العنوان:	تحويل الخوارزميات وحذف البث البياني لغرض التنفيذ النبضي
المصدر:	مجلة الزرقاء للبحوث والدراسات
الناشر:	جامعة الزرقاء - عمادة البحث العلمي
المؤلف الرئيسي:	الجنابي، علاء
مؤلفين آخرين:	يعقوب، نادية، الياسري، سناء(م . مشارك)
المجلد/العدد:	مج 5, ع 2
محكمة:	نعم
التاريخ الميلادي:	2003
الشهر:	كانون أول / شوال
الصفحات:	37 - 52
رقم MD:	351580
نوع المحتوى:	بحوث ومقالات
قواعد المعلومات:	EduSearch, EcoLink, IslamicInfo, HumanIndex, AraBase
مواضيع:	الخوارزميات التكرارية ، البث البياني ، المصفوفات النبضية ، الإحلال، المعادلات المتداخلة
رابط:	http://search.mandumah.com/Record/351580

16) Li, G. J., and B. W. Wah, "The Design of Optimal Systolic Arrays", **IEEE Trans. Comp., Vol. C-34, No. 1**, Jan. 1985, pp. 66-77.
17) Gusev, M., and D. J. Evans, "**Algorithm Transformation for the Data Broadcast Elimination Method**", Computer Studies Dept., Rep.646, Nov 1991, Loughborough University, UK.
18) Quinton, P., "Automatic Synthesis of Systolic Arrays from Uniform Recurrence Equations", **Proc. Of the 11th Annual Intern. Symposium on Computer Architectures**, 1984, pp. 208-214.
19) Rajopadhye, S. V., and R. M. Fujimoto, "Synthesizing Systolic Arrays from Recurrence Equations", **Parallel Computing, Vol. 14**, 1990, pp. 163-189.
20) Yaacoby, Y., and P. R. Cappello, "Scheduling a System of Nonsingular Affine Recurrence Equations onto Processor Arrays", **J. of VLSI Signal Processing, Vol. 1,** 1989, pp. 115-125.
21) Gusev, M., and T. Tasic, "A Method for Broadcast Elimination", **Parallel Computing '91, Int. Conf., London**, 1992, pp. 303-310.
22) Lam, M. S., "**A Systolic Array Optimizing Compiler**", Ph.D. thesis, Carengie-Mellon University, USA, 1987.

تحويل الخوارزميات وحذف البث البياني لغرض التنفيذ النبضي

علاء الجنابي

نادية يعقوب

سناء الياسري

الملخص

تستند هذه الدراسة إلى مسألة بناء مصفوفات نبضية من تمثيل المستوى للخوارزميات التكرارية. وقد تم تطوير طريقة لتحويل الخوارزمية التكرارية إلى صيغة أحادية الإحلال من المعادلات المتداخلة، كما تم تقديم طريقة لحذف البث البياني. فالمعادلات المتداخلة الناتجة ذات اعتمادية ثابتة، ويمكن أن تطبق مباشرة على المصفوفات النبضية باستخدام طريقة اعتمادية البيانات لتنتج تصاميم متعددة.

mapping on SAs and lead to different designs using the dependence method.

References

1) Kung, H. T., “Why Systolic Architectures?”, **computer J., Vol. 15, no.1**, Jan. 1982, pp. 37-46.

2) Koc, C. K., P. R. Capello, and E. Gallopoulos, “Decomposing polynomial Interpolation for Systolic Arrayes”, **Intern. J. Computer Math., Vol. 38**, 1991, pp. 219-239.

3) Sarkar, S., A. K. Majumdar, and R. K. Sen, “A Hardware Efficient Systolic Solution to the Two-Dimensional Discrete Forier Transform”, **Microprocessing and Microprogramming, Vol. 33**, 1991, pp. 111-117.

4) Evans, D. J., and Gusev M., “Systolic and VLSI Processor Arrays for Matrix Algorithms”, **Parallel and Distributed Computing Handbook**, Editors: Albert Y. Zomaya, McGraw-Hill Pub., N.Y., 1996, pp. 500-536.

5) Tsay, J. C., and P. Y. Chang, “Some New Designs of 2D Arrays for Matrix Multiplication and Transitive Closure”, **IEEE Trans. On Parallel and Distributed System, Vol. 6, No. 4,** April 1995, pp. 4-8.

6) Evans, D. J., “Block Matrix Multiplication and LU Factorization Systolic Arrays”, **Int. Jour. Comp. Maths. Vol. 76, No. 1,** 2000, pp. 45-57.

7) Lin, F. C., and k. Chen, “On the Design of An integrated Systolic Array for Solving Simultaneous Linear System of Equations”, **Computer J., Vol. 33, No. 3**, 1990, pp. 252-260.

8) Melhem, Rami, “A Systolic Accelerator For Iterative Solution of Sparse Linear Systems”, **IEEE Trans. Comp., Vol. 38, No.11** Nov. 1989, pp. 1591-1595.

9) Evans D. J. and Kane A. J., “Systolic Neural Algorithms”, ***Int. Jour. Comp. Maths.***, **Vol. 70, No. 4**, 1999, pp. 617-647.

10) Kumar, V. K., and Y. C. Tsai, “ Mapping Dynamic Programming onto a Linear Systolic Array”, **J. of VLSI Signal Processing, Vol. 1**, 1990, pp. 335-343.

11) Gusev, M., and D. J. Evans, “**New Linear Systolic Array for String Comparison Algorithm**”, Computer Studies Dept., Rep. No. 658, March 1993, Loughborough University, UK.

12) Sarkar, S., A. K. Majumdar, and R. K. Sen, “Enhanced Systolic Arrays Implementation for Some Graph Problems”, **Microprocessing and Microprogramming, Vol. 40**, 1994, pp. 499-520.

13) Al-Ubaidy, Alaa M. E., “**A Software Tool for Designing Systolic Arrays**”, M. Sc. Thesis, Basrah University, Iraq, 1995.

14) Moldovan, D. I., “On the Design and Analysis of VLSI Systolic Arrays”, **proc. IEEE, Vol. 71**, Jan. 1983, pp.113-120.

15) Moldovan, D. I., “On the Analysis and synthesis of VLSI Algorithm”, **IEEE Trans. Comp., Vol. 31**, Nov. 1982, pp.1121-1126.

$$D_1=\begin{bmatrix}1 & 1 & 0\\ 0 & -1 & 1\end{bmatrix},$$
$$D_2=\begin{bmatrix}-1 & 1 & 0\\ 0 & -1 & 1\end{bmatrix},\quad D_3=\begin{bmatrix}1 & -1 & 0\\ 0 & 1 & 1\end{bmatrix},$$
$$D_4=\begin{bmatrix}1 & 1 & 0\\ 0 & -1 & -1\end{bmatrix},$$
$$D_5=\begin{bmatrix}-1 & -1 & 0\\ 0 & 1 & -1\end{bmatrix},\quad D_6=\begin{bmatrix}1 & -1 & 0\\ 0 & 1 & -1\end{bmatrix},$$
$$D_7=\begin{bmatrix}-1 & 1 & 0\\ 0 & -1 & -1\end{bmatrix},\quad D_8=\begin{bmatrix}-1 & -1 & 0\\ 0 & 1 & 1\end{bmatrix}.$$

4.2 The Polynomial Problem

The polynomial problems is defined iteratively as:

$$p_n(x)=\sum_{j=0}^{n} a_j * x^j$$

where x represents the points at which the polynomial is to be evaluated, $a_0, \ldots, a_n$ represents the coefficients, and n is the degree of the polynomial.

By using Horner rule the following is obtained:

$p_n(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x\, a_n) \ldots ..))$

This form can be written in the following form:

```
For $i$:= $n$-1 to 0 do
    $p$:= p * $x$ + $a[j]$
```

where p is initialized to $a[n]$. If a series of points x_i, $0 \le i \le$ m, are given and we wish to compute $p(x_i)$ then we have:

```
For $i$:= 1 to $m$ do
    For $j$:= $n$-1 to 0 do
        $p[i]$:= $p[i]$ + $x[j]$ * $a[j]$
```

where $p[1], \ldots\ldots p[m]$ are initialized to $a[n]$. The equivalent SURE is:

```
For $i$:=1 to 3 by 1 do
  For $j$:= 2 to 0 by -1 do
    Begin
      $p[i,j]$:=$p[i,j+1]$+$x[i,j+1]$*$a[i-1,j]$
      $x[i,j]$:=$x[i-j+1]$
      $a[i,j]$:= $a[i-1,j]$
    End;
```

The initial points are:
$p[i,3]$:=$a[3]$,
$x[i,3]$:=$x[i]$,
$a[0,j]$:=$a[j]$

The final results are: $p[i,0]$

The possible dependence matrices (dependence vectors order is: x, a, and p):

$$D_1=\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & -1\end{bmatrix},$$
$$D_2=\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & -1\end{bmatrix},\quad D_3=\begin{bmatrix}0 & -1 & 0\\ -1 & 0 & -1\end{bmatrix},$$
$$D_4=\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 1\end{bmatrix}$$
$$D_5=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 1\end{bmatrix},$$
$$D_6=\begin{bmatrix}0 & -1 & 0\\ -1 & 0 & 1\end{bmatrix},\quad D_7=\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 1\end{bmatrix},$$
$$D_8=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & -1\end{bmatrix}.$$

5. Conclusion

This paper studied the problem of data broadcast elimination in systolic algorithms. A method to develop an iteractive algorithm to a single assignment form of recurrence equation is presented. The resultant SARE is with DB property that must be eliminated. Therefore, we presented a method for data broadcast elimination that converts SARE to SURE, which is suitable for mapping onto SAs. The presented method is clear and easy to implement and produces different presentation (dependence matrices) for the algorithms that are suitable for direct

Step 1: consider the function $x(m_1, m_2)$ where $1 \leq m_1, m_2 \leq 2m$. The system of linear equations can be obtained as follows: $i+j= m_1$, and $j+k= m_2$. This is a system of two equations in three variables (n=3).

Step 2.2: since k is the last index, the equation $k= m_3$, $1 \leq m_3 \leq m$, is added to the system. Then this system is solved by Gauss elimination method to yield the first point as $p_1=[m_1-m_2+m_3,\ m_2-m_3,\ m_3]^T$ and simply replace m_3 by m_3+1 to obtain the second point as $p_2=[m_1-m_2+m_3+1,\ m_2-m_3-1,\ m_3+1]^T$.

Step 3: the obtained propagation vector $r=p_2-p_1=[1\ \ -1\ \ 1]^T$.

Step 4: the following RE is obtained according to r:
$x(i,j,k):=x(i-1,j+1,k-1)$. Thus, the function in the original equation is replaced with $x(i-1,j+1,k-1)$.

Step 5: the following initial conditions are satisfied:

$x(0,j,k):=x(j,j+k)$,
$x(i,j,0):=x(i+j,j)$,
$x(i,m+1,k):=x(i+m+1,m+1+k)$.

From these steps, we obtain the equivalent SURE as follows:

$y(i,j,k):=f(\ldots\ldots, x(i-1,j+1,k-1), \ldots\ldots)$,
$x(i,j,k):=x(i+1,j+1,k-1)$,
$1 \leq i,j \leq \mathrm{m}$

with the initial conditions:

$x(0,j,k) := x(j,j+k)$,
$x(i,j,0):= x(i+j,j)$,
$x(i,m+1,k):=x(i+m+1, m+1+k)$.

4. Applications

In our experimental work, two problems are considered, the convolution and the polynomial problems which are widely used in image processing applications. SURE for both problems are shown considering that (n=5 and k=3) for the convolution problem and (n=2 and m=3) for the polynomial problem respectively. The possible dependence matrices for both problems are also shown.

4.1 The Convolution Problem

The convolution problem is defined iteratively by:

$$y_i = \sum_{j=1}^{k} w_j x_{i+j-1},\quad 1 \leq i \leq n-k+1,\ n \geq k$$

and its single assignment form is as follows:

```
For $i$:= 1 to $n$-$k$+1 do
  For $j$:= 1 to $k$ do
    $y[i]:=y[i] + w[j] * x[i+j-1]$
```

where $y[1],\ldots, y[n-k+1]$ are initialized to zero. The equivalent SURE is:

```
For $i$:=1 to 3 by 1 do
  For $j$:= 1 to 3 by 1 do
   Begin
    $y[i,j]:=y[i,j-1]+w[i-1,j] *x[i-1,j+1]$
    $w[i,j]:=w[i-1,j]$
    $x[i,j]:= x[i-1,j+1]$
  End;
```

The initial points are:
$y[i,0]:=0$,
$w[0,j]:=w[j]$,
$x[0,j+1]:=x[j]$,
$x[i-1,4]:=x[i+2]$

The final results are: $y[i,3]$

The possible dependence matrices are (dependence vectors order is: w, x, and y):

applying two consecutive values for the missing index as follows:

Step 2.1: In case of $(n-1)$ equations in $n-1$ variables, this system can be solved to give the values of the $(n-1)$ variables, which do not depend on the value of the missing index (i.e., has zero column in matrix A). So these values are the same in the two consecutive points resulted from applying two consecutive values for the missing index.

Step 2.2: In case of $(n-1)$ equations in n variables, this system cannot be solved until an equation, which is constructed from the missing last index and a hypothetical constant, which should be in the range of the last index, is added. This new equation becomes the n^{th} equation in the system, which gives a value for the last index.

Step 3: Compute the dependence (or propagation) vector, which is the difference vector between the two consecutive points.

Step 4: Add new recurrence equation to the SRE according to the obtained propagation vector.

Step 5: Determine the initial conditions of the SRE, i.e., the value of each function in its initial points by substituting the points that include lower bound-1 as a value for the index propagated by negative value or upper bound+1 for the index propagated with positive value in the original SRE.

Now we consider some examples of functions with DB to explain how our method is applied to these functions converting the broadcasting into propagation.

Example 4: Consider the following SRE:

$x(i,j) := f(\ldots, b(j,j-1), \ldots)$, $1 \le i, j \le m$

We apply the previous method on this system as follows:

Step 1: consider the function $b(m_1,m_2)$ where $1 \le m_1 \le m$, $1 \le m_2 \le m-1$. the system of linear equations can be obtained as follows: $j=m_1$ and $j-1=m_2$ yields that $j=m_2+1=m_1$. This is a system of one equation in two variables ($n=2$).

Step 2.1: since i is the missing index, then we let $i=m_2$ and $i=m_2+1, 1 \le m_2 \le m$, to get the following two consecutive points: $p_1=[m_2, m_1]^T$ and $p_2=[m_2+1, m_1]^T$.

Step 3: the obtained propagation vector $r=p_2-p_1=[1\ 0]^T$.

Step 4: the following RE is obtained according to r: $b(i,j):=b(i-1,j)$. Thus the function in original equation can be replaced with $b(i-1,j)$.

Step 5: the following initial condition is satisfied:

$$b(0,j):=b(j,j-1).$$

Therefore, the equivalent SURE obtained from DB elimination method is:

$$x(i,j):=f(\ldots\ldots, b(i-1,j), \ldots\ldots),$$
$$1 \le i, j \le \mathrm{m}$$
$$b(i,j):=b(i-1,j)$$

With the initial condition :

$$b(0,j)=b(j,j-1).$$

Example 5: Consider the following SRE:

$y(i,j,k) := f(\ldots\ldots, x(i+j, j+k), \ldots\ldots),$

$$1 \le i,j,k \le m.$$

The method is applied on this system as follows:

then pipelining the resultant (non-systolic) architecture by a technique called data pipelining which enables to derive SAs from SARE.

Although this approach does not requires any modification on the algorithm to be mapped onto SAs. It requires very complicated mathematical treatments to perform such mapping.

The other approach points out the DB problem in SARE and tries to find a method to eliminate such problem by converting SARE to an equivalent SURE. Gusev and Tasic [21] have followed this approach. They define the dependence vector $d=p-q$ between two index points p and q as $d=A*p+b$ where A is $N\times N$ constant matrix and b is n-dimensional vector. They did additional replacement, evaluating $q = p-d = p-A*p-B = (I-A)p-b$ and achieve $q=B*p-b$ where $B=(I-A)$ is $N\times N$ matrix of linear index dependence and I is $N\times N$ identity matrix. They analyzed the function $f_j(q)$ that is broadcasted to more than one index point p. This dictates the conditions q=const. Since $q=B*p-b$=const, it follows that $B*p=b+q$=const. So they addressed $B*p=r$ as a system of linear equations upon the vector p and the matrix B is with rank less than n, where r is the propagation vector according to which a new recurrence will be added to SARE transforming the broadcasting into propagation.

However, they did not point out how the system $B*p=r$ can be solved to get the vector r. We follow the idea of second approach and propose method that is easy to implement and to construct.

3.3.1 The Proposed Elimination Method

The basic idea for data broadcast elimination method depends on the definition of SARE. In this definition the indices of the function is given by $Ap+b$ which means that each index of p is changing according to a linear equation and the function is used in the points along the line of missing index, that we call the broadcasting line; i.e., The function is broadcasted to all points along this line. The space of index point is of integer coordinates with equal space, so finding the relation between two consecutive points on the broadcast line (as a dependence vector which is also called propagation vector) is the same as that between the others on the same line. Then we change the broadcasting into propagation, see Fig. 2 (c). Therefore, different algorithm presentations (dependence matrices) can be obtained by negating the propagation vector of each function separately, of every two functions together, and all functions together. The synthesis steps of stage two are applied for each dependence matrix that may result different systolic designs.

we can describe the elimination method as follows:

Step 1: Determine the system of linear equation for the proposed value of the broadcast function. This system is of n-1 equations in either n-1 variables or n variables.

Step 2: Determine two consecutive points on the broadcast line by ongoing manner without returning the intermediate values to the main memory. Thus, economy of storage is of minor importance for our purposes. Indeed, it is the structure of the relationships in the algorithm, which is central. Hence we need to distinguish between distinct uses of the same symbol.

Definition 9: A single assignment form is a form where every value of the computed functions is calculated only once during the execution of the algorithm.

Transforming an iterative form into single assignment form is done by introducing counter values and changing the Equation (1) in the following recursive form:

$f(p') = f(p'') * arguments$ ..... (2)

where:

$p' = [p_1, p_2, \dots, s]$

$p'' = [p_1, p_2, \dots, s-i_s]$

Practically this processing expands the index point by a parameter labeled with s, expressing the counter values. Adding the counter as a new parameter, the index space of the function is expanded and for each index point one function value is calculated. Therefore, a single assignment form of Equation 2 is obtained. New data dependence is introduced according to the resulting single assignment form. This dependence presents the propagation of the output value from one recurrence to another that has the following dependence vector:

$d = p' - p''$

$= [p_1, p_2, \dots, s] - [p_1, p_2, \dots, s-i_s]$

$= [0, 0, \dots, i_s]$.

Such d is always constant. This actually what we need for our systolic implementation.

After this processing for the iterative algorithm, if all of resultant REs are with DB then the DB elimination technique is applied for these functions. Otherwise, if there is one (or more) function with full index form and matrix A is not rank deficient then the system is stopped deciding that the input algorithm is I/O bound problem which is not suitable for systolic implementation [1].

3.3 Data Broadcast Elimination

Most work on the problem of synthesizing an SA from SREs is restricted to SURE. In this section, this restriction is relaxed to include SARE with data broadcast property. A method is presented to eliminate the data broadcast converting the SARE to an equivalent SURE.

There are two approaches to handle SARE. One approach maps SARE directly to SAs without a need to determine explicitly the DB problem. Yaacoby and Cappello [20] have applied this approach and gave sufficient conditions for SARE to be computable. Also, they gave necessary and sufficient conditions for existence of an affined schedule, along with a procedure that constructs the schedule vector, when one exists. Moreover, Rajopadhye and Fujimoto [19] have presented a technique for synthesizing systolic architectures form SARE by first determining timing and allocation functions and

Theorem 3: An algorithm with a function which is not defined by a full index form or defined with a full index from having two indices that depend on the same parameter can be expressed by a recurrence equation with affine data dependence.

Proof: Consider the following RE:

$f(p)=g(\ldots,f_i(\beta_i(p)),\ f_j(\psi_j(p)),\ \ldots)$

Let $q'=\beta_j(p)=[p_1,\ p_2,\ \ldots,\ p_s,0,p_{s+1},\ \ldots,\ p_n]$ and $q''=\psi_j(p)=[p_1,\ p_2,\ldots,\ p_s,\ \ldots,p_{k-1},\ p_k,\ p_{k+1},\ \ldots,\ p_n]$. q' and q'' are linearly depending on p, this is because $q'=A_1*p+b_1$ and $q''=A_2*p+b_2$, such that A_1 and A_2 differ from the identity matrix, where A_1 and A_2 respectively are:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & & \cdots & & \cdot & & & \cdot \\ \cdot & & \cdots & & \cdot & & & \cdot \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdot & & \cdots & & \cdot & & & \cdot \\ \cdot & & \cdots & & \cdot & & & \cdot \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & & & \cdot & & & \cdot & \cdot & & \cdot \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & & & & & \cdot & \cdot & & & \cdot \\ 0 & 0 & \cdots & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdot & & & & & \cdot & \cdot & & & \cdot \\ 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

From Theorem 2 and Theorem 3 we obtain that an algorithm with functions not expressed in full index form is in AREs with DB property.

3.2 Algorithm Transformation

All algorithms which are iterative, are suitable for parallel implementation, but we want to transform such algorithms in another form that permits an efficient mapping to VLSI architectures. In case of SAs, recurrence equations are the most adjustable forms.

The programmer heuristically writes his programs in an iterative form. This form is given by the following equation:

```
For $p_1 = L_1$ to $U_1$ by $I_1$ do
        .
        .
  For $p_n = L_n$ to $U_n$ by $I_n$ do
    For $p_s = L_s$ to $U_s$ by $I_s$ do
      $f(p) = f(p) * arguments$    .... (1)
```

where

- f is the function to be computed
- * is an arithmetic (logic) operation
- *arguments* are input data that are processed during the iteration.

The acceptable algorithm to our synthesizer must be presented in an iterative form which is the starting point for the synthesizer work.

3.2.1 Single Assignment Form

In writing programs, it is common to use the same symbol to represent several distinct quantities each of which stands for some intermediate results which is gradually replaced by more complete results. One reason for this practice is to reduce storage requirements. Inevitably, such a practice obscures the natural relationships that exist among the parts of the algorithm.

In the case of systolic algorithm in mind, the results are developed in an

3. Data Broadcast Problem

Data broadcasting occurs in SARE in the situation where one value of a considered function is used as a parameter for the computation of more than one function. It is also identified as a data transfer *one to many*, i.e., one data is broadcasted to many computations. Fig. 2(a) shows a broadcast function. Also "many to one" identified another type of DB; see Fig 2(b), which is not of our consideration.

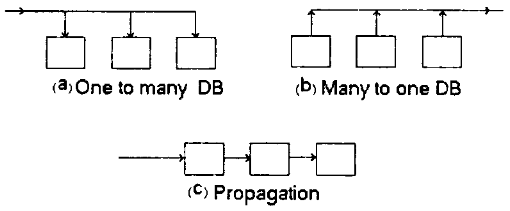

Fig. 2 Data Broadcasting

Lemma 1 [21]: Data broadcast in a SARE exists if there are at least two different index points in the domain of the system such that:
$(\exists\ p_1, p_2 \in I^n ; p_1 \neq p_2)$ and $(f(A * p_1 + b) = f(A * p_2 + b))$ or rank A is less than n.

Proof: Suppose that the computation of f for p_1 requires the function $f(q)$, but $Ap_1+b=Ap_2+b=A$ so $f(q)$ is used again when computing f for p_2. This means that the function $f(q)$ is broadcasted for two computations p_1 and p_2. The second condition is valid because $A(p_1-p_2)=0$ and $p_1 \neq p_2$, therefore rank A is less than n.

DB cannot be implemented on processor arrays because of the requirements of VLSI technology for local interconnection and communication. Therefore, the solution of data broadcast elimination problem has great potential. Section 3.3.1 suggests a method to eliminate data broadcast in SARE converting it into SURE.

3.1 Functions with data Broadcasting

If a function in a RE is not expressed in full index form then the output from the function δ_{i_j} is a vector with dimension smaller than n. Let $q=\delta_j(p)=[q_1, q_2, \ldots, q_n]$. Define the function Γ_j: $Z^n \rightarrow Z^{n-1}$, which is given by: $\Gamma_j(p) = [q_1, q_2, \ldots, q_{s-1}, q_{s+1}, \ldots, q_n]$. Notice two index points p_1 and p_2 that always exist such that $p_1 \neq p_2$, and $\Gamma_j(p_1)=\Gamma_j(p_2)$. So, the next theorem is valid.

Theorem 2 [21]: Function Γj: $Z^n \rightarrow Z^{n-1}$ in a recurrence equation defines a data broadcast.

The index space of the function not defined by full index form can be fulfilled by an identity mapping for existing indices and filling the missing indices by zero to obtain the index point q'. Define the function β_j: $Z^n \rightarrow Z^n$ and given by $\beta j\,(q)=q'$ such that: $q'_s=0$ and $q'_i = q_i$, $1 \leq i \leq n$ and $i \neq s$.

The above shows one class of the broadcast functions. The other class is the function defined in full index form, but there are two indices in this form depending on the same parameter in p. Define the function ψ_j: $Z^n \rightarrow Z^n$, which is given by: $\psi_j(p) = [q_1, q_2, \ldots, q_s, \ldots q_k, \ldots, q_n]$ such that $q_s=q_k$. It easy to see that Theorem 2 can be extend to include the function ψ that defines data broadcast.

interconnections and the links have a constant delay (again independent of location in the array). Thus if we imagine a 'snapshots' taken every time instant as the computation progress, we get a three-dimensional dependency structure in a space time domain. Any point, $p=[x, y, t]$ in the domain represents the computation that processor $[x, y]$ performs at time instant t. Since the architecture is systolic, the point $q=[x', y', t']$ that the computation performed by the processor $[x, y]$ at time t dependence on, can only be displaced by a constant in t-axis (because of constant delays), and by a small constant in the x-y plane (because of nearest-neighbor interconnections). Thus the dependencies are uniform, and the computation can be described by a SURE.

2.5 Affine Recurrence Equations

From Example 2, one can say that presenting an algorithm as SURE is not a commoning used way, and this way of presentation may be difficult to those who are not familiar with recurrence equations. A large number of interesting problems cannot naturally be expressed as SURE.

Also based on Theorem 1 it is clear that UREs are too close to the target implementation of SAs rather it is useful high level specification. UREs are more suitable as intermediate representation of the problem after specifying the problem by another form. A more general class of REs is the AREs whereas the name suggests, the dependencies are affine function of the point [19].

Definition 8: An SRE as given in Definition 4 is said to be a SARE *iff*
$\delta_{i_j}(p) = A_{i_j} + p + b_{i_j}$,

for $1 \le i \le m$, $1 \le j \le k$

where A_{i_j} is $N \times N$ constant matrix .

AREs are the superset of UREs, so if each A_{i_j} is the identity matrix then the system is SURE. In general, the SAREs raise the data broadcast problem that should be eliminated for efficient systolic implementation.

Example 3: A more appropriate specification for matrix multiplication would be the following SRE:
$w[i,j,k]:=w[i,j,k-1]+x[i,k,0]*y[k,j,0]$
with the initial conditions
$w[i,j,0]: = 0$,
$x[i,k,0]: =x[i,k]$,
$y[k,j,0]: =y[k,j]$.
From Definition 8 we can obtain the following:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, b_1 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$A_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

where A_1 and b_1 are associated with w, A_2 and b_2 are associated with x, A_3 and b_3 are associated with y. Since A_2 and A_3 are differing from the identity matrix, and the above SRE is affined.

function that denotes any significant computation while the others merely being auxiliary functions, and also there is one dependence vector for each function. The definition of each system is given in [18] as follows:

Definition 7: A SURE over the domain I^n is defined by:
$f_1(p)=g(f_1(p+b_1),f_2(p+b_2),\ldots,f_k(p+b_k))$
$f_2(p)=f_2(p+b_2)$

$$\begin{matrix}\cdot\\\cdot\\\cdot\\\cdot\end{matrix}$$

$f_k(p)=f_k(p+b_k)$
The dependence matrix of this system is as follows:

$$D=[d_1, d_2,\ldots, d_k]$$

where $d_1=-b_1$, $d_2=-b_2$ ……., and $d_k=-b_k$ and d_i is the dependence vector associated with the function f_i.

In spite of such restricted system, many interesting problems (typically numerical and matrix computations) can be expressed as such recurrences. The next example shows a SURE for matrix multiplication problem $W_{n\times n}=X_{n\times n}*Y_{n\times n}$ which is defined iteratively as:

$$w_{ij}=\sum_{k=1}^{n}x_{ik}*y_{kj},\ 1\le i,j\le n$$

Example 2: The following SURE, is defined over the domain:
$I^3=\{(i,j,k):\ 1\le i,j,k\le n\}$, computes the matrix $W_{n\times n}$ that result from multiplying matrix $X_{n\times n}$ by matrix $y_{n\times n}$ as follows:
$w[i,j,k]:=w[i,j,k-1]+x[i,j-1,k]*y[i-1,j,k]$
$x[i,j,k]:=x[i,j-1,k]$
$y[i,j,k]:=y[i-1,j,k]$
The initial conditions are:
$w[i,j,0]:=0$,
$x[i,0,k]:=x[i,k]$,
$y[0,j,k]:=y[k,j]$
and the final results are:
$w[i,j,n]$
This SURE can completely be described by $<I^3, D>$ where I^3 is defined above and D is as follows:

$$D=\begin{bmatrix}d_1 & d_2 & d_3\end{bmatrix}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

where d_1, d_2, and d_3 are associated with y, x, and w respectively.

For SURE, the data dependence method can be efficiently used for mapping algorithms onto SAs. However, the following theorem shows the relation between the actual work of SAs and SURE. Whereas the aim of this paper is the reverse of this theorem.

Theorem 1 [19]: For any SA, there exits a SURE that computes exactly the same function as the SA.
Proof: consider a processor in physical two-dimensional SA. The local memory of an individual processor in an SA can be viewed as a set of shift registers. Some of these can be directly viewed as (finite) delay on the communication lines, and the remaining (which correspond to values that are actually update during each cycle) can also be viewed as shift registers that connect the processor to itself. Each processor as such, independent of its location in the array, has nearest neighbor

in another index point then there is a data dependency between theses two index points which can be described by the data dependence vector d (the difference between the two index points). The dependence vectors of algorithm actually dictate its communication requirements.

Definition 5: d is said to be a dependence vector for a function f when the following holds if $f(p)=g(\ldots,f(q),\ldots)$ then $d=p-q$. The matrix $D=[d_1, d_2, \ldots, d_j]$ ($j \geq 1$) of all dependence vectors is called the dependence matrix.

Recall the SRE in Example 1, there are some data dependences between the functions available in this system. For example, $b(4,2)=a(5,2)*b(6,1)+b(4,1)$. The following dependence vectors can describe these dependences:

$d_1=(-1\ \ 1)^T$ for pair $< a(i, j), a(i+1, j-1)>$

$d_2=(-1\ \ 0)^T$ for pair $< b(i, j), a(i+1, j)>$

$d_3=(-2\ \ 1)^T$ for pair $< b(i, j), b(i+2, j-1)>$

$d_4=(\ 0\ \ 1)^T$ for pair $< b(i, j), b(i, j-1)>$

With these dependence vectors we form the dependence matrix D (the order of columns is not important)

$$D=[d_1\ d_2\ d_3\ d_4]=\begin{bmatrix} -1 & -1 & -2 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

For this algorithm all four dependence vectors exist in almost every index point of the index set. The dependence vector d_1 is associated with function a, whereas the rest vectors are associated with function b. Notice that the pair of points that generate a dependence vector may be obtained either from a function generated and used in a single equation, or from a function generated in one equation and used in other different equations. Therefore, one function may have more than one dependence vector.

2.4 Uniform Recurrence Equations

UREs define a computation where the dependencies can completely be described by a finite number of constant vectors, regardless of the size of the domain. The algorithm presented as SURE is modeled as $<I^n, D>$ where I^n is a lattice with nodes representing computations and the dependence vectors in D representing edges that connect these computations. The SRE in Example 1 is uniform. Each node of computation takes a unit execution time. The lattice is mapped onto a space-time domain, the time and space coordinates of each node indicate when and where in the array to perform the computation. This modeling of the problem is powerful for mapping regular computation onto a regular layout of simple cells [22]. The definition of SURE is as follows:

Definition 6: An SRE as given by Definition.4 is called system of URE (SURE) *iff*

$\delta_{i_j}(p) = p + b_{i_j}$, for $1 \leq i \leq m$, $1 \leq j \leq k$

where b_{i_j}'s are constant n-dimensional vectors.

Our work is restricted to the case where in SURE there is only one initial conditions of the SRE. Therefore, $L^n \cap I^n = \phi$.

Definition 4: The SRE over the domain I^n is defined by m mutual equations as [19, 20, 21]:

$$f_i(p) := g_i(f_{i_1}(\delta_{i_1}(p)), f_{i_2}(\delta_{i_2}(p)), \cdots, f_{i_k}(\delta_{i_k}(p)))$$

$$\text{for } 1 \le i \le m,\ k \ge 1,\ p \in I^n$$

where:

- The functions $f_1, f_2, \ldots, f_r$, $(r \ge m)$ are real valued functions with domain $I^n \cup L^n$:
  $f_h : I^n \cup L^n \rightarrow R$, where R is the set of reals and $1 \le h \le r$
- The notation f_{i_j} denotes the j^{th} parameter for calculating the i^{th} function in the system where:
  $i_j \in \{1, 2, \ldots, r\}$
- The sign := means that the function f_i is being calculated and gets the value of the function g_i.
- The function g_i is a real valued function defined as:
  $g_i : R^{i_k} \rightarrow R$, for $i_k \ge 1$
- The function δ_{i_j} are integer vector valued with domain I^n, i.e.,
  $\delta_{i_j} : I^n \rightarrow I^n \cup L^n$,
  for $1 \le i \le m$, $1 \le j \le k$

Notice that there are two types of functions in the SRE, i.e., the calculated and the input functions. The SRE is defined by m REs that compute the value of the calculated functions $f_1, f_2, \ldots, f_n$. The functions $f_{m+1}, f_{m+2}, \ldots, f_r$ that are used for calculating the SRE are called input functions. Also, notice that the initial conditions for SRE must be determined i.e., the value of the function that may be used in a point outside the index set.

The equations in Example 1 below are an example of a SRE and they are used to illustrate some of the previous definitions.

Example 1: consider the following equations:

$a(i, j) := a(i+1, j-1)$

$b(i, j) := a(i+1, j) * b(i+2, j-1) + b(i, j-1)$

for $1 \le i \le n-2$, $1 \le j \le n-1$

Thus we have:

- The index points:
  $p = [i, j]$, $1 \le i \le n-2$, $1 \le j \le n-1$.
- The initial points:
  $L^2 = \{(i, 0), (n-1, j), \text{and } (n, j), 1 \le i \le n-2, 1 \le j \le n-1\}$.
- This SRE is defined by two recurrence equations (m=2), and a, and b corresponds to the functions f_1 and f_2 of Definition 4 respectively.
- The functions g_1, g_2 are defined as follows:
- g_1: $R \rightarrow R$, where k=1, and given by: $g_1(x) = x$,
  g_2: $R^3 \rightarrow R$, where k=3, and given by: $g_2(x, y, w) = x*y+w$
- The *functions* $\delta_{1_1}, \delta_{2_1}, \delta_{2_2}, \delta_{2_3}$ are given as follows:

$$\delta_{1_1}(i, j) = (i+1, j-1),$$
$$\delta_{2_1}(i, j) = (i+1, j),$$
$$\delta_{2_2}(i, j) = (i+2, j-1),$$
$$\delta_{2_3}(i, j) = (i, j-1).$$

2.3 Data Dependence in System of Recurrence Equations

If a data item generated (calculated) in one index point is used converts SARE to SURE that can be efficiently mapped onto SAs using the data dependence method.

The rest of this paper is organized as follows: In section 2 we present a method to transform an iterative algorithm to a System of Recurrence Equations (SRE). Section 3 illustrates the DB problem and presents a method for DB elimination. In Section 4 we apply the proposed method on two applications. Section 5 concludes the results.

2. Algorithm Transformation

2.1 Recurrence Equations

REs have been well known to mathematician for expressing a large class of functions over the domain $M \subset Z$ where Z is the set of integers. The RE specifies f_n at the point $n \in M$ in terms of f at other points in the domain.

Definition 1: An equation that relates a number a_n in a sequence $a_1, a_2, \ldots, a_n, \ldots$ to some of its predecessors is called recurrence equation. A recurrence of the form:

$f(n)=a_n= c_1 a_{n-1}+c_2 a_{n-2}+ \ldots +c_r a_{n-r}$

is called a linear RE with constant coefficients if all the c_i's are constant, for $1 \leq i \leq r$, $r \geq 1$.

Consider for example the well-known factorial and fibonacci functions that are defined respectively as follows:

$$f(n)=\begin{cases}1 & \text{if } n=0\\ n*f(n-1) & \text{if } n>0\end{cases}$$

and

$$f(n)=\begin{cases}0 & \text{if } n=0\\ 1 & \text{if } n=1\\ f(n-1)+f(n-2) & \text{if } n>1\end{cases}$$

Of these two examples, the second imposes a linear RE, while the first does not.

However, our primary concern is in solving RE, i.e., addressing the following problem: Given a RE describing $\boldsymbol{a_n}$ in terms of some $\boldsymbol{a}$'s, determine a closed-form expression for $\boldsymbol{a_n}$, i.e., an expression for $\boldsymbol{a_n}$ that does not involves any $\boldsymbol{a}$ terms [19].

2.2 Algorithm Presentation as SRE

Our objective here is somehow different from the previous definition of REs. We are interested in using the recurrences as an algorithm for computing the function, and in implementing it on a systolic architecture.

We therefore make some definition considering SRE is defined for some points in the n-dimensional Euclidean space as index points and the domain of the SRE as index set as defined below.

Definition 2: Each point in the domain of SRE is called index point, denoted by p, and defined by:

$$p=[p_1, p_2, \ldots, p_n]^T$$

where $p_i \in Z$, $1 \leq i \leq n$.

Definition 3: The nonempty set $I^n \subset Z^n$ of all index points belonging to the domain of the SRE is an index set of the RE.

Remark: We denote by $L^n \subset Z^n$ the set of all index points that define the

Systolic algorithms exhibit some features that make them suitable for direct hardware implementations. They are highly parallel/pipelined algorithms, specified on the basis of simple operations, with a high degree of homogeneity in operation and regularity in communication. The early systolic algorithms were obtained, probably, in a heuristic way. They are oriented to matrix problems (matrix multiplication, LU-decomposition, etc). Later, automatic methodology to design systolic algorithms has been proposed. The benefits of a design methodology are saving in design time, correctness of designs and the possibility of obtaining several solutions and choosing the best one according to given criterion [13]. In fact, algorithms represented with nested loops [14, 15] or recurrences [16] are preferable with such methodologies.

The main concern in algorithm development for systolic implementation is the local communication between PEs in the array so that the share common bus should be avoided by increasing the processing speed, and the access to memory becomes very slow in comparison to processing speed. Therefore, these constraints modify the approaches in parallel algorithm designs. Data Broadcasting (DB) which may exist in many algorithms, means one data item (input or computed) is used in many computations. This can be realized only by a share common bus and by a global memory access. Therefore, the data broadcast elimination problem is of great interest [17].

Data dependence method [14, 15] is one of the famous techniques used for mapping algorithms onto SAs. This method accepts an algorithm with constant data dependence. Informally, an algorithm, which is suitable to be applied for data dependence method, is represented as a partially ordered subset of a multidimensional integer lattice (called index set). The points of the lattice correspond to (i.e., they are the indices of) computations, and the partial order reflects the dependencies between them. These dependencies are represented as vectors that connect points of the lattice. If a given dependence vector (precedence vector) between any two-lattice points is constant, then the dependence is said to be uniform. If all dependence vectors are uniform then the algorithm is said to be a uniform dependence algorithm.

A definition of such algorithm as a class of Recurrence Equations (REs) called System of Uniform Recurrence Equations (SURE), which is proposed by Quinton [18]. This form of presenting an algorithm is not the common one, so we first present a method to develop an iterative algorithm (which is usually used by programmers when coding their programs) to a single assignment form of REs. The resultant recurrences form a System of Affine Recurrence Equations (SARE), a more general class of REs, with DB property. Then a method for DB elimination is proposed which

Algorithm Transformation and Data Broadcast Elimination for Systolic Implementation

Alaa E. Aljanaby[*]
Nadia Y. Yousif[**]
Sana J. Alyaseri[***]

Abstract

The problem of synthesizing systolic arrays from a high level specification of an iterative algorithm is concerned. A method to develop the iterative algorithm to the single assignment form of recurrence equations, and a method for data broadcast elimination are presented. The resultant recurrences are with constant dependencies and can be mapped directly onto systolic array using the dependence method producing different designs.

Keywords: *Systolic Arrays, Data Broadcasting, Recurrence Equations.*

1. Introduction

Systolic Arrays (SAs) are first proposed by H.T. Kung [1] as an excellent matching to the criteria concerning the design of special-purpose VLSI systems. The systolic system consists of a set of simple interconnected cells of few types that move data in a regular fashion. An array structure for such a system provides simple, regular and local communication paths between cells. Information in SA flows between cells in a pipelined fashion, and communication with outside world occurs only at the boundary cells.

The basic principle of SA approach –in comparison with the traditional architecture of a computer system – is that the single processing unit is replaced by an array of Processing Elements (PEs) each capable of doing simple operations. A much higher computation throughput can be achieved without having to increase memory bandwidth, see Fig. 1. Thus focus of SA researchers is to make sure that once a data item is brought out from memory, it will be effectively used many times while traveling along the SA. SAs are not always linear; they can be two-dimensional, rectangular, triangular or hexagonal to make use of higher degree of parallelism. SAs can be effectively used in many applications such as digital signal and image processing [1, 2, 3], matrix arithmetic [4, 5, 6, 7, 8], neural networks [9], dynamic programming [1, 10], string comparison [11], and graph problems [12].

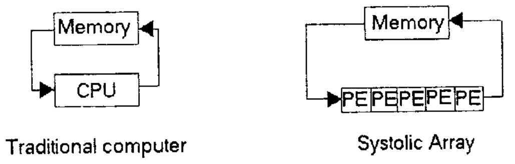

Fig. 1 Traditional Computer and Systolic Architectures

* Lecturer, Computer Dept., Zarka Private University, Zarka - Jordan.
** Associate Prof., Computer Dept., Philadelphia University, Jordan
***Lecturer, Al-Husien College, Al-Balqaa Applied University, Jordan