VECTOR REDUCTION METHODS FOR ARITHMETIC PIPELINES*

Lionel M. Ni
Department of Computer Science
Michigan State University
East Lansing, MI 48824

Kai Hwang
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Abstract: Vector reduction arithmetic accepts a vector as input and produces a scalar output. This class of vector operations forms the basis of many scientific computations. In a pipelined processor, a feedback loop is required to reduce vectors. Since the output of the pipeline depends on previous outputs, improper control of the feedback loop will destroy the benefit from pipelining. A generalized computing model is proposed to schedule the activities in a vector reduction pipeline. Two new vector reduction methods, ***symmetric*** and ***asymmetric***, are proposed and analyzed for pipelined processing. These two methods compare favorably with the known *recursive reduction* method in achieving higher pipeline utilization and in eliminating large memory for intermediate results. An *interleaving method* is proposed to reduce multiple vectors to multiple scalars in a single arithmetic pipeline. The pipeline can be fully utilized by interleaved multiple vector processing.

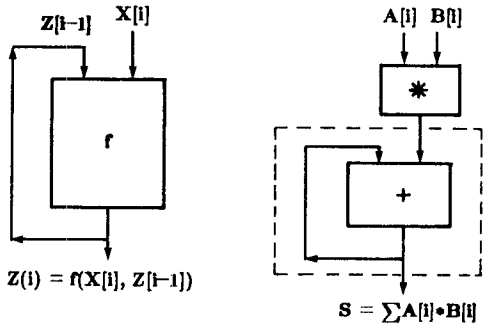
I. INTRODUCTION

Vector reduction arithmetic accepts single vector as input and produces a scalar output. This class of vector operations forms the basis in many scientific computations [Hwan79], [Kogg81], [HwSN81]. Perhaps the most common example is the *Vector Summation*, which computes the sum of all elements, a input vector. This operation is needed in performing the *Inner-Product* among many matrix operations. Other vector reduction operations include the searching for the *Maximum* or the *Minimum Element* and computing the *Mean Value* of the elements of an input vector. *Table 1* presents some vector reduction operations, that one highly demanded in a modern vector supercomputer [HwSu83]. We classify primitive vector operations into four types: $f_1: V_1 \times V_2 \rightarrow V_3$; $f_2: V_1 \times S \rightarrow V_2$; $f_3: V \rightarrow S$; and $f_4: V_1 \rightarrow V_2$. Vector reduction operation belongs to type f_3, which is the only type that produces a scalar output. Many vector computations, such as convolution, matrix multiplication, and discrete Fourier transformation, include vector reduction operations as evaluation primitives [HwCh82], [HwBr83].

A vector reduction function, $f: V \rightarrow S$, can be evaluated recursively either in an array processor or in a pipelined computer. Let X[i] be the i-th element of a vector for $1 \leq i \leq N$. Let Z be the scalar output. We have

$$Z = f(X[1], X[2], \ldots, X[N]) \qquad (1)$$

Let Z[i] be the intermediate result after the first [i] input elements are used in the evaluation of f. A recursive evaluation of f is defined by



(a) A vector reduction processor (b) Vector inner product using reduction arithmetic

Fig. 1. The functional structure of a vector reduction processor.

$$Z[i] = f(X[i], Z[i-1]) \qquad \text{for } 2 \leq i \leq N \qquad (2)$$

where $Z = Z[N]$ and $Z[1] = f(X[1])$.

In a scalar processor, the evaluation of the vector reduction arithmetic involves a DO loop operating on one element at a time as specified in Eq. (2). *Figure 1(a)* shows the input and output specification of a vector reduction processor. For an example, the inner product of two vectors can be carried out by cascading a multiplier with a vector reduction adder shown in *Fig. 1(b)*. Parallelism and pipelining are two major techniques in achieving high performance in arithmetic processor design. Vector arithmetic in an array processor has been studied in [KoSt73], [ChKu75], [HwNi80], [NiHw81]. In the pipelined evaluation of vector reduction arithmetic, a feedback path is enough for data routing and accumulation. This eliminates the use of expensive routing network and excessive intermediate buffer in an SIMD array processor.

Pipelining is made possible by subdividing the function into small segments. The data flow depends on the rate at which new data can be fed into the pipeline. If a function is partitioned into K segments, at most K times speedup can be achieved by the pipeline unit. This peak performance can be achieved, only if the input elements are independent of each other and the input string is sufficiently long. However, due to recurence with feedback inputs, the peak performance can not be achieved in most cases.

The existence of feedback implies a certain degree of sequentialism. Because of this sequentialism, pipelining does not help in the direct implementation of Eq. (2). Equation 2 must be modified in order to appeal to pipelining. Improper control of the feedback around a pipeline can destroy the high throughput. This paper pro-

* This research was supported in part by the U.S. Engineering Foundation under grant RI-A-82-11 at Michigan State University and in part by U.S. National Science Foundation under grant ECS-80-16580 at Purdue University.

poses methodologies to construct and to schedule vector reduction arithmetic pipelines.

A general recurence problem accepts a string of inputs X[1], X[2],...,X[N], and produces an output string Z[1], Z[2],...,Z[N] where each Z[i] is a function of X[i] and of Z[i-1] through Z[i-m] for some m [KoSt73]. In vector reduction operations, only the final term Y = Z[N] is desired. Vector summation and inner product instructions have been implemented in IBM 3838 [IBMc76], TIASC [Step75], STAR-100 [CDCo75], CRAY-1 [Cray77], Cyber-205 [Kasc79], and ESL systolic processor [KuYe82]. This paper provides a generalized pipeline model for the evaluation of vector reduction arithmetic under different scheduling strategies.

We shall first consider the reduction of a single vector into a scalar. The *recursive reduction* method [Kuck78] is reviewed in section 2. The proposed *symmetric vector reduction* and the *asymmetric vector reduction*, methods are described in Sections 3 and 4. These three methods are compared in processing time and memory requirements. Section 5 discusses the situation in which multiple vector instructions are reduced in an arithmetic pipeline. A new scheduling method for multiple vector inputs through a single pipeline is proposed. Different scheduling methods are compared against the number of vectors being processed in an interleaved fashion.

2. RECURSIVE VECTOR REDUCTION

The recursive reduction method is restricted to vectors of length equal to an integer power of 2. The N elements are divided into two halves. N/2 pairs of elements are processed in the first iteration through the pipeline (*Fig. 2.*). The intermediate result vector of N/2 elements

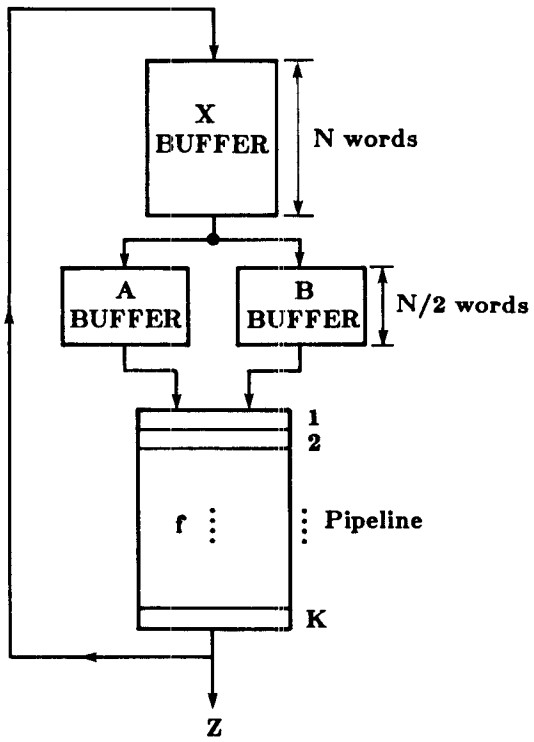


Fig. 2. The hardware organization of a pipelined vector reduction processor for implementing the recursive reduction method.

is again divided into two halves, N/4 elements each, in the second iteration. After $\log_2 N$ iterations, the final scalar result is obtained. Large memory buffers are needed to hold the intermediate results.

A generalized procedure based on the recursive reduction of vector with arbitrary length N is specified below. The input buffer X holds the input elements X[1] through X[N] initially. It will be used later to hold the intermediate results. The two buffers A and B are used to hold the divided subvectors. When N is large, these buffers will significantly increase the hardware cost and time delays.

Algorithm-RR: *Recursive Vector Reduction*
Input: X[1:N]
Output: Z=X[1]+...+X[N]
Procedure
```
M = N;
For i = 1 to ⌈log2N⌉ do
begin
  For j = 1 to ⌊M/2⌋ do
  begin
    A[j] = X[2j-1];
    B[j] = X[2j]
  end
  perform X[i] = f(A[i],B[i]) for 1≤i≤⌊M/2⌋
  if (M mode 2)=1 then X[⌊M/2⌋+1] = X[M];
  M = ⌈M/2⌉
end
```

Let N_i be the number of elements in the X buffer before the i-th iteration, with $N_1 = N$. The input has $\lfloor N_i/2 \rfloor$ pairs of vector elements. The following identities are needed in later proofs.

$$N_i = \lceil N/2^{i-1} \rceil \quad \text{for } i=1 \text{ to } \lceil \log_2 N \rceil \tag{3}$$

and

$$\lfloor N_i/2 \rfloor = \lfloor (N+2^{i-1}-1)/2^i \rfloor = \lceil (N-2^{i-1})/2^i \rceil \tag{4}$$

The i-th iteration requires $(K-1)+\lfloor N_i/2 \rfloor$ pipeline cycles. The total data transfer time between X buffer and A, B buffers equals 2(N-1) cycles, one cycle per element. The total processing time is obtained below in terms of the vector length N and the pipeline size K.

Theorem 1:
The recursive vector reduction method requires $T_r(N,K)$ cycles,

$$T_r(N,K) = \sum_{i=1}^{\lceil \log_2 N \rceil} \left[(K-1) + 3 \left\lceil \frac{N-2^{i-1}}{2^i} \right\rceil \right]$$

$$= (K-1)\lceil \log_2 N \rceil + 3(N-1) \tag{5}$$

Proof of Theorem 1 is rather straightforward. Summing all iterations, N-1 pairs of operands are fed into the pipeline. This method requires to use excessive buffers to hold the intermediate results. Extra time is needed to transfer data between memory and buffers. Two cycles are needed in transferring a pair of elements from the X memory to the A and B buffers. The pipeline utilization will be low due to filling and draining overhead in each iteration.

Two improved vector reduction methods are to be presented, in which the excessive buffer area (A, B) can be eliminated by using a feedback path from the output to the input of the pipeline. The pipeline utilization can be significantly improved in these new vector reduction methods.

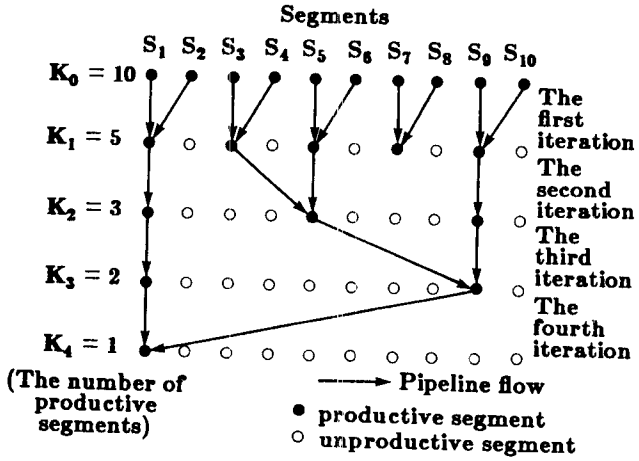


Fig. 3. The pattern of productive segments before and after each pipeline iteration for the symmetric reduction in a pipeline with K=10 segments.

3. SYMMETRIC VECTOR REDUCTION

The hardware organization of a vector reduction pipeline is shown in *Fig. 3*. There are K segments in the pipeline. C is a constant input and B(t) is the feedback input at the t-th pipeline cycle. The feedback input will be latched if e=1. B(t-j) indicates the feedback input at the (t-j)-th cycle. The pipeline is synchronized by a common clock. Each pipeline segment takes one cycle delay. The memory system can supply one vector element per cycle.

The reduction operator, f, is commutative and we assume $N \geq K$. The N elements are partitioned into K groups.

$$Y = f(P(1), P(2),...,P(K)) \tag{6}$$

where

$$P(i) = f(X[N-i+1],X[N-K-i+1],X[N-2K-i+1],...)$$

$$\text{for } 1 \leq i \leq K \tag{7}$$

If (N *mod* K) = 0, then P(i) is the reduction result of N/K elements specified in (7). Otherwise, P(i) is the reduction result of $\lceil N/K \rceil$ elements for $1 \leq i \leq$ (N *mod* K) and of $\lfloor N/K \rfloor$ elements for (N *mod* K) $< i \leq K$. A pipeline segment is *productive* for a given clock period, if the segment is actively involved in the computation during the period. Otherwise the segment is called *unproductive*. This implies that the result generated by an unproductive segment will be ignored.

The number of cycles required to evaluate an input vector, T(N,K), can be divided into four phases.

$$T(N,K) = T_i(N,K) + T_p(N,K) + T_m(N,K) + T_d(N,K) \tag{8}$$

where T_i is the time needed to fill up the pipeline, T_p is the time needed to partition the N elements into K groups, T_m is the time required to merge K groups into one group, and T_e is the time required to drain the pipeline. Since the input elements, X[i]'s, are supplied at the rate of one element per cycle, the following results are obtained.

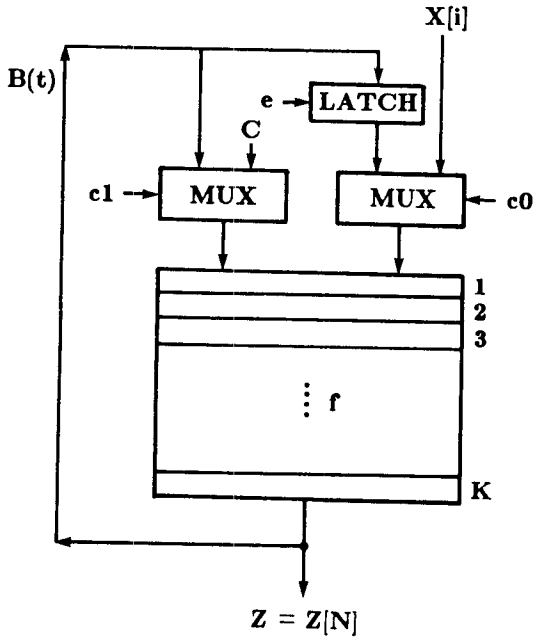
$$T_i(N,K) = \text{Min}\{N,K\} \tag{9}$$

$$T_p(N,K) = \text{Max}\{0,N-K\} \tag{10}$$

The time functions T_i and T_p, add to N cycles. The input elements from local memory or from the output of another pipeline. The control inputs will be (c1,c0,e)=(0,0,0) and (c1,c0,e)=(1,0,0) for the evaluation of Eqs. (9) and (10), respectively. After N cycles, all K segments are productive and the i-th segment will be operating on P(i). The merging of K groups is done by combining two groups at a time. The number of groups is reduced by half after each iteration. Thus, $\lceil \log_2 K \rceil$ iterations result in a single group. In this group-merging phase, we propose a *Symmetric Reduction* (SR) method. All groups have to pass through the pipeline once in each iteration. Groups at two productive segments are merged into a new group as demonstrated in *Fig. 3*. The productive segments are indicated by solid dots.

Figure 3. shows the pattern of productive segments in each iteration for a pipeline of 10 segments, where K_i denote the number of productive segments after the i-th iteration and K_0=K. The first pipeline segment always processes group P(1). The distance between two productive segments is always 2^i after the i-th iteration. The control sequence is illustrated below for merging K′ groups in a pipeline with $K' \leq K$. The following function will be used to indicate whether a given integer is even or odd.

$$e(x) = \begin{cases} 0 & \text{if x is even} \\ 1 & \text{if x is odd.} \end{cases} \tag{11}$$



c1	c0	INPUT PAIR
0	0	C , X[i]
0	1	C , B(t-j)
1	0	B(t) , X[i]
1	1	B(t) , B(t-j)

Fig. 4. The hardware organization of a pipelined vector reduction processor with K segments.

Algorithm-GM: *Symmetric Vector Reduction*
Input: K′ productive segments.
Output: K′ groups are merged into one group with time delay $T_{SR}(K')$.
Procedure:

```
Begin
    t=0;
    for i=1 to ⌈log2K'⌉ do
    begin
      if K' < K then {t=t+1 to t=t+K-K', set
          set (c1,c0,e)=(0,0,0)};
      if (K'-1)mod(2^(i-1)) > 0 then {for t=t+1 to
              t=t+(K'-1)mod(2^(i-1)),
              set (c1,c0,e)=0,0,0,)};
      G=0
      for j=1 to ⌈K'/2^(i-1)⌉ - e(K'/2^(i-1)) do
      begin
        for t=t+1, set (c1,c0,e)=(G,G,G');
        if i>1 then {for t=t+1 to t=t+2^(i-1)-1,
              set (c1,c0,e)=(0,0,0)};
        G=G'
      end;
      for t=t+1, set (c1,c0,e)=(E,1,0)
    end;
    T_SR(K')=t;
End.
```

If N<K, then K' =N; otherwise, K' =K. The above algorithm can be applied to arbitrary value of N and $T_{SR}(N,K)=T_{SR}(K')$. Each iteration requires K cycles, plus a few cycles in the latch if the number of productive segments is odd. *Table 2(a)* shows the contents of successive pipeline segments during the merging of six groups using the SR method. The following lemmas are used in evaluating the time required in the group-merging phase. Lemma 1 can be proved by induction.

Table 1. Representative Vector Reduction Operations

Reduction Arithmetic	Expression
Vector Summation	$S = \sum_{i=1}^{N} X_i$
Search for Maxima	$S = \max\{X_i, j=1,N\}$
Search for Minima	$S = \min\{X_i, i=1,N\}$
Vector Chain Product	$S = \prod_{i=1}^{N} x_i$
Vector Inner Product	$S = \sum_{i=1}^{N} a_i b_i$
Mean Value	$M = \sum_{i=1}^{N} X_i/N$

Lemma 1:

The number of productive segments after the i-th iteration equals

$$K_i = \lceil K_{i-1}/2 \rceil = \lceil K/2^i \rceil \quad (12)$$

Lemma 2:

If K is not an integer power of 2, then

$$\sum_{i=0}^{\lceil \log_2 K \rceil - 1} 2^i \cdot e(K_i) - \sum_{i=0}^{\lceil \log_2 (K+1) \rceil - 1} 2^i \cdot e((K+1)_i) = 1 \quad (13)$$

Proof:

Since K is not a power of 2, K can be written as $K=2^m d$, where $m \geq 0$ and d is an odd number. Thus, $e(K_i)=0$ for $i<m$, $e(K_m)=1$, $e((K+1)_i)=1$ for $i>m$, $e((K+1)_m)=0$, and $(K+1)_{m+1}=\lceil (d+1)/2 \rceil = \lceil d/2 \rceil = K_{m+1}$. From Lemma 1, we have $K_i=(K+1)_i$ for $i>m$. Since K is not an integer power of 2, this implies that $\lceil \log_2 K \rceil = \lceil \log_2 (K+1) \rceil$. Equation (13) is thus proved.

Q.E.D.

Theorem 2

In the group-merging phase of the symmetric reduction method, the total time delay equals

$$T_{SR}(N,K) = \begin{cases} g(K) & \text{if } N \geq K \\ g(N)+(K-N)\cdot\lceil \log_2 N \rceil, & \text{if } N < K \end{cases} \quad (14)$$

where $g(M) \triangleq M \cdot \lceil \log_2 M \rceil + 2^{\lceil \log_2 M \rceil} - M$.

Proof:

By applying mathematical induction and Lemma 2, $\sum_{i=0}^{\lceil \log_2 K \rceil - 1} 2^i \cdot e(K_i) = 2^{\lceil \log_2 \rceil} - K$, if $K \neq 2^k$ for some k. If K_{i-1} is even, then K cycles are needed in the i-th iteration. Otherwise, it takes extra 2^{i-1} cycles to merge with a dummy segment as stated in Algorithm-GM. If N<K, each iteration requires (K-N) additional cycles to cover the delay in dummy segments. Since it takes $\lceil \log_2 N \rceil$ iterations, Eq. 14 is thus proved.

Q.E.D.

Once all groups are merged into one group, K cycles are needed to drain the pipeline in the last phase. The draining delay equals

$$T_d(N,K) = K \quad (15)$$

4. ASYMMETRIC VECTOR REDUCTION

The symmetric reduction method demands extra cycles to complete the operation as shown in *Table 2(a)*. In the second iteration, three groups are merged into two groups. But it takes four more cycles to put P(1) at the first pipeline segment. We describe below an *Asymmetric Reduction* (AR) method, which eliminates those unnecessary delays as illustrated in *Table 2(b)*.

In this AR method, the reduction pipeline needs to record the state of each segment. Denote the state of the latch as S_0 and the states of pipeline segments as $S_1,S_2,...,S_K$. $S_i=1$ indicates that the i-th segment is productive; otherwise, it is 0. The state of the pipeline is expressed by $(S_0,S_1,...,S_K)$. Initially, we have $S_0=0$ and $S_1=S_2=\cdots=S_K=1$, if $K \leq N$, and $S_1=S_2=\cdots=S_N=1$ and $S_{N+1}=\cdots=S_K=0$, if K>N. The group-merging operation is terminated when $S_1=1$ and $S_2=\cdots=S_K=0$.

The control signals are determined by the current states of S_0, S_1, and S_K. *Table 3* shows the partial state transition table and the control sequence for the fast reduction method. The situation of having both S_0 and S_1 equal to 1 will never occur. *Merge* (C,C) means that two dummy inputs are inserted in the pipeline. This will result in an unproductive segment. The next state is determined by the present state as follows:

$$\begin{aligned} S_0 &= S_0 \oplus S_K \\ S_1 &= S_0 S_K \\ S_i &= S_{i-1} \qquad \text{for } 2 \leq i \leq K \end{aligned} \quad (16)$$

The control outputs are determined by

$$\begin{aligned} c1 &= c0 = S_0 S_K \\ e &= \bar{S}_0 S_K \end{aligned} \quad (17)$$

The above equations can be easily modified to cover three other phases. We define a special function $h(M) \triangleq M \cdot \lceil \log_2 M \rceil - 2^{\lceil \log_2 M \rceil} + M$ to be used in the following theorem.

Table 2. Contents of pipeline segments (K=6) of each cycle during the group merging phase.

	1	2	3	4	5	6
		1	2	3	4	5
	5-6		1	2	3	4
		5-6		1	2	3
i=1	3-4		5-6		1	2
		3-4		5-6		1
	1-2		3-4		5-6	
		1-2		3-4		5-6
			1-2		3-4	
				1-2		3-4
	3-6				1-2	
i=2		3-6				1-2
			3-6			
				3-6		
	1-2				3-6	
		1-2				3-6
			1-2			
				1-2		
i=3					1-2	
						1-2
	1-6					

(a) Symmetric Reduction (SR) method.

	1	2	3	4	5	6
		1	2	3	4	5
	5-6		1	2	3	4
		5-6		1	2	3
i=1	3-4		5-6		1	2
		3-4		5-6		1
	1-2		3-4		5-6	
		1-2		3-4		5-6
			1-2		3-4	
				1-2		3-4
	3-6				1-2	
i=2		3-6				1-2
			3-6			
				3-6		
	1-2				3-6	
		1-2				3-6
			1-2			
				1-2		
i=3					1-2	
						1-2
	1-6					

(b) Asymmetric Reduction (AR) method.

Table 3. Partial state transition table and control output of the sequence controller for the asymmetric reduction method.

PRESENT STATE			NEXT STAGE		OUTPUT			COMMENT
S_0	S_1	S_K	S_0	S_1	c1	c0	e	
0	0	0	0	0	0	0	0	merge (C,C)
0	0	1	1	0	0	0	1	merge (C,C), latch B(t)
0	1	0	0	0	0	0	0	merge (C,C)
0	1	1	1	0	0	0	1	merge (C,C), latch B(t)
1	0	0	1	0	0	0	0	merge (C,C)
1	0	1	0	1	1	1	0	merge (B(t),B(t-j))
1	1	0	X	X	X	X	X	never occur
1	1	1	X	X	X	X	X	never occur

Theorem 3:

In a pipeline with K segments, the number of cycles requires in the group-merging phase of the asymmetric reduction method equals

$$T_{AR}(N,K) = \begin{cases} h(K) & \text{if } N \geq K \\ h(N) + (K-N)\lceil \log_2 N \rceil & \text{if } N < K \end{cases} \quad (18)$$

In the AR method, the distance between any two productive segments is no longer a constant. But the distance between the first and the second productive segment is still 2^{i-1} before the i-th iteration. Thus, if K_{i-1} is odd, it takes $K-2^{i-1}$ cycles in the i-th iteration; otherwise, it takes K cycles. Equation 18 thus can be proved similarly to the proof of Theorem 2.

Table 4 shows the total number of cycles required for group-merging for K=1 to 16 and under the SR and the AR methods, respectively. It also shows the number of

Table 4. Number of cycles required in each iteration of the group merging phase for different pipeline sizes.

K	i=1			i=2			i=3			i=4			$T_m(K)$	
	PS	SR	AR	PS	SR	AR	PS	SR	AR	PS	SR	AR	SR	AR
1													0	0
2	1	2	2										2	2
3	2	4	2	1	3	3							7	5
4	2	4	4	1	4	4							8	8
5	3	6	4	2	7	3	1	5	5				18	12
6	3	6	6	2	8	4	1	6	6				20	16
7	4	8	6	2	7	7	1	7	7				22	20
8	4	8	8	2	8	8	1	8	8				24	24
9	5	10	8	3	11	7	2	13	5	1	9	9	43	29
10	5	10	10	3	12	8	2	14	6	1	10	10	46	34
11	6	12	10	3	11	11	2	15	7	1	11	11	49	39
12	6	12	12	3	12	12	2	16	8	1	12	12	52	44
13	7	14	12	4	15	11	2	13	13	1	13	13	55	49
14	7	14	14	4	16	12	2	14	14	1	14	14	58	54
15	8	16	14	4	15	15	2	15	15	1	15	15	61	59
16	8	16	16	4	16	16	2	16	16	1	16	16	64	64

PS: Productive segements after the i-th iteration
SR: Symmetric reduction method
AR: Fast reduction method

cycles required and the number of productive segments in each iteration. Note that the number of cycles required in the SR method is at least K for each iteration; whereas, it is at most K for the AR method.

Precisely, the SR method requires $2(2^{\lceil \log_2 N \rceil}-K)$ more cycles than the AR method, if $N \geq K$. If $K = 2^k$ for some k, the total processing time will be the same for both methods. In the worst case of $K=2^k+1$, the SR method requires 2(K−2) more cycles than the AR method. The number of pipeline segments, K, in a typical vector processor is in the range (2,15). Thus, when N is much greater than K, the difference in processing time between the SR method and the AR method is at most 2(K−2), which is insignificant. The following result indicates the superiority of the proposed SR and AR methods over the recursive reduction method described in Kuck's book.

Theorem 4:

If $N >> K$, the recursive reduction method requires $3N+\mathbf{0}(K \log_2 N)$ pipeline cycles; whereas, the SR or AR method each requires $N+\mathbf{0}(K \log_2 K)$ pipeline cycles to complete. The saving in vector reduction time in these two proposed methods is $2N+\mathbf{0}(K\cdot(\log_2 N-\log_2 K))$.

Theorem 4 summarizes the results obtained in Theorems 1 to 3. This implies that the improvement made by the proposed methods increases linearly with the

vector length. If N is smaller than K, all three methods require approximately $0(K\log_2 N)$ cycles.

A significant advantage of the proposed SR and AR methods over the recursive reduction method lies in the saving of N memory cells (in A and B buffers) for holding the intermediate results. When N is large, this saving is considered significant. This implies also the saving of 2N pipeline cycles for transferring the vector operands from the X memory (usually in the main memory), to the A and B buffers (usually in the cache). All these savings in time and memory make the proposed methods attractive for practical implementation in vector arithmetic processors.

5. MULTIPLE VECTOR PROCESSING

Several vector inputs may independently request the use of a vector reduction pipeline. For example, to find one row of a product matrix from the multiplication of two matrices involves multiple vector inputs and multiple scalar outputs. The scheduling of multiple vector inputs in a single reduction pipeline is studied below. Assuming M independent vectors of N elements each, the scalar result for the j-th vector is defined by

$$Y[j] = f(X[j,1], X[j,2],...,X[j,N]) \quad \text{for } 1 \le j \le M \qquad (19)$$

The memory is assumed to supply one element per pipeline cycle. The simplest method to evaluate Eq. (19) is to process one vector at a time. The total sequential processing time will be $M \cdot T(N,K)$. With careful control, the draining of the last vector may overlap with the filling of the pipeline with the next vector input. Thus, the above computation time can be reduced to be $M \cdot T(N,K) - (M-1) \cdot T_e(N,K)$. Even with this overlapped operations between vectors, the pipeline is still not fully utilized during the group-merging phase. A more efficient method is to interleave the processing of the multiple vector inputs.

Consider the case of M=K first. For interleaved processing, the memory will provide X[1,1], X[2,1],...,X[M,1] at the first M cycles, X[1,2], X[2,2],...,X[M,2] at the next M cycles, and so on. After the first M cycles, the i-th segment will operate with X[M+1-i,1] for $1 \le i \le K$. Another M cycles later, the i-th segment will operate with X[M+1-i] and X[M+1-i,2]. After $M \cdot N$ cycles, the i-th segment will operate with X[M+1-i,1], X[M+1-i,2],..., and X[M+1-i,N]. At the end, K=M more cycles are needed to drain the pipeline. The total processing time will be K(N+1). This approach is considerably faster than processing the vectors sequentially, because no merging phase is needed. Pipeline segments are unproductive only at the filling up phase and at the draining phase. The idea of interleaved processing is to have all pipeline segments shared by as many vectors as possible.

If M>K, not all M vectors can be allocated with pipeline segments. In order to store the remaining M-K vectors, M-K dummy segments are introduced as shown in *Fig. 5*. Since the number of input vectors varies. the length of the dummy segment buffer, D, is adjustable by program control. Virtually, each vector is assigned with one segment for the arithmetic reduction. With D dummy segments inserted, the processor can be viewed as an M-segment pipe. The total processing time will become M(N+1).

If M<K, some vectors may be allocated with more than one segments, and different vectors may use different numbers of segments. In this case, the control of the pipeline will be very difficult because the procedure of merging groups for each vector will be input dependent. Since K is not always an integer multiple of M, one way to allocate vectors with an equal number of segments is to leave some segments idle which results in low pipeline utilization. With the help of a dummy segment buffer, the pipeline can be fully utilized by choosing $D = \lceil K/M \rceil M - K$.

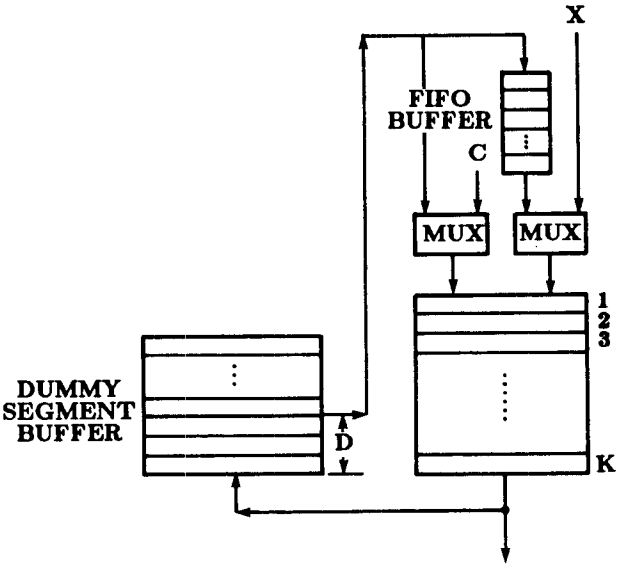


Fig. 5. The hardware organization of a pipelined vector reduction processor with a dummy segment buffer and a FIFO buffer for multiple vector processing.

Let $Q = \lceil K/M \rceil = (D+K)/M$ be the number of segments allocated to each vector including the dummy segments. By feeding the elements into the pipeline in an interleaved fashion all D+K segments will be occupied by groups of vectors in $M \cdot N$ cycles. Each vector has Q groups in Q segments. Since Q>1, a group-merging phase is needed. In *Fig. 4*, a latch was used to hold the group to be merged with the next group. For multiple vector processing, each vector needs its own latch. Thus, a FIFO latch is provided in Fig. 5 for this purpose.

Both the symmetric reduction and the asymmetric reduction method can be used during the group-merging phase. However, the control sequence needs to be modified. The control sequence for each vector is the same as that of single vector processing. The number of iterations will be $\lceil \log_2 Q \rceil$ during the group-merging phase. Since M vectors are processed in an interleaved fashion, each control output must be repeated M times, one for each input vector. The size of the FIFO latch is chosen to be K-1. The total processing time becomes $M \cdot N + Q \cdot T_m(N,Q) + K$. The quantity $T_m(N,Q)$ depends on the reduction method to be used. It was evaluated in Theorem 2 and Theorem 3 for the SR and AR methods respectively.

If the number of required vectors becomes large, the dummy segment buffer may be cost prohibitive. In this case, the vector inputs must be partitioned such that one block of vectors is processed at a time according to the above procedure. In multiple vector processing, the problem of low pipeline utilization due to group merging can be essentially eliminated.

6. CONCLUSIONS

The main contributions in this paper are summarized below in three areas:

- Two vector reduction methods have been proposed using an arithmetic pipeline with feedback connections. These methods eliminate the use of large buffer memory for holding intermediate results.
- The new vector reduction methods result in significant time saving of $2N+\mathbf{0}(K\cdot\log_2 N-K\cdot\log_2 K)$ pipeline cycles and in memory saving of N memory words for intermediate results.
- An interleaved method is proposed for reducing multiple vectors into multiple scalars simultaneously. The pipeline can be fully utilized with maximum speedup in this multiple vector reduction scheme.

References

[ChKu75] Chen, S.C., and Kuck, D.J., "Time and parallel processing bounds for linear recurrence systems," *IEEE Trans. on Computers,* Vol. C-24, July 1975, pp. 701,717.

[CDCo75] Control Data Corp., *Control Data STAR-100 Computer Hardware* Reference Manual, Publication No. 60256000, 1975.

[Cray77] Cray Research Inc., *Cray-1 Computer System Hardware Reference Manual,* Publication No. 2240004, Minnesota, 1977.

[Hwan79] Hwang, K., *Computer Arithmetic: Principles, Architecture, and Designs,* Wiley & Sons, Inc., New York, N.Y., 1979.

[HwNi80] Hwang, K., and Ni, L.M., "Resource Optimization of a Parallel Computer for Multiple Vector Processing," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sept. 1980, pp. 831,836.

[HwSN81] Hwang, K., Su, S.P., and Ni, L.M., "Vector computer architecture and processing techniques," *Advances in Computers,* Vol. 20, M. Yovits (Ed.), Academic Press, Inc., 1981, pp. 115-197.

[HwCh82] Hwang, K., and Cheng, Y.H., "Partitioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Trans. Computers,* Vol. C-31, No. 12, Dec. 1982, pp. 1215-1224.

[HwSu83] Hwang, K., and Su, S.P., "Modeling and scheduling of multiple-pipeline vector supercomputers," *TR-EE 83-2,* School of E.E., Purdue University, Lafayette, Indiana, Jan. 1983.

[HwBr83] Hwang, K., and Briggs, F.A., *Parallel Computer Architecture,* McGraw-Hill Book Co., New York, N.Y. (in press to appar).

[IBMc76] IBM Inc., *IBM 3838 Array Processor Functional Characteristics,* No. 6A24-36390, file no. S370-08, IBM Corp., Endicott, N.Y., October 1976.

[Kasc79] Kascic, M.J., "Vector Processing on the Cyber 200," in *Infotech State of the Art Report,* "Supercomputer," Infotech Int. Ltd., Maidenhead, England, 1979.

[KoSt73] Kogge, P.M., and Stone, H.S., "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. on Computers,* Vol. C-22, August 1973, pp. 786-793.

[Kogg81] Kogge, P.M., *The Architecture of Pipelined Computers,"* McGraw-Hill Book Co., 1981, pp. 139-140.

[Kuck76] Kuck, D.J., *The Structure of Computers and Computations,* Vol. 1, John Wiley & Sons, Inc., 1978, pp. 255-259.

[KuYe82] Kulkarni, A.V., and Yen, D.W.L., "Systolic processing and an implementation for signal and image processing," *IEEE Trans. on Computers,* Vol. C-31, October 1982, pp. 1000-1009.

[NiHw81] Ni, L.M., and Hwang, K., "Performance Modeling of Shared-Resource Array Processors," *IEEE Trans. Software Engineering,* Vol. SE-7, No. 4, July 1981, pp. 386-394.

[Step75] Stephenson, C.M., "Case study of the pipelined arithmetic unit for the TI Advanced Scientific Computer," *Proc. 3rd Ann. Symp. Computer Arith.,* 1975, pp. 168-173.