論文 98-23-8-11

Algorithm-Based Fault Tolerant Vector Convolution on Array Processor

Gi-Yong Song* *Regular Members*

배열프로세서상에서 알고리즘 기반 결함허용 벡터 컨버루션

정회원 송 기 용*

ABSTRACT

An algorithm-based fault tolerant scheme for the vector convolution is proposed employing the positive and negative checksum vectors that are defined in this paper based on the encoder vector. The proposed scheme is implemented on the array processor, and then the amount of redundancy is examined through the complexity analysis.

요 약

본 논문에서는 인코더 벡터(encoder vector)에 입각하여 양, 음 체크썸 벡터(positive, negative checksum vector)를 정의하고, 이를 벡터 컨버루션 (vector convolution)에 적용하여 알고리즘 기반 결함허용 벡터 컨버루션 방식을 제안하였다. 또한 제안된 방식을 배열구조에서 구현하고 복잡도 해석을 통하여 추가 리던던시(redundancy)의 규모를 검토하였다.

I. Introduction

The rapid progress in VLSI technology has reduced the cost of hardware, allowing the application of the multiple copies of low-cost processor to the scientific and engineering problems which require a large amount of computation. For applications involving such a large computational capability, an array processor[1] is a good example of those machines that have been conceived and implemented as a high-performance, low-cost architecture. An array processor is a regular array of simple processing elements that have nearest-neighbor interconnection patterns. The simplicity, modularity and expandability of an array processor make it suitable for area-efficient layouts. In an array processor, communication between processing elements is regular and local, massive parallelism is exhibited, and a relatively low number of I/O operations is required. These properties limit its practical use to that of solving only regular, computebound problems with a high degree of parallelism; those involving vector and matrix operations are typical[2].

In addition to achieving high system performance with the use of multiple copies of identical processor,

* 충북대학교 컴퓨터공학과
論文番號 : 97335-0919
接受日字 : 1997年 9月 19日

it is also important to ensure the correctness of results computed by such complex systems which are extremely prone to transient and intermittent failures. As the number of processing elements increases, the probability of anyone of them failing becomes quite high and, for problems taking a long time to solve, it is rather unlikely to get error-free operation for such a long period. High reliability could be achieved be adopting such a fault tolerant technique as triple modular redundancy[3], quadded logic[4], and recomputing with shifted operand[5]. These techniques have usually been general ones that can be applied at the module level in a system. The generality makes them applicable to most problems, but they also suffer from the requirements of a large amount of overhead. In the context of array processor targeted to perform special purpose computations, it is often possible to derive some special techniques for achieving error detection or fault tolerance concurrently with normal system operation through on-line checks on the results of the computations. The algorithm-based fault tolerance[6-8] is a novel technique to deal with precisely those issues: designing schemes for fault tolerance specific to the algorithms being executed. The method encodes data at a high level, and the algorithms are designed to operate on encoded input data and produce encoded output data. This technique has been applied to vector and matrix computations which form the basis of many computation-intensive tasks. The concept of algorithm-based fault tolerance has also been applied on algorithms for solving such problems as sorting and evaluation of arithmetic expressions and polynomials[9].

In this paper, a fault tolerant vector convolution scheme is proposed on the basis of algorithm-based fault tolerance method. Then, it is implemented on the linear array, and the complexity is analyzed.

Ⅱ. Fault-tolerant Vector Convolution

We need to determine the encoding scheme to implement algorithm-based fault tolerance in a specific algorithm. Based on the concept of an encoder vector [10], in which the information contained in a vector is compressed into a single element and preserved during computations, two types of checksum vectors are defined, and then the fault tolerant vector convolution scheme is proposed on the ground of the checksum vectors defined earlier.

Definition 1 : The *positive checksum vector*, $\overline{\boldsymbol{a}}$, of a vector $\boldsymbol{a}$ having n elements is an n-element vector with each element $\overline{a}$ generated as

$$\overline{a_j} = \sum_{i=0}^{n-1} a_i \quad \text{for } 0 \le j \le n-1$$

Definition 2 : The *negative checksum vector*, $\widetilde{\boldsymbol{a}}$, of a vector $\boldsymbol{a}$ having n elements is an n-element vector with each element $\widetilde{a_j}$ generated as

$$\widetilde{a_j} = \sum_{i=0}^{j} a_i - \sum_{i=j+1}^{n-1} a_i \quad \text{for } 0 \le j \le n-1$$

We will take a brief look at the convolution of vectors[11]. Let a and b be two n-element vectors.

$$\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})^t$$
$$\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})^t$$

The positive folded convolution, denoted by PCON $(\boldsymbol{a}, \boldsymbol{b})$ is a column vector

$$\mathrm{PCON}(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})^t$$

where, for $0 \le m < n$

$$r_m = \sum_{j=0}^{m} a_j b_{m-j} + \sum_{j=m+1}^{n-1} a_j b_{n+m-j}$$

Likewise, the *negative folded* convolution, denoted by

NCON ($\boldsymbol{a}$, $\boldsymbol{b}$) is a column vector

$$\mathrm{NCON}(\boldsymbol{a},\ \boldsymbol{b}) = \boldsymbol{r} = (r_0,\ r_1,\ \ldots,\ r_{n-1})^t$$

where, for $0 \le m < n$

$$r_m = \sum_{j=0}^{m} a_j b_{m-j} - \sum_{j=m+1}^{n-1} a_j b_{n+m-j}$$

Definition 3 : The *convolution sum*, r, is a inner product of the convolution vector and an n-element vector $(1, 1, \ldots, 1)$.

For the positive folded convolution,

$$r = \mathrm{PCON}(\boldsymbol{a},\ \boldsymbol{b})^t \cdot (1,\ 1,\ \ldots,\ 1)$$
$$= \sum_{m=0}^{n-1} \left[\sum_{j=0}^{m} a_j b_{m-j} + \sum_{j=m+1}^{n-1} a_j b_{n+m-j} \right]$$

For the negative folded convolution,

$$r = \mathrm{NCON}(\boldsymbol{a},\ \boldsymbol{b})^t \cdot (1,\ 1,\ \ldots,\ 1)$$
$$= \sum_{m=0}^{n-1} \left[\sum_{j=0}^{m} a_j b_{m-j} - \sum_{j=m+1}^{n-1} a_j b_{n+m-j} \right]$$

Definition 4 : The *convolution checksum*, $\hat{r}$, is a inner product of the vector $\boldsymbol{a}$ and a permutation of the checksum vector, $\bar{\boldsymbol{b}}$ ($\tilde{\boldsymbol{b}}$).

For the positive folded convolution,

$$\hat{r} = (a_0,\ a_1,\ \ldots,\ a_{n-1}) \cdot (\check{b}_{n-1},\ \check{b}_{n-2},\ \ldots,\ \check{b}_0)$$
$$= \sum_{m=0}^{n-1} a_m \check{b}_{n-m-1}$$

For the negative folded convolution,

$$\hat{r} = (a_0,\ a_1,\ \ldots,\ a_{n-1}) \cdot (\check{b}_{n-1},\ \check{b}_{n-2},\ \ldots,\ \check{b}_0)$$
$$= \sum_{m=0}^{n-1} a_m \check{b}_{n-m-1}$$

The fault tolerant positive/negative folded convolution of vector $\boldsymbol{a}$ and $\boldsymbol{b}$ is formally described in Algorithm FT_P/N_CON ($\boldsymbol{a}$, $\boldsymbol{b}$) below.

Algorithm FT_P/N_CON(a,b)

```
1.    if positive
      /* Generation of positive checksum vector */
2.          then for m←0 to n−1
3.                  do for j←0 to n−1
4.                        do b̄_{n−m−1} ← b̄_{n−m−1} + b_j
      /* Compute positive folded convolution */
5.                     for j←0 to m
6.                        do r_m ← r_m + a_j * b_{m−j}
7.                     for  j← m+1 to n−1
8.                        do r_m ← r_m + a_j * b_{n+m−j}
      /* Compute convolution checksum in positive convolution */
9.                     r̂ ← r̂ + a_m b̄_{n−m−1}
      /* Generation of negative checksum vector */
10.         else for m←0 to n−1
11.                 do for j←0 to n−m−1
12.                       do b̌_{n−m−1} ← b̌_{n−m−1} + b_j
13.                    for j← n−m to n−1
14.                       do b̌_{n−m−1} ← b̌_{n−m−1} + b_j
      /* Compute negative folded convolution */
15.                    for j←0 to m
16.                       do r_m ← r_m − a_j * b_{n+m−j}
17.                    for j← m+1 to n−1
18.                       do r_m ← r_m − a_j * b_{n+m−j}
      /* Compute convolution checksum in negative convolution */
19.                    r̂ ← r̂ + a_m b̌_{n−m−1}
      /* Compute convolution sum */
20.   for m←0 to n−1
21.       do r ← r + r_m
22.   if r ≠ r̂ then return("false signal")
```

Each line is self-explanatory. Two quantities, $\hat{r}$, derived from the checksum vector in line 9 (19) and, r, obtained from the convolution output in line 21 are the objects on which the equality test is to be performed. The fault tolerant capability of the proposed scheme is established from the following theorem.

Theorem 1 : Assuming that multiple errors do not cancel one another while the computations proceed,

the FT_P/N_CON ($\boldsymbol{a}$, $\boldsymbol{b}$) can detect any error that may occur during the process of convolution as soon as the convolution output is generated.

Proof : Let $\bar{b}(\check{b})$ be the positive(negative) checksum vector of the input vector $\boldsymbol{b}$. Then the convolution checksum, $\hat{r}$, is the inner product of input vector $\boldsymbol{a}$ and $(\bar{b}_{n-1}, \bar{b}_{n-2}, \ldots, \bar{b}_0)$ in positive convolution or $(\check{b}_{n-1}, \check{b}_{n-2}, \ldots, \bar{b}_0)$ in negative convolution respectively. Let r be the inner product of the convolution output vector and an n-element vector (1, 1, ..., 1), then for the positive folded convolution,

$$
\begin{aligned}
r &= \sum_{m=0}^{n-1} r_m \\
&= \sum_{m=0}^{n-1} \Big[\sum_{j=0}^{m} a_j b_{m-j} + \sum_{j=m+1}^{n-1} a_j b_{n+m-j} \Big] \\
&= \sum_{m=0}^{n-1} a_{n-m-1} \sum_{i=0}^{n-1} b_i \\
&= \sum_{m=0}^{n-1} a_m \bar{b}_{n-m-1} \\
&= (a_0, a_1, \ldots, a_{n-1}) \cdot (\check{b}_{n-1}, \check{b}_{n-2}, \ldots, \bar{b}_0) \\
&= \hat{r}
\end{aligned}
$$

and for the negative folded convolution,

$$
\begin{aligned}
r &= \sum_{m=0}^{n-1} r_m \\
&= \sum_{m=0}^{n-1} \Big[\sum_{j=0}^{m} a_j b_{m-j} - \sum_{j=m+1}^{n-1} a_j b_{n+m-j} \Big] \\
&= \sum_{m=0}^{n-1} a_{n-m-1} \Big[\sum_{i=0}^{m} b_i - \sum_{i=m+1}^{n-1} b_i \Big] \\
&= \sum_{m=0}^{n-1} a_m \check{b}_{n-m-1} \\
&= (a_0, a_1, \ldots, a_{n-1}) \cdot (\check{b}_{n-1}, \check{b}_{n-2}, \ldots, \check{b}_0) \\
&= \hat{r}
\end{aligned}
$$

Therefore, an inconsistency on the values of two quantities, $\hat{r}$ and r, happens if and only if some errors occur on the computations of convolution which do not cancel one another. □

By checking the integrity of computations with two theoretically identical quantities, the correctness of the convolution output is verified as soon as it is produced.

Ⅲ. Implementation and Complexity analysis

The convolutions constitute some of the most compute-intensive tasks in signal or image processing. Though computationally demanding, convolution is nevertheless a highly regular computation. Therefore, by exploiting the regularity inherent to the computation, cost-effective high-throughput array processor structure can be built to perform vector convolution, as argued in some articles[12-14]. The linear array for computing the positive/negative vector convolutions consists of four identical linearly-connected processing elements. as depicted in Figure 1 for $n=4$.

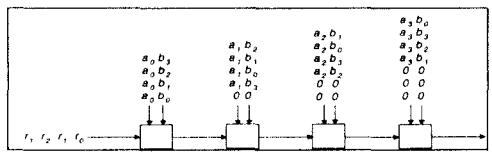

Fig. 1 Linear array for computing vector convolution

The input data are arranged in a staggered form in order to synchronize the flow of output operands with the arrivals of respective input at each entry point. The multiplication of input pairs and the addition of its result to the output operand which flows into the current processing element from left neighbor are performed repetitively until all of the input data are consumed. Once the rightmost processing element has completed operations on its first input pair, an element of the convolution output flows out of the array at each unit delay. The internal structure of processing element is shown in Figure 2.

The adder is to execute addition or subtraction depending on the type of convolution, positive or negative, under the command of master control unit. The

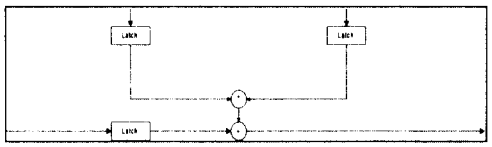

Fig. 2 Cell structure

processing of vector convolution with fault tolerant capability needs generation of checksum vector and operations associated with the correctness verification as well as the functions performed in the original array. Each element of the checksum vector, which is often assumed to be given along with the input data[6], is computed in its corresponding processing element because the assumption that the checksum is given requires a separate preprocessing with extra time overhead of $0(n)$. This requirement of performing normal procedure and computing the checksum concurrently in each processing element needs some extra functional units. Also, an extra processing element is required to perform the correctness checking by comparing outputs from two different data paths. The linear array for vector convolution with fault tolerant capability is shown in Figure 3 for $n=4$.

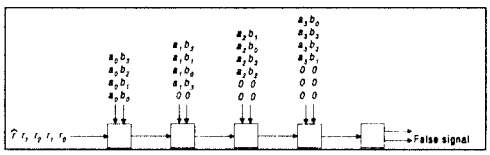

Fig. 3 Linear array for the fault tolerant vector convolution

Let us examine the internal structure of the processing element. All of the processing elements except the rightmost extra one have the same internal structure as is shown in Figure 4.

An adder, multiplexer and checksum buffer are combined with the original structure to accommodate the checksum computation. When positive convolution is involved, the datapath for checksum successively adds the incoming vector element to the accumulated sum to form the positive checksum vector while other part is executing the routine procedure of computing convolution. In the case of negative convolution the datapath repeatedly adds or subtracts the incoming data to or from the accumulated operand, resulting in the negative checksum vector. The internal structure of the rightmost redundant processing element performing the equality test is depicted in Figure 5.

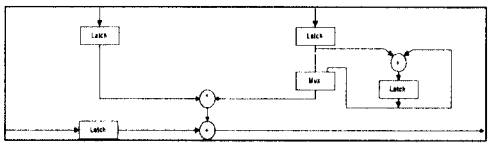

Fig. 4 Cell structure of the fault tolerant array

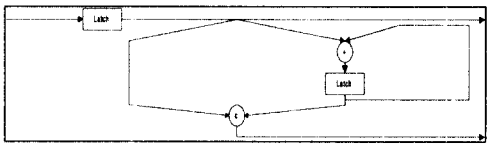

Fig. 5 Redundant cell structure

The equality test is done on two quantities, the accumulated sum of the elements of convolution vector and the incoming checksum-based quantity, to verity the correctness of the convolution outputs. The linear array with fault tolerance requires one more processing element than the original array which needs n processing elements to process the instance of input size of n. The redundancy in the number of processing elements caused by fault tolerance is always one regardless of the instance size. Also, each processing element but the extra one in the array with fault tolerance has more complicated internal structure than its counterpart in the original array because of the hardware requirement for the checksum computation. In order to reflect the real hardware cost, we should take these two factors into account; the extra processing element and modification upon the original processing element. We will adopt k_1, k_2 as a propor-

tionality constants of hardware complexity among three types of the processing elements; one in the original array, one in the array with fault tolerance, and the extra one. Although the exact values of k_i's depend on the design methodology and implementation technology of the processing elements, we will only consider the relative magnitude that can be applied to any case without loss of generality. Let k_1 be the ratio of hardware complexity of processing element in the array with fault tolerance to that of original array, then k_1 becomes slightly larger than 1 taking into account the multiplier unit and the number of buffers in the original array. If we estimate the value of k_1 at circuit level with typical values for the number of transistors in each logic gate, the value of k_1 reaches about 1.17 for the 8-bit data flow path structure with array multiplier adopted. When we denote with k_2 the ratio of the hardware complexity of the extra processing element to that of the original array, k_2 has magnitude far less than 1. This reasoning can be regarded as an appropriate one based on the internal structure of each processing element. This hardware complexity of each case is listed in Table 1.

The constant u represents the hardware complexity of a processing element of the original array. Regarding the time overhead, first, we assume the unit delay, d, which includes logic delay, bus propagation and latch delay, then the unit delay represents the minimum clock cycle time that guarantees proper operation of the array. Since the operations for processing the ordinary procedure and those for computing checksums are performed in fully overlapped manner, there is no time overhead due to the fault tolerant capability except for the delay caused by the extra processing element. The analysis of time complexity is also listed in Table 1.

Table 1. Complexity comparison

	Hardware	Time
Original array	nu	nd
Array with fault tolerance	$(k_1 n + k_2)u$ $(k_1 > 1, k_2 \ll 1)$	$(n+1)\,d$

Ⅳ. Conclusions

The positive and negative folded vector convolution with fault tolerant capability is presented on the ground of algorithm-based fault tolerance method, and implemented on the linear array. The complexity analysis shows that the overheads on hardware resources and time are allowable without requiring excessive redundancy. The correctness of the output convolution vector is verified as soon as it is produced from the array, and in case of the occurrence of an erroneous output, it can cause special routine to start instead of going through successive process that would be performed on the erroneous data otherwise.

References

1. Subrata Dasgupta, *Computer Architecture; A Modern Synthesis, Volume 2: Advanced Topics*, pp 242-246 John Wiley & Sons, New York, 1989.
2. P.Banerjee, J.T.Rahmeh, C.Stunkel, V.S.Nair, K.Roy, V.Balasubramanian, and J.A.Abraham, "Algorithm-based fault tolerance on a hypercube multiprocessor," *IEEE Trans. Comput.*, Vol 39, pp 1132-1145. September 1990.
3. A.Avizienis, "Fault-tolerance: The Survival Attribute of Digital Systems," *Proc. IEEE*, Vol.66, pp.1109-1125, October 1978.
4. J.G.Tryon, 'Quadded logic', in *Redundancy Techniques for Computing Systems*. Wilcox and Mann, eds., Washington, D. C.: Spartan Books. pp.205-228, 1962.
5. J.H.Patel and L.Y.Fung. "Concurrent Error Detection in ALUs by Recomputing with Shifted Operands," *IEEE Trans. Computers*, Vol.C-31, pp.589-595, July 1982.
6. K.-H.K.Huang and J.A.Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Trans. Computers*, Vol.C-33, No.6, pp.518-528,

June 1984.

7. A.Roy-Chowdlhury and P.Banerjee, "Tolerance Determination for Algorithm-Based Checks Using Simplified Error Analysis Techniques," *Proc. of 1993 International Symposium on Fault-Tolerant Computing: FTCS-23*, pp.290-298, June 1993.
8. F.T.Assaad and S.Dutt, "More Robust Tests in Algorithm-Based Fault-Tolerant Matrix Multiplication," *Proc. of 1992 International Symposium on Fault-Tolerant Computing: FTCS-11*, pp.430-439, July 1992.
9. P.Banerjee and J.A.Abraham, "Fault-secure algorithms for multiple processor systems." in *Proc. 11th Int. Symp. Comput.* Architect., Ann Arbor, MI, June 1984, pp.279-287.
10. V.S.S.Nair and J.A.Abraham, "General Linear Codes for Fault Tolerant Matrix Operations on Processor Arrays," in *Proc. Int. Symp. Fault-Tolerant Comput.*, Tokyo, pp.180-185. June 1988.
11. S.Lakshmivarahan and S.K.Dhall, *Analysis and Design of Parallel Algorithms*, pp.195-198. McGraw-Hill Book Co., 1990.
12. H.T.Kung, "Why Systolic Architectures?", *Computer* Vol.15, No.1, pp.37-46, January 1982.
13. J.M.Kim and S.M.Reddy, "Easily Testable and Reconfigurable Two-dimensional Systolic Arrays," *Computer System Science and Engineering*, Vol.7, No.3, pp.160-169, 1992.
14. H.T.Kung, L.M.Ruane, and D.W.L.Yen, "A Two-level Pipelined Systolic Array for Convolutions," in *VLSI Systems and Computations*, H.T.Kung, R.F.Sproull, and G.L.Steele, Jr.(eds.), Carnegie-Mellon University, Computer Science Press, Oct. 1981, pp.255-264.

송 기 용(Gi-Yong Song) 정회원
1974년~1980년 : 서울대, 동대학원(전자 공학),
1995년 : Univ. of Southwestern Louisiana전산공학박사,
1983년~현재 : 충북대학교 공대 컴퓨터공학과 재직중.
※주관심분야 : 결함허용 Computing, 컴퓨터 구조, 논리설계등