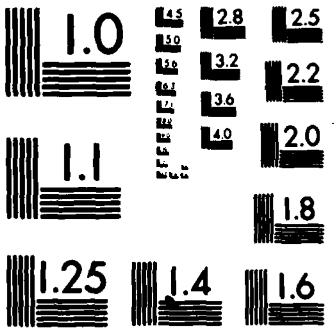
AD-A171 151 IRREGULAR WAVEFRONTS IN DATA-DRIVEN DATA-DEPENDENT COMPUTATIONS(U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS R G MELHEM 1/1

UNCLASSIFIED JUN 86 ICMA-86-94 N00014-85-K-0339 F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

(14)

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS

Technical Report ICMA-86-94 June 1986

IRREGULAR WAVEFRONTS IN DATA-DRIVEN DATA-DEPENDENT COMPUTATIONS

by

Rami G. Melhem

Department of Mathematics and Statistics

University of Pittsburgh

(14)

Technical Report ICMA-86-94 June 1986

IRREGULAR WAVEFRONTS IN DATA-DRIVEN DATA-DEPENDENT COMPUTATIONS

by

Rami G. Melhem

Institute for Computational Mathematics and Applications
Department of Mathematics and Statistics
University of Pittsburgh
Pittsburgh, PA 15260

To be presented in the

International Workshop on Systolic Arrays

University of Oxford

2-4 July 1986

IRREGULAR WAVEFRONTS IN DATA-DRIVEN DATA-DEPENDENT COMPUTATIONS[*]

Rami G. Melhem[**]

ABSTRACT

Data driven networks may be more efficient than clocked networks for computations which require data dependent local cycles. However, the performance of such networks may not be easily predicted because of possible delays in computations due to internal data conflicts. In this paper, a technique, which is based on the irregular propagation of computation fronts, is suggested for the study of the behavior of networks with data dependent operations.

*) This work is supported, in part, under ONR contract N00014-85-K-0339

**) On leave from the department of Computer Science, Purdue University, West Lafayette, IN 47907

IRREGULAR WAVEFRONTS IN DATA-DRIVEN, DATA-DEPENDENT COMPUTATIONS

RAMI G. MELHEM

INTRODUCTION

Both systolic (Kung 1982) and data driven networks (Kung 1984) share the advantages of efficient communications and fast specialized cells which repeat the execution of specific local cycles. However, systolic networks are less flexible than data driven networks in the sense that if the execution time of the local cycles is not constant, then the period of the global synchronization clock should be taken large enough to accommodate the slowest local cycle.

In this paper, we consider networks in which the execution time of local cycles depends on the input data. Typically, this may occur if the local cycles contain branching statements. Although data driven networks are self-synchronized, and hence, local cycles are allowed to have different execution times, it is not obvious that the execution of the entire network may benefit from the fast execution of some local cycles. More specifically, internal data conflict may force a "potentially short" local cycle to wait extensively for its input.

The study of speed and efficiency of data driven networks with data dependent operations is extremely hard due to the asynchronous nature of the networks. Hence, we suggest a technique for the estimation of a lower bound on the performance of such networks. Namely, we introduce a simpler, hypothetical, type of computations, which we call pseudo-systolic. It is obtained by forcing some synchronization on the data driven network such that its execution alternates between communication and processing phases. Clearly, the additional synchronization may only slow down execution, and hence, the analysis of pseudo-systolic computations provide upper bounds on the execution time of the corresponding data driven computations.

The state of a pseudo-systolic computation at any given time may be represented by a computation front. However, unlike the wave front concepts described by Weiser and Davis (1981) and Kung (1984), the progress of the pseudo-systolic computation causes an irregular propagation of the computation front. This irregularity reflects the differences in the execution time of the local cycles.

Computation fronts may be systematically constructed by the application of some conditions which are necessary for the consistency of data flow in pseudo-systolic networks. The constructed fronts may then be applied to the estimation of the execution time of the corresponding computation. This methodology is first illustrated in Section 1 with an example of a linear array with simple cells. Then it is applied in Section 2 to 2-dimensional arrays and in Section 3 to networks with complex cells.

DATA DRIVEN NETWORKS WITH TRIVIAL/NONTRIVIAL LOCAL CYCLES

A data driven network is defined here as a set of cells, each having a certain number of input/output ports, and a set of unidirectional communication links, each connecting an output port of some cell to an input port of another cell.

Each communication link directed from a cell q to a cell k is regarded as a queue Q capable of holding a certain number of data items. It is natural to assume that Q is empty at the beginning of the operation of the network. However, it is sometimes useful to initialize Q such that it contains some data items. In order to be more specific, let QX and QY represent two links directed from cell q to cell k, and assume that, initially, QY is empty while QX contains one data item (a zero for example). Now if, during operation, cell q writes $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ on QX and QY, respectively, then the sequence of items read by cell k from the same queues will be $0, x_1, x_2, \ldots$ and $y_1, y_2, \ldots$, respectively. This skewing effect is equivalent to the one obtained in systolic (clocked) networks by the insertion of a delay element on the x-stream communication line.

An Example

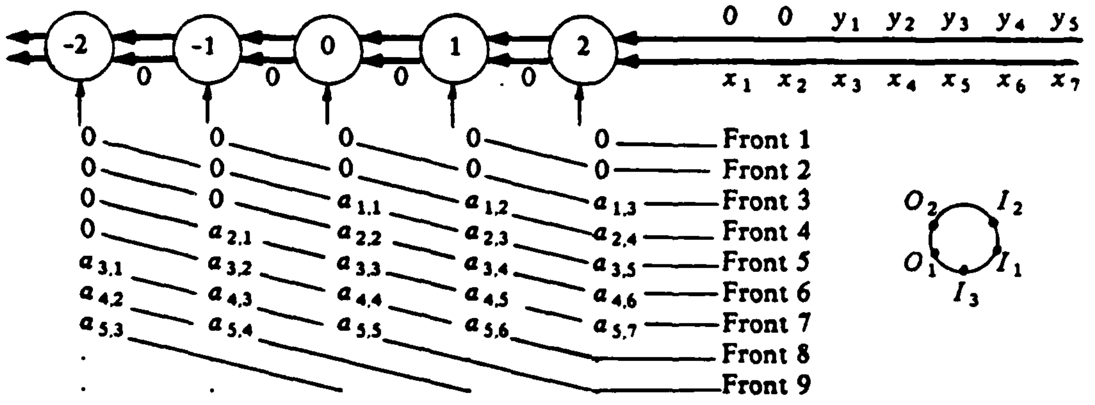

-2
-1
0
1
2
0 0 y_1 y_2 y_3 y_4 y_5
x_1 x_2 x_3 x_4 x_5 x_6 x_7
0
Front 1
Front 2
Front 3
Front 4
Front 5
Front 6
Front 7
Front 8
Front 9
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$a_{4,2}$ $a_{4,3}$ $a_{4,4}$ $a_{4,5}$ $a_{4,6}$
$a_{5,3}$ $a_{5,4}$ $a_{5,5}$ $a_{5,6}$ $a_{5,7}$
O_2 I_2
O_1 I_1
I_3

Figure 1: Matrix/Vector Multiplication (w=2)

The network shown in Figure 1 may be used for the multiplication of an $n \times n$ banded matrix A by a vector x. For simplicity, we assume that the number w of lower diagonals of A is equal to the number of upper diagonals, and hence, the bandwidths of A is $W = 2w + 1$.

The network is composed of W cells, where each two consecutive cells k and $k-1$ are connected by two links, which we call the x-link and the y-link. The queues on the y-links are set to be initially empty and those on the x-links are set to contain a single data item, namely a "zero". Using the naming convention shown in Figure 1 for the I/O ports, and using the notation $O \leftarrow \alpha$ and $\beta \leftarrow [I]$ to indicate that α is written on port O and that the value at port I is read into β, respectively, we may describe the operation of each cell in the network by the following algorithm:

```
ALG1: Repeat Forever
        1) wait until the queues at $I_1$, $I_2$ and $I_3$ are not empty
        2) $\xi \leftarrow [I_1]$; $\eta \leftarrow [I_2]$; $\alpha \leftarrow [I_3]$
        3) $\eta = \eta + \alpha * \xi$
        4) wait until the queues at $O_1$ and $O_2$ are not full
        5) $O_1 \leftarrow \xi$; $O_2 \leftarrow \eta$
```

The five steps in ALG1 constitute a local cycle which is repeated, indefinitely, by each cell in the network. We assume that the computation time (step 3) and the communication time (steps 2 and 5) do not depend on the values of the data items being processed, and we denote these times by τ_m and τ_c, respectively. With this, the execution time of any local cycle which is not delayed by a wait in steps 1 or 4 is given by $\tau_c + \tau_m$. It is clear, however, that the time of a local cycle may be longer than $\tau_c + \tau_m$ if execution is delayed in steps 1 or 4.

DEFINITION: The "basic local cycle time" is the time needed to complete the execution of a local cycle, excluding any delay caused by a wait for new input or a wait for the consumption of old output.

The sequence of inputs to the network is shown in Figure 1 ($y_1, y_2, \ldots$ are set to zero). If each input is made available to the network as soon as it is needed, then it is easy to verify that the elements y_i, $i = 1, \ldots, n$ of the product vector $y = Ax$ are produced at port O_2 of cell $-w$ at time $(2+i+W)(\tau_c+\tau_m)$.

The assumption that the basic local cycle time is constant leads to a mode of operation in which, after few initial cycles, all the local cycles are automatically synchronized. The progress of the computation in this case may be best represented by the propagation of a computation front as shown in Figure 1.

For highly sparse matrices, the above network is clearly inefficient because most of the elements received on ports I_3 of cells $-w, \ldots, w$ are zeroes, thus leading to trivial and unnecessary computations in step 3 of ALG1. In this case, we may improve the performance of the network, and in the same time reduce the amount of its communication with the outside world, by supplying only the non zero element of A along with their positions. More specifically, the input $a_{i,i+k}$ to port I_3 of cell k is omitted if $a_{i,i+k} = 0$, while if $a_{i,i+k} \neq 0$, the value of the index i is supplied on an additional port, which we call I_4. The precise operation of each cell may be described by the following algorithm:

```
ALG2: $CT = -2; \alpha \leftarrow [I_3]; \iota \leftarrow [I_4]$            /* $CT$ is a local counter */
      Repeat Forever
          1) wait until the queues at $I_1$ and $I_2$ are not empty
          2) $\xi \leftarrow [I_1]; \eta \leftarrow [I_2]; CT = CT + 1$
          3) If($\iota = CT$) then     3.1) $\eta = \eta + \alpha * \xi$
                                       3.2) $\alpha \leftarrow [I_3]; \iota \leftarrow [I_4]$
          4) wait until the queues on $O_1$ and $O_2$ are not full
          5) $O_1 \leftarrow \xi; O_2 \leftarrow \eta$
```

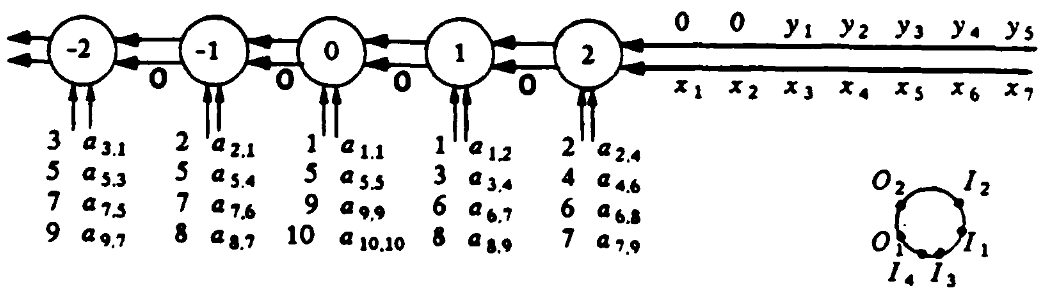


Figure 2: The Multiplication Network for Sparse Matrices (w=2)

The input to the modified network is shown in Figure 2 for a specific sparse matrix A. Any element of A which is not included in the input is assumed to be zero.

Unlike ALG1, the basic local cycle time in ALG2 depends on the input data. More specifically, it may assume one of two values depending on the result of the comparison in step 3. Let the two values be τ_0 and τ_1 for cycles which skip, and do not skip, respectively, step 3.

The architecture of each cell and the technology used to construct the network determine τ_0 and τ_1. However, if we assume that the communication protocols are implemented in hardware, and that the operations in step 3.1 are floating point operations, then it is reasonable to assume that $\tau_0 << \tau_1$. We will call an execution of a cycle which skips steps 3.1 and 3.2 a trivial execution of the cycle. Although trivial executions of local cycles reduce the average basic local cycle time, the efficiency of the entire network is determined by the delay introduced in steps 1 and 4 of local cycles. More specifically, a computational cell executing a non-trivial cycle may hold data which is necessary for the execution of a trivial local cycle in a neighboring cell. In other words, the execution time of the network is determined by internal data conflict.

Given the asynchronous nature of the computation, it seems that simulation is the only method for the precise estimation of its execution time. However, this requires the specification of τ_o and τ_1 which depends on architecture and technology. In the next section, we introduce a method which isolates and measures the effect of internal data conflict on the performance of data driven/data dependent networks.

Pseudo Systolic Synchronization and Irregular Computation Fronts

In order to establish an upper bound on the execution time of networks with trivial/non trivial local cycles, we consider a hypothetical mode of operation which we call "pseudo systolic". It is obtained by adding some global synchronization to the network such that communication and computation take place in two alternating phases. Clearly, the delay introduced by the additional synchronization may only slow down execution, and hence, a study of the pseudo systolic mode of operation may be viewed as a worst case analysis of the asynchronous mode of operation.

More specifically, we assume that all the cells in the network are connected to a "hypothetical" controller which senses the state of execution of each cell. The controller, presumably, forces successive executions of trivial local cycles of all the cells to take place in a single phase which we call a communication phase. This phase is then followed by a processing phase, in which only non trivial executions of local cycles take place. A communication phase followed by a processing phase is called a global cycle.

For example, a pseudo systolic version of the network of Figure 2 may be obtained by replacing step 3 in ALG2 with the following

```
3) IF ($\iota = CT$) THEN 3.1) wait for SYNC
                          3.2) $\eta = \eta + \alpha * \xi$
                          3.3) $\alpha \leftarrow [I_3]; \iota \leftarrow [I_4]$
```

where SYNC is a signal sent by the hypothetical controller at the end of a communication phase. More specifically, during a communication phase, data is moving in the network until each cell is either blocked in steps 1 or 4 due to data conflict or blocked in step 3.1. At this point the controller issues SYNC, and all the cells which are blocked in step 3.1 execute 3.2 and 3.3 simultaneously, while the other cells remain idle. This is a processing phase.

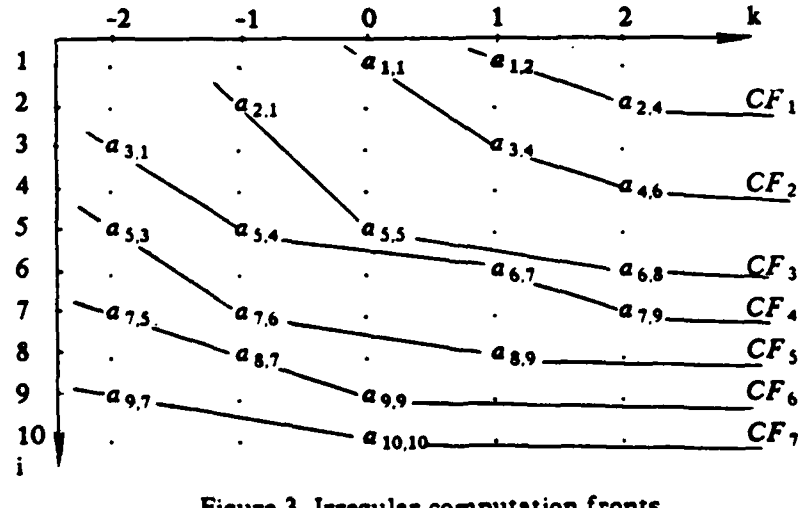


Figure 3. Irregular computation fronts.

Let M_t be the subset of cells which are not idle during the processing phase of the t-th global cycle. Clearly, only the cells in M_t contribute to the advancement of the computation during the t-th global cycle. Hence, the progress of the computations may be represented by the propagation of a front which includes the data items being operated upon by cells in M_t.

More specifically, if $\alpha(t,k) = a_{i,i+k}$, where $a_{i,i+k}$ is the element of A which is at cell k at the beginning of the t-th processing phase, then the t-th computation front may be defined by

$$CF_t = \{\alpha(t,k) ;\ k \in M_t\}$$

For example, we show in Figure 3 the computation fronts for the matrix/vector operation given in Figure 2. The number of fronts, namely 7, indicates the number of global cycles. Although, for small computations, it is possible to construct the fronts by tracing the execution of the network, it is obvious that a more systematic construction method is needed for large computations. This is discussed in the next subsection.

Consistency of Data Flow Conditions

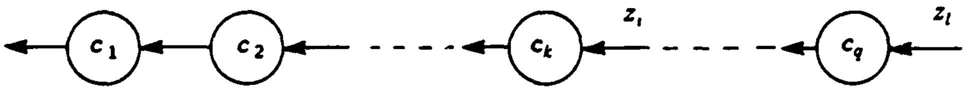


Figure 4. Data flow in a linear array

Let $z_1, z_2, \ldots$ be a sequence of data items which are flowing through a linear array of cells $c_1, c_2, \ldots$ as shown in Figure 4. Let also d_z be the maximum capacity of each queue corresponding to a communication link in the array. Clearly, during any particular instant, the capacity of the queues should not be exceeded, and the order of items in the data stream should be preserved. This may be formally stated as follows:

CONDITION (1): If $q > k$, and at any instant, z_i is at cell c_k and z_l is at cell c_q, then

$$d_z(q-k) \geqslant l-i > 0. \tag{1}$$

CONDITION (2): If z_l and z_i arrive at cell c_k at times t and τ, respectively then

$$l < i \implies t < \tau. \tag{2}$$

The above two conditions, which are necessary and sufficient for the consistency of data streams in linear arrays, may be used to derive the relations between the elements of computation fronts in pseudo systolic computations. For example, consider the multiplication network described in the last section, and let $a_{i,i+k}$ and $a_{l,l+q}$, be two elements in a specific front CF_t. This means that both cells k and q are active during the t-th global cycle, and hence, y_i and x_{i+k} are at ports I_2 and I_1 of cell k, respectively, and y_l and x_{l+q} are at ports I_2 and I_1 of cell q, respectively. Assuming that the capacity of the x-links and y-links in the network are d_x and d_y, respectively, we may apply condition (1) to get

$$d_y(q-k) \geqslant (l-i) > 0 \qquad \text{and} \qquad d_x(q-k) \geqslant (l+q)-(i+k) > 0$$

that is

$$\min\{d_y, d_x - 1\} \geqslant \frac{l-i}{q-k} > 0. \tag{3}$$

Along the same line of thinking, condition (2) is translated to

$$(a_{i,i+k} \in CF_t, a_{l,l+k} \in CF_\tau) \text{ and } (i > l) \implies t > \tau. \tag{4}$$

Computation fronts for any specific input matrix pattern may be constructed by applying (3) and (4). More specifically, if each element $a_{i,i+k}$ of the input is associated with a position (i,k) as shown in Figure 3, then, equation (3) bounds the slope of the line segment joining any two elements in the same front. For example, for $d_y = 3$ and $d_x = 4$, we obtain

$$a_{i,i+k}, a_{l,l+q} \in CF_t \implies 3 \geqslant \frac{l-i}{q-k} > 0. \tag{5}$$

Also (4) implies that computation fronts may not overlap. Hence, starting from the right upper element in the graphical representation of the input (Figure 3), we may construct the fronts recursively. More specifically, given $CF_1, \ldots, CF_{t-1}$, we construct CF_t such that

1) it includes as many elements of A as allowed by (5), and
2) there is no non zero elements of A between CF_t and CF_{t-1}.

As the above example shows, conditions (1) and (2) may be used to design algorithm for the automatic construction of the computation fronts, thus providing a method for the determination of the number of global cycles needed to complete the computations.

Finally, we would like to note that the structure of each local cycle does affect the permissible slopes of the computation fronts. More specifically, the condition (1) was derived assuming that each cell in Figure 4 performs a cycle of the form [read z ; compute ; write z]. If, however, the value of z is not altered, then it is possible to write the local cycle in the form [read z ; write z ; compute]. In this case, the condition (1) becomes

$$d_z(q-k) > l-i \geqslant 0. \tag{6}$$

The Execution Time of Pseudo-Systolic Computations

Let N be the number of global cycles needed to complete a specific pseudo systolic computation. If $\tau_o << \tau_1$, then the execution time of the computation is $T \approx N\tau_1$. However, if we are not willing to neglect τ_o, then T may be computed from

$$T = \sum_{t=1}^{N} (\Delta_t + \tau_1 - \tau_o) \tag{7}$$

where Δ_t is the time for the communication phase in the t-th global cycle. An upper bound may be imposed on Δ_t, $t = 1, \ldots, N$ by studying the changes in the data profiles which take place during the communication phases. This is illustrated in the remaining of this section by means of the previous matrix/vector multiplication example.

First, we define for each global cycle t the x and y data profiles. Namely, the x-data profile is a function $xP_t : [-w, w] \times \{0,1,2,\ldots\} \rightarrow \{1,\ldots,n\}$, defined such that $xP_t(k,u) = i$ if, at the end of the t-th global cycle, x_i is the u-th element in the x-link queue at cell k, and $xP_t(k,u) = \uparrow$ (undefined) if the length of that queue is less than u. For simplicity, we let $xP_t(k,0)$ be the last value of x read by cell k during the t-th global cycle. The y-data profile is defined in a similar way.

Next, we show how to construct xP_t from CF_t (the construction of yP_t is similar). For each cell $k \in M_t$, we know that $xP_t(k,0)=i+k$, where $a_{i,i+k} \in CF_t$. Moreover, given any two cells $k, q \in M_t$ such that 1) $a_{i,i+k}$, $a_{l,l+q} \in CF_t$ and 2) for $k < c < q$, c is not in M_t, we know that the elements $x_{i+k}, \ldots, x_{l+q-1}$ should occupy consecutive locations on the x-link queues, starting at cell k. In other words, the profile between cells k and q is obtained from:

```
IF ($k$ is the largest cell in $M_t$) THEN $last = n$ ELSE $last = l + q - 1$
$\beta = 0; u = 0$
For $\lambda = i+k, \ldots, last$   DO
        $xP_t(k+\beta, u) = \lambda$
        IF ($u < d_x$) THEN $u = u + 1$ ELSE $u = 0; \beta = \beta + 1$
For c = $\beta+1, \ldots, q-1$   DO $xP_t(c,0) = last$
```

At time 0, we may assume that all the elements are stored in the buffer of cell w, that is

$$xP_o(k,q) = \begin{cases} q & \text{if } k = w \text{ and } q = 1,\ldots,n \\ \uparrow & \textit{otherwise.} \end{cases}$$

We also define the pseudo inverse function loc_t such that $loc_t(x_j)=k$ if $xP_t(k,u)=j$ for some u. Now, during the t-th communication phase (denoted from now on by CP_t). The data movement in the network causes a change in the x and y profiles from xP_{t-1} and yP_{t-1} to xP_t and yP_t. If we assume that $d_x = d_y$, then, it is easy to show that $xP_t(k,u) = yP_t(k,u)+k$.

Hence, changes in the two profiles occur simultaneously and the time Δ_t of CP_t may be calculated by considering only the change in one profile, say the x-profile.

Consider a specific cell k and let $xP_{t-1}(k,0) = i+k$ and $xP_t(k,0) = l+k$. Clearly, during CP_t, cell k should execute $xP_t(k,0) - xP_{t-1}(k,0) = l-i$ trivial local cycles. On the other hand, if $loc_{t-1}(x_{l+k}) = k'$, then x_{l+k} should travel across $(k'-k)$ cells during CP_t, in order to reach its position in xP_t. In other words, the communication activity in cell k during CP_t requires a time

$$\Delta_t(k) \leqslant (xP_t(k,0) - xP_{t-1}(k,0))\tau_o \;+\; (k'-k)\tau_o .$$

But CP_t terminates when the communication activities in all the cells terminate, that is

$$\Delta_t = \max\{\Delta_t(k);\, -w \leqslant k \leqslant w\}.$$

This, when used in (7) provides an estimate for the execution time of the pseudo systolic computation in terms of the parameters τ_o and τ_1.

APPLICATION TO 2-D NETWORKS

In this section we demonstrate how to construct the computation fronts for 2-D computations with trivial/non trivial local cycles. More specifically, we consider the multiplication $C = AB$ of two $n \times n$ banded sparse matrices on a $(2w+1) \times (2w+1)$ array (Weiser and Davis, 1981), where w is the half band width of A and B. The array is shown in Figure 5 along with its input. For simplicity, we assume that all the elements in the bands of A and B are supplied to the network, and that each cell is capable of recognizing and skipping trivial operations. We also assume that the capacities of the queues on the internal communication links are arbitrarily large and that each queue does initially contain one data item, namely a zero. More precisely, the operation of each cell is described by

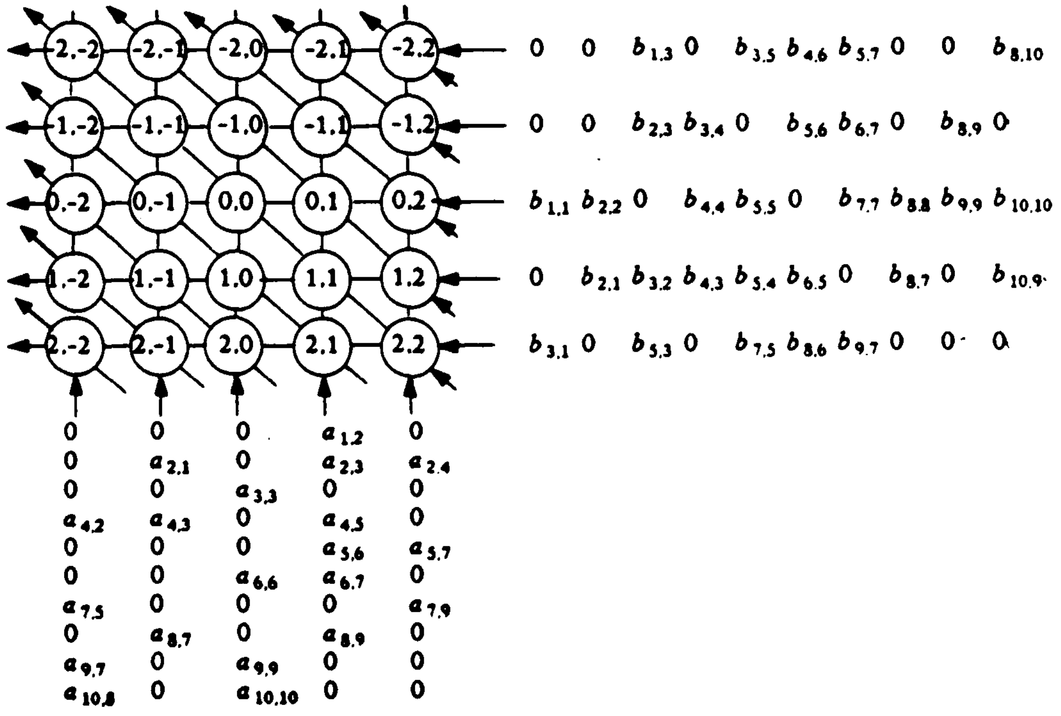

-2,-2
-2,-1
-2,0
-2,1
-2,2
-1,-2
-1,-1
-1,0
-1,1
-1,2
0,-2
0,-1
0,0
0,1
0,2
1,-2
1,-1
1,0
1,1
1,2
2,-2
2,-1
2,0
2,1
2,2
0 0 $b_{1,3}$ 0 $b_{3,5}$ $b_{4,6}$ $b_{5,7}$ 0 0 $b_{8,10}$
0 0 $b_{2,3}$ $b_{3,4}$ 0 $b_{5,6}$ $b_{6,7}$ 0 $b_{8,9}$ 0
$b_{1,1}$ $b_{2,2}$ 0 $b_{4,4}$ $b_{5,5}$ 0 $b_{7,7}$ $b_{8,8}$ $b_{9,9}$ $b_{10,10}$
0 $b_{2,1}$ $b_{3,2}$ $b_{4,3}$ $b_{5,4}$ $b_{6,5}$ 0 $b_{8,7}$ 0 $b_{10,9}$
$b_{3,1}$ 0 $b_{5,3}$ 0 $b_{7,5}$ $b_{8,6}$ $b_{9,7}$ 0 0 0
0 0 0 $a_{1,2}$ 0
0 $a_{2,1}$ 0 $a_{2,3}$ $a_{2,4}$
0 0 $a_{3,3}$ 0 0
$a_{4,2}$ $a_{4,3}$ 0 $a_{4,5}$ 0
0 0 0 $a_{5,6}$ $a_{5,7}$
0 0 $a_{6,6}$ $a_{6,7}$ 0
$a_{7,5}$ 0 0 0 $a_{7,9}$
0 $a_{8,7}$ 0 $a_{8,9}$ 0
$a_{9,7}$ 0 $a_{9,9}$ 0 0
$a_{10,8}$ 0 $a_{10,10}$ 0 0

Figure 5. A matrix/matrix multiplication computation

```
ALG3: Repeat
        1) wait until the queues on ports $I_1$, $I_2$, and $I_3$ are not empty
        2) $\alpha \leftarrow [I_1]$; $\beta \leftarrow [I_2]$; $\gamma \leftarrow [I_3]$
        3) $O_1 \leftarrow \alpha$; $O_2 \leftarrow \beta$
        4) IF ($\alpha \neq 0$ and $\beta \neq 0$) THEN4.2) wait for SYNC
                                           4.2) $\gamma = \gamma + \alpha * \beta$
        5) $O_3 \leftarrow \gamma$
```

Note that in ALG3, we assumed pseudo systolic operation. Of course, the normal data driven operation may be obtained by removing step 4.1 from the local cycle of each cell.

The multiplication $C = AB$ may be decomposed into $C = \sum_{r=-w}^{w} AB_r$, where each B_r contains the elements in the r-th off diagonal of B. In the network of Figure 5, the operation $C_r = AB_r$ is performed by the cells in row r of the 2-D array. Given that each row (r, k), $k = -w, \ldots, w$ of cells is a linear array, we may apply the same rules discussed earlier to the construction of the fronts for that row. These fronts will be denoted by $CF_1^{(r)}, CF_2^{(r)}, \ldots$

The t-th computation front of row r, $CF_t^{(r)}$, may be defined as either the set of elements of A, or the set of elements of B, which are operated upon during the t-th global cycle. The two sets are related and may be derived from each other. Here, we will choose the first alternative, that is, we will use A to represent the propagation of the computation.

The condition which relates the elements in the same front may be derived as follows: Let $a_{i,i+k}$ and $a_{l,l+q}$ be in $CF_t^{(r)}$. Then $b_{i+k,i+k-r}$ and $b_{l+q,l+q-r}$ should be cells (r, k) and (r, q), respectively, during the t-th global cycle. Noting that the values on the b data stream are written in ALG3 as soon as they are read, we may apply (6) to get

$$\infty > \frac{l-i}{q-k} \geqslant -1 \qquad (8)$$

which determines the slope of the computation fronts. In Figure 6, we show the fronts for the computation of Figure 5. Note that, for any given row r, some non zero element of A are not included in any front (in Figure 6, these elements are masked by a circle). More specifically, if $b_{j,j-r} = 0$, for some j, then by ALG3 the operation $a_{j-q,j} * b_{j,j-r}$ is skipped for $q = -w, \ldots, w$, and hence the elements $a_{j-q,j}$ are excluded from any front for row r.

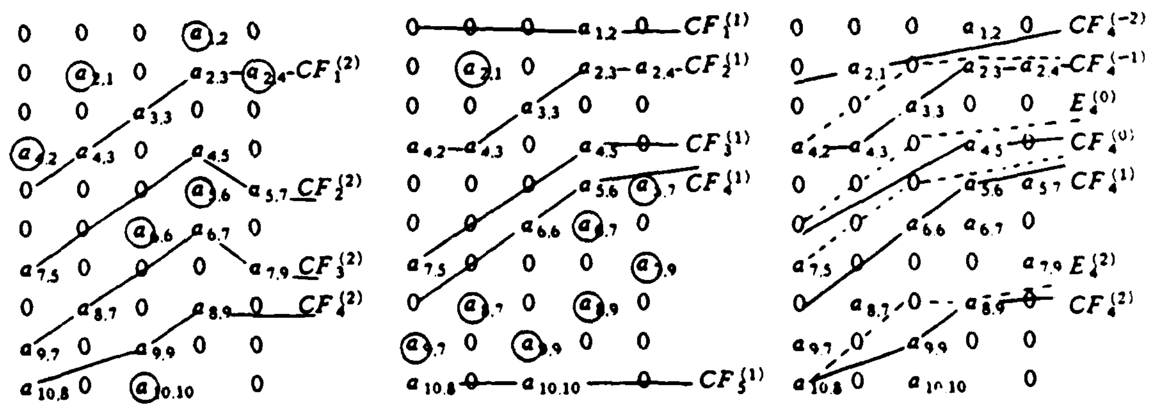


Figure 6. The computation fronts

Unfortunately, the computations performed by the different rows in the array are not independent. Namely, row $r-1$ receives data from row r. Loosely speaking, this implies that $CF_t^{(r-1)}$ follows $CF_t^{(r)}$. Stated differently, during any global cycle t, the fronts $CF_t^{(-w)}, \ldots, CF_t^{(w)}$ are ordered in a non overlapping way.

In order to be more specific, consider two consecutive rows r and $r-1$, and assume that $a_{i,i+k} \in CF_t^{(r)}$ and $a_{l,l+q} \in CF_t^{(r-1)}$ with $k > q$. The application of the condition (6) on both the a-data stream and the result stream along the 45° links gives

$$l < i + (k-q-1). \tag{9}$$

This means that if $CF_t^{(r)}$ passes through position (i,k), for some i and k, then $CF_t^{(r-1)}$ may not pass by a position (l,q) which violates (9). For a fixed i and k, equation (9) is a straight line with slope 135° starting at (i,k). Hence, (l,q) should not cross that line. The piecewise linear curve composed from such lines for all the elements in $CF_t^{(r)}$ is called the envelope of $CF_t^{(r)}$, which we denote by $E_t^{(r)}$. The envelope $E_t^{(r)}$ should be used to separate $CF_t^{(r-1)}$ from $CF_t^{(r)}$ during the construction of the former (see Figure 6).

In summary, after the construction of the computation fronts $CF_t^{(r)}$, $t = 1,2,...$ for row r, the fronts for row $r-1$ may be constructed as follows.

1) Mask the elements of A which correspond to zeroes in the $(r-1)$–st diagonal of B.
2) Compute the envelopes $E_t^{(r)}$, $t = 1,2,...$ for the fronts in row r.
3) Construct the fronts in row $r-1$ recursively, such that, given $CF_1^{(r-1)},....,CF_{t-1}^{(r-1)}$, the front $CF_t^{(r-1)}$ satisfies the following.
   1) it contains as many elements of A as allowed by (8).
   2) there is no non zero elements of A between $CF_t^{(r-1)}$ and $CF_{t-1}^{(r-1)}$
   3) $CF_t^{(r-1)}$ does not cross $E_t^{(r)}$

COMPUTATIONS WITH COMPLEX DATA DEPENDENT CYCLES

The concept of irregular computation fronts may be applied to data dependent network even when the execution of each local cycle is more complex than the simple trivial/non trivial scheme discussed so far. More specifically, we will discuss the case in which the basic local cycle time may be any multiple of a unit time τ_1. In this case a pseudo systolic computation may be assumed, such that successive snap shots of the computation fronts are obtained at τ_1 time intervals.

For example, let each input item $a_{i,i+k}$ in Figure 1 be a sparse $m \times m$ submatrix and each y_i and x_{i+k} be an m-dimensional subvector, and assume that step 3 of ALG1 is a "smart" matrix/vector operation which skips trivial multiplications and additions. In other words, the time consumed in the operation $y_i = y_i + a_{i,i+k} x_k$ in step 3 is equal to $p_{i,i+k} \tau_1$, where $p_{i,i+k}$ is the number of non zero elements in $a_{i,i+k}$ and τ_1 is the time of a multiply/add operation.

The consistency of data flow condition (1) may be applied to derive the same equation (5) which governs the slopes of the computation fronts. However, a cell which operates on an element $a_{i,i+k}$ terminates the corresponding local cycle in time $p_{i,i+k} \tau_1$, which means that $p_{i,i+k}$ computation fronts should pass through $a_{i,i+k}$. In Figure 7, we show the computation fronts for the case in which each input $a_{i,i+k}$ is a 2×2 submatrix, and hence $0 \leq p_{i,i+k} < 4$. Non zero elements are simply denoted by an x.

The time for communication phases may be computed as discussed earlier, except that, now, the time τ_o required for steps 2 and 5 is the time for the transmission of $m \times m$ matrices and m-dimensional vectors.

It was noticed earlier that the separation between any two fronts CF_{t-1} and CF_t is an indication of the communication activities during the t-th global cycle. However, as seen in Figure 7, only a small fraction of the cells are involved in communication during any specific global cycle, due to the slow propagation of fronts. Hence, the separation between communication activities and computation activities does, in this case, result in an estimate of the execution time, which is rather pessimistic.

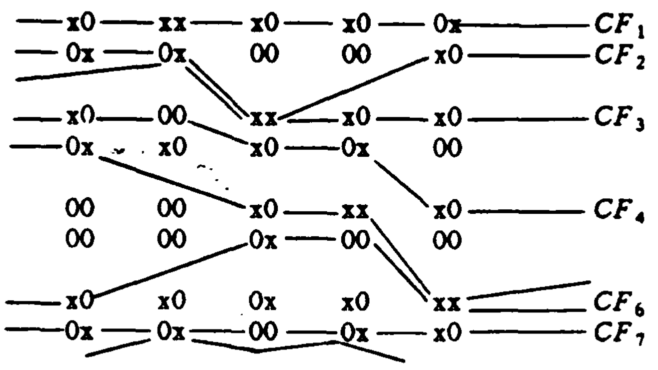


Figure 7. Partitioned Matrix/Vector multiplication

A less pessimistic bound on the execution time may be obtained if we assume that τ_0 is a multiple $\sigma \tau_1$ of the unit time τ_1 and redefine a global cycle to be a span τ_1 of time during which any particular cell may process data (step 3 of ALG1), communicate data (steps 2 and 5) or sit idle (steps 1 and 4). In this case, the number of fronts passing through any particular element $a_{i,i+k}$ is augmented to $p_{i,i+k}+\sigma$, and the execution time of the network is given by $N\tau_1$ where N is the total number of fronts. We will not discuss this approach here any further.

CONCLUSION

We presented a methodology for the estimation of the efficiency of data driven networks with data dependent operations. It is based on the irregular propagation of computation fronts and it takes into account the effect of internal data conflict without any assumption about the architecture of the network or the technology used to implement it. The main objective of the paper was to illustrate the methodology and the different related concepts. Its application to the performance study of specific networks is presented in Melhem (1986a) and (1986b).

REFERENCES

Kung, H. T. (1982). Why Systolic Architecture. Computer 15.1, 37-46.

Kung, S. Y. (1984). On Supercomputing with Systolic/Wavefront Array Processors. Proceedings of the IEEE, 72.7, 867-884.

Melhem, R. G. (1986a). Application of Data Driven Networks to Sparse Matrix Multiplication. Proceedings of the 1986 International Conference on Parallel Processing, to appear.

Melhem, R. G. (1986b). Parallel Solution of Linear Systems with Sparse Striped Matrices: Parts 1 and 2, Technical Reports ICMA-86-91/92, University of Pittsburgh.

Weiser U. and Davis A. (1981). A Wavefront Notation Tool for VLSI Array Design, in VLSI Systems and Computations, ed. by H. T. Kung, B. Sproull and G. Steele, Computer Science Press.

END

DTIC

9-86