fig. 2.** Relevant portion of a systolic array for multiplications of band matrices with $BA=-3$, $TA=2$, $BB=-1$, $TB=1$

We adopt the convention that $A^j[i,k],=A[i,k]$, for $j\geq 0$. As described in the last section, let $BA, TA$ be such that $A[i,j]=0$ unless $BA\geq i-j\geq TA$. It can be shown for band matrices that

$$A^{j+1}[i,k]=\begin{cases}A[i,k], & \text{if } (i-j>TA) \text{ or } (j-k<BA)\\ A^j[i,k]-L[i,j]\times U[j,k], & \text{otherwise}\end{cases}$$

The form of computation on $A$ suggests matrix multiplication. Hence, we attempt using the invariant for matrix multiplication, with variables suitably renamed for this problem. In the following invariant we have constrained certain indices because $L$ is a lower diagonal and $U$ an upper diagonal matrix.

**Invariant:**

$t=i+j+k\Rightarrow$
$[(j\geq 0, i\geq j, k>j\Rightarrow X[i-j, j-k]=L[i,j])$ and
$(j\geq 0, i>j, k\geq j\Rightarrow Y[i-j, j-k]=U[j,k])$ and
$(i\geq 0, k\geq 0, i\geq j, k\geq j\Rightarrow Z[i-j, j-k]=A^j[i,k])$
$]$

As before, we give initial conditions satisfying the invariant and show how to preserve the invariant when $t$ is increased by 1. The major difference from matrix multiplication is that $L$, $U$, unlike in matrix multiplication, are not available initially and have to be computed.

### 4.1 Initial conditions

For $t=0$, the first two conditions in the invariant are vacuously satisfied because there are no $i, j, k$

satisfying these conditions. The last condition, for any $i \geq 0$, $k \geq 0$, can be satisfied by letting $j = -(i + k)$ and hence.

$Z[2i+k, -i-2k] = A^j[i, k] = A[i, k]$ {since $j<0$}

### 4.2 *Preserving the invariant*

As before, we use

$v = i - j$ and $w = j - k$.

It follows from the invariant that we need only consider the cases for $v \geq 0$ and $w \leq 0$.

#### 4.2.1 Preserving the first condition in the invariant

We now consider preservation of,

$t = i+j+k$ and $j \geq 0$, $i \geq j$, $k > j \Rightarrow$
$X[v, w] = L[i, j]$.

For any $i, j, t$ where $i \geq j \geq 0$ and $t > i+2j$: there is some $(v, w)$, $v \geq 0$, $w \leq 0$ such that, $X[v, w] = L[i, j]$.

We now ask ourselves how this requirement is to be met. If $t > i+2j$, $L[i, j]$ has already been computed and has to be assigned to the proper $X[v, w]$. If $t = i+2j$, then $L[i, j]$ is to be computed and assigned to the appropriate $X[v, w]$.

*Case 1.* $t > i+2j$ {equivalently, $w<0$}:

Then, $X[v, w] = L[i, j]$.

The invariant is preserved by:

$X[v, w-1] := X[v, w]$

*Case 2.* $t = i+2j$ {equivalently, $w=0$}:

The invariant is preserved by computing $L[i, j]$ according to its defining equation and then assigning it to the proper variable.

At the following step, i.e., the $(t+1)$th step, the only $k$ satisfying

$t+1 = i+j+k$

is $k = j+1$.

Hence at this step we must have

$X[v, -1] = L[i, j]$

substituting for $L[i, j]$:

$X[v, -1] := \{L[i, j] =\}$
$A^j[i, j]/U[j, j]$ if $i>j \sim 1$ if $i=j$.

From the invariant, at $t = i+2j$,
$A^j[i, j] = Z[i-j, 0]$.

Also, at $t = i+2j\{i>j, k=j\}$, $U[j, j] = Y[i-j, 0]$.

Hence, the following assignment preserves the invariant {$i<j$ is rewritten as $v>0$}:

$X[v, -1] := Z[v, 0]/Y[v, 0]$ if $v>0 \sim 1$ if $v=0$.

#### 4.2.2 Preserving the second condition in the invariant

By similar reasoning we identify two cases.

*Case 1.* $t > 2j+k$ {equivalently $v>0$}:

$Y[v+1, w] := Y[v, w]$

*Case 2.* $t = 2j+k$ {equivalently $v=0$}:

$Y[1, w] := A^j[j, k]$

At $t = 2j+k$, $A^j[j, k] = Z[0, j-k]$. Hence,

$Y[1, w] := Z[0, w]$

#### 4.2.3 Preserving the third condition in the invariant

From the equations,

$$A^{j+1}[i, k] = \begin{cases} A[i, k], \text{ if } i-j > TA \text{ or } j-k < BA \\ A^j[i, k] - L[i, j] \times U[j, k], \text{ otherwise} \end{cases}$$

By similar arguments we derive the following assignment to preserve the invariant.

$Z[v-1, w+1] :=$
$Z[v, w]$ if $v > TA$ or $w < BA \sim$
$Z[v, w] - X[v, w] \times Y[v, w]$ if $v \leq TA$ and $w \geq BA$

**Program** $P2$ {L-U decomposition of a band matrix}

```
  initially : $t = 0$,
              ⟨ for all $i \geq 0, k \geq 0$ : $Z[2i+k, -i-2k] = A[i,k]$ ⟩

  assign    : ⟨ for all $v,w$::

   ( ||     $X[v, w-1]$  :=  $X[v,w]$ if $w < 0 \sim Z[v,0]/Y[v,0]$ if $w = 0$ and $v > 0$
                           $\sim$ 1 if $w = 0$ and $v = 0$

     ||     $Y[v+1, w]$  :=  $Y[v,w]$ if $v > 0 \sim Z[0,w]$ if $v = 0$

     ||  $Z[v-1, w+1]$  :=  $Z[v,w] - X[v,w] \times Y[v,w]$ if $v \leq TA$ and $w \geq BA$
                           $\sim$ $Z[v,w]$ if $v > TA$ or $w < BA$ )

                                                          if $t = (v - w)$ mod 3 ⟩

     ||              $t$  :=  $t + 1$

end {$P2$}
```

## 5 Discussion

Programs $P1$, $P2$ capture the essence of the algorithms given in [1] for the corresponding problems. The multiple assignment statement in each program can be implemented by associating a node with each $(v, w)$ and carrying out the operations in each step as given by the algorithm. The algorithm tells us that node $v, w$ accepts inputs along $X[v, w]$, $Y[v, w]$, $Z[v, w]$ and produces results along $X[v, w-1]$, $Y[v+1, w]$ and $Z[v-1, w+1]$ in each step $t$, where $t=(v-w)$ mod 3. Some ingenious optimizations have been applied in [1] so that only one kind of node – that receives three values $A$, $B$, $C$ and computes $A+B\times C$ – may be used almost completely throughout the systolic array.

This work is part of an ongoing project called UNITY [2–4] to provide a unified framework for the development of sequential, parallel and distributed programs. A thesis of UNITY is that early stages of program design should not be concerned with architectural and programming language issues: these concerns are appropriate only for later stages of design. Another thesis is that diverse applications – ranging from VLSI algorithms to communication protocols, from command and control systems to spread-sheets – are *programs and amenable to a common design strategy*. UNITY has yet to give conclusive proof of these theses – but we are hopeful.

VLSI implementations require considerably more than an algorithmic description. We have only addressed concerns dealing with correctness arguments and systematic program development. It is straight-forward to map our programs to circuits with limited fan-in and where each line is directed from one node to one node. However, not all such circuits can be implemented in VLSI.

## 6 References

1. Kung HT, Leiserson CE (1980) Algorithms for VLSI Processor Arrays (Section 8.3). In: Mead C, Conway L (eds) Introduction to VLSI Systems. Addison-Wesley
2. Chandy KM (1985) Concurrency for the Masses. Invited Address: Third Annual ACM Symposium on Principles of Distributed Computing, August 1984, Vancouver, Canada. Proceedings of Fourth Annual ACM Symposium on Principles of Distributed Computing
3. Chandy KM, Misra J (to be published) An Example of Stepwise Refinement of Distributed Programs: Quiescence Detection. ACM Trans Program Lang Syst
4. Chandy KM, Misra J (1985) Programming and Parallelism: The Proper Perspective. Research Report, Computer Sciences Department, University of Texas, November 1985
5. Leiserson C, Rose F, Saxe J (1983) Optimizing Synchronous Circuitry by Retiming. In: Bryant R Third Caltech Conference on VLSI California Institute of Technology, March 1983, pp 87–116
6. Li GJ, Wah BW (1985) The Design of Optimal Systolic Arrays. IEEE Trans Comput. C-34: 66–77
7. Chen MC (1985) A Parallel Language and its Compilation to Multiprocessor Machines or VLSI. Research Report, Yale University, DCS-RR-432, October 1985
8. Chen MC (1985) The Generation of a Class of Multipliers: A Synthesis Approach to the Design of Highly Parallel Algorithms in VLSI. Research Report, Yale University, DCS-RR-442, December 1985