ISSN: 0128-7680
Pertanika J. Sci. & Technol. **20 (1)**: 89 – 96 (2012)

Exact Parallel Plurality Voting Algorithm for Totally Ordered Object Space Fault-Tolerant Systems

Abbas Karimi[1*], Faraneh Zarafshan[1], Adznan Jantan[2], Abdul Rahman Ramli[2], M. Iqbal b. Saripan[2] and S. A. R. Al-Haddad[2]

[1]Department of Computer Engineering, Faculty of Engineering, Arak Branch, Islamic Azad University, Arak, Iran.

[2]Department of Computer and Communication Systems Engineering, Faculty of Engineering, Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia

[*]E-mail: A-karimi@iau-arak.ac.ir

ABSTRACT

Plurality voter is one of the commonest voting methods for decision making in highly-reliable applications in which the reliability and safety of the system is critical. To resolve the problem associated with sequential plurality voter in dealing with large number of inputs, this paper introduces a new generation of plurality voter based on parallel algorithms. Since parallel algorithms normally have high processing speed and are especially appropriate for large scale systems, they are therefore used to achieve a new parallel plurality voting algorithm by using *(n/log n)* processors on EREW shared-memory PRAM. The asymptotic analysis of the new proposed algorithm has demonstrated that it has a time complexity of *O(log n)* which is less than time complexity of sequential plurality algorithm, i.e. Ω *(n log n)*.

Keywords: Fault-tolerant, Parallel Algorithm, PRAM, Voting Algorithm

INTRODUCTION

Fault-tolerance is the knowledge of manufacturing the computing systems which are able to function properly even in the presence of faults. These systems compromise a wide range of applications, such as embedded real-time systems, commercial interaction systems and e-commerce systems, Ad-hoc networks, transportation (including rail-way, aircrafts and automobiles), nuclear power plants, aerospace and military systems, and industrial environments in all of which, a precise inspection or correctness validation of the operations must occur (e.g. where poisonous or flammable materials are kept) (Latif-Shabgahi *et al*., 2004). In these systems, the aims are to decrease the probability of the system hazardous behaviour and to keep the systems functioning even in the occurrence of one or more faults.

Redundancy is one of the important methods in achieving fault-tolerance and it can be implemented in three forms, including static (fault masking methods), dynamic (fault detection, fault diagnosis, fault isolation and fault location), and hybrid (masking faults and fault detection and location).

The aim of static redundancy is masking the effect of fault in the output of the system. *N*-Modular Redundancy (NMR) and *N*-Version Programming (NVP) are two principal methods of static redundancy in hardware and software, respectively. Three modular redundancy (TMR) is

Received: 1 October 2010
Accepted: 4 March 2011
[*]Corresponding Author

the simplest form of NMR which is formed from N=3 redundant modules and a voter unit which arbitrates among the modules' outputs (*Fig. 1*).

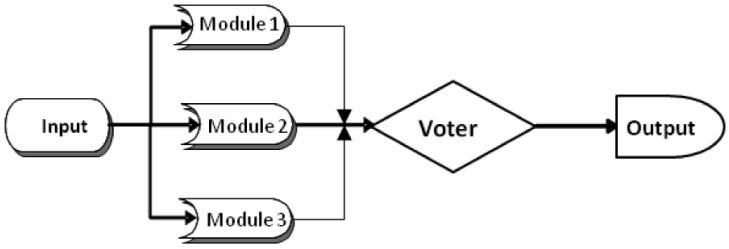

Fig. 1: TMR System

Voter performs a voting algorithm in order to arbitrate among different outputs of redundant modules or versions and to mask the effect of fault(s) from the system output. Many voting algorithms have been defined in the literature, each with particular strengths and weaknesses that make a particular voting algorithm more proper than the other one in a given application.

Plurality voter is one of several voting algorithms which is applied in the fault-tolerant control systems. The main advantages of this voter are high reliability (Blough *et al*., 1990) and high availability (Yacoub *et al*., 2002), in comparison with other popular voting algorithms. On contrary to some severe voters like majority and its extended forms, such as smoothing and predictive, plurality voter can operate flexibly if less than majority elements are in agreement (Latif-Shabgahi *et al*., 2004).

In this paper, the parallel algorithms are used on the EREW shared-memory systems to present a new generation of voter – and this is known as Parallel Plurality Voter (PPV) – which provides the plurality voter extension without enlarging the calculations, and is suitable for large-scale systems and with optimal processing time.

The current paper is organized in the following manner. In the next section, background and related works are described, whereas the sequential and parallel plurality voting algorithms are presented in later sections. Result and discussion section deals with the performance analysis of the new algorithm and its comparison with the sequential algorithm. Finally, the conclusions and future works are explained.

BACKGROUND AND RELATED WORKS

Voting algorithms have been extensively applied in situations where choosing an accurate result out of the outputs of several redundant modules is required. Generalized voters including majority, plurality, median and weighted average were first introduced by Lorczak *et al* (1989).

Majority voter is perhaps the most applicable voter that produces an output among n variant results, where at least $\lfloor (n+1)/2 \rfloor$ variant results agree (Latif-Shabgahi *et al*., 2004). Meanwhile, plurality voter is a relax form of majority voter, in which even if less than $\lfloor (n+1)/2 \rfloor$ variant results are in agreement, voter can report consensus. Thus, plurality and majority are actually extended forms of m-out-of-n voting, in which at least m modules out of n modules should be in agreement; otherwise, voter cannot produce the output. The m-out-of-n voting method is a suitable choice for the systems where the number of the voter inputs is large. The other generalized voter is the median voter that always chooses the mid-value of voter inputs as the system output. The most significant limitation of this particular algorithm is that the number of the voter inputs is assumed

to be odd (Lorczak *et al*., 1989). In the weighted average algorithm, the weighted mean of the input values is calculated as the voting result. The weight value is assigned to each voter input in various methods (Latif-Shabgahi, 2004; Latif-Shabgahi *et al*., 2003; Lorczak *et al*., 1989; Tong *et al*., 1991), and then, the calculated weights, w_i, are used to provide the voter output, $y=\sum w_i.x_i/\sum w_i$, where x_i refers to each voter inputs and y is the voter output. Average voter is a special case of the weighted average voter in which all the weights are assumed to be equal to *1/n*. Two latest methods amalgamate variant results to make the output (Latif-Shabgahi *et al*., 2004); as a result, the output may be clearly different from the input values. Although this feature is useful in such applications as image processing filters (to merge the results of the adjacent pixels), in many safety critical and high reliable systems the voters like majority, plurality, and median are preferred, particularly due to their selecting strategy.

Based on the type of agreement, Plurality voter can be exact or inexact. In the exact voting, agreement achievement requires that the redundant results to be exactly the same, while in the inexact voting, agreement means that the multiple results might be different. Nevertheless, their difference from each other is smaller than a predefined, application specific threshold (Latif-Shabgahi *et al*., 2004). Although inexact voting is closer to real conditions, to avoid the complexities due to selecting threshold, and to simply design parallel algorithm, the research is limited to the exact plurality voter.

The other important issue about all the above mentioned voters, including plurality, is their dependency on the structure of the input space (Parhami, 1992, 1994). Hence, while the number of voter inputs increases, there will be increases in the complexity of calculations, in addition to harmful effects on the speed of processing in the control system.

To address the problems of plurality voter mentioned, an effective parallel plurality algorithm was proposed based on the shared memory EREW by using the parallel algorithms.

PARALLEL ALGORITHM IN A SHAREDMEMORY SYSTEM

So far, parallel voters have been taken into account in the studies by several authors (Karimi *et al*., 2010; Lei *et al*., 1993; Parhami, 1996). In addition, an optimal parallel average voter has been designed and analyzed in Karimi *et al*. (2010). It has also been demonstrated that the time complexity of the mentioned algorithm is *O(log n)* with *n/log n* processors in an EREW shared memory system. In Lei *et al*. (1993), an efficient parallel algorithm was proposed to find the majority element in shared-memory and message passing parallel systems and its time complexity was determined, while an approach for the parallelized *m*-out-of-*n* voting through divide-and-conquer strategy was presented and analyzed in Parhami (1996).

Employing parallel algorithms is of the beneficial techniques to address the problems of sequential plurality voter for large object space applications such as public health systems, geographical information systems, Data Fusion, Mobile Robots, Sensor Networks, etc. Meanwhile, conducting large calculative operations using parallel algorithms is faster due to the modern processors, compared to when such operations are conducted by sequential algorithm. Moreover, multiple processing resources typically available in the applications dealing with a large number of inputs can be utilized for performing parallel voting algorithm (Parhami, 1996).

In this section, a new parallel plurality voting algorithm is presented on EREW shared-memory systems. First, sequential plurality voting is introduced and this is followed by introducing and describing the parallel plurality algorithm with inspirations from the functions of this algorithm and using divide-and-conquer method and Brent's theorem. Divide-and-conquer strategy and Brent's theorem are used in the designation and the optimization of the parallel plurality voting algorithm, respectively. To compare the new algorithm with plurality voter in this research, the asymptotic analysis was employed.

Sequential Plurality Voting

The algorithm for the sequential plurality voter in totally ordered space for *n* inputs is presented in Parhami (1994), as follows:

Procedure Exact Sequential Plurality Voting (Totally Ordered)

Sort (ascending order) in place the set of records (x_i, v_i) with x_i as key. Use the end –marker $(x_{n+1}, v_{n+1}) = (\infty, 0)$.

$y:=z:=x_1;$

$u:=w:=v_1;$

For $i=2$ ***to*** $n+1$ ***do***

While $x_i=z$ ***do***

$u:=u+v_i;$
$i:=i+1;$

End While.

If $u>w$ ***Then***

$w:=u;$
$y:=z;$

End If.

$z:=x_i;$
$u:=v_i;$

End For.

End.//*end of procedure*//

Parallel Plurality Voting

Basically, there are two architectures for the multi-processor systems. One is the shared-memory multi-processor system and the other is message passing (Lei *et al*., 1993). In a shared-memory parallel system, it is assumed that *n* processor has either shared the public working space or has a common public memory.

To present the parallel plurality voting (PPV) in the PRAM machines with EREW shared-memory technology, the following assumptions are taken into account:

- Array A [1...n] with n elements, comprises $a_1, a_2, \ldots, a_n$,where each a_i , i=1...n, is the output of i^{th} module.
- The number of redundant modules, n, is considered as the power of 2.
- Array A is divided to $p= (n/\log n)$ sub-arrays each of which contains at most $\log n$ element.
- Divide-and-conquer method is used to implement the algorithm.
- For enhancing the algorithm, the number of required processors is assumed equal to the number of sub-arrays i.e. p.

The pseudo-code of our optimal parallel plurality voter (PPV) is presented as follows:

Pseudo-code of parallel plurality voter

Procedure Exact Parallel-Plurality Voting *(Totally Ordered Space)*

Input: *X* is an array of the *n* elements $x_1,x_2,\ldots,x_n$ where $n=2^k$.
Output: Return *Y* as the output of the Parallel Plurality Voting (PPV).

Step 1. // Partitioning *X* //

X is subdivided into $p = \frac{n}{\log n}$ subsequence's X_i of length *log n* each, where $1 \le i \le p$;

Step 2. *// Find the number of iterative elements (k) in each partition //*

(2.1) $k \leftarrow 1$

(2.2) ***For*** *i=1* ***to*** *p* ***do in Parallel***

$Z_i \leftarrow X[(i-1)\log n + 1]$
$u_i \leftarrow 1$ *// set tally for each element //*
For $j=((i-1)\log n + 2)$ ***To*** Min$\{i*\log n, n\}$ ***Do***
If $X_j=Z_i$ ***Then***
Find iterative Elements in each partition.
$u_i \leftarrow u_i + 1$ // update tally //
Else
$k \leftarrow k+1$
$Z_k \leftarrow X_j$
$u_k \leftarrow v_j$
End If.
End For.
End For.

Step 3.

(3.1) ***IF*** *k=n* ***Then*** *// n is the number of elements //*
"There is no plurality agreement"
Else
Z is subdivided into $p = \frac{k}{\log k}$ subsequence's Z_i of
Length *(log k)* each, where $1 \le i \le p$;

(3.2) **Broadcast** Z_i to processor P_i

(3.3) $L \leftarrow 1$
// Find Repetitive Elements in other partitions and update its tally //

(3.4) ***For*** *i=0* ***to*** $\lceil \log p - 1 \rceil$ ***do***
For *j=1* ***to*** *p* ***do in Parallel***
IF *(j is Odd)* ***Then***
$T_j \leftarrow Z[(j-1)*\log k + 1]$
$\forall S \in \{((j-1)*\log k + 2)$ **and** $Min\ \{2(i+j)*\log k,(j-1)k\}\}$
Merge each of pair subsequence (T_j, T_{j+1}) *and* ***do in Parallel.***
IF $T_j=Z_s$ ***Then***
$u_j \leftarrow u_j + 1$

Else

$W(0,L) \leftarrow T_j$; $W(1,L) \leftarrow u_j$;
$L \leftarrow L+1$; $T_j=Z_s$; $u_j \leftarrow u_s$

End IF.

End IF.

End for.

End for.

// Matrix W is an amalgamation of array X and the number of each element iteration(tally) //

Step4.

(4.1) ***For*** $i=1$ ***to*** $(L/\log L)$ ***do in Parallel***

$Y(0,i) \leftarrow W(0, (i-1)*\log L +1)$
$Y(1,i) \leftarrow W(1, (i-1)*\log L +1)$

For $j=((i-1)*\log L +2)$ ***To*** $Min\{i*\log L, L\}$ ***do***

If $Y(1,j) > Y(1,i)$ ***Then***

$Y(1,i) \leftarrow Y(1,j)$

$Y(0,i) \leftarrow Y(0,j)$

End If.

End For.

End For.

// Return the Result if PPV exists //

(4.2) ***For*** $i=0$ ***to*** $\left\lceil \log\left(\frac{L}{\log L}\right)\right\rceil - 1$ ***do***

For $j=1$ ***to*** $(L/\log L)$ ***do in Parallel***

If *(j is Odd)* ***Then***

If $Y(1,(j-1)*\log L +1)) > Y(1,(j*\log L +1))$ ***Then***

$Result \leftarrow Y(0,(j-1)*\log L +1))$

Else

$Result \leftarrow Y(0,(j*\log L +1))$

End IF.

End IF.

End For.

End For.

End.

In next section, the four steps of the algorithm PPV are examined and compared with the time complexity of the sequential algorithm given in Sequential Plurality Voting section. In order to define an optimal parallel plurality voting with less time complexity and minimal number of processors, the Brent's theorem (Brent, 1973) was used to partition the input elements into predetermined groups in the PRAM systems with *p* processor.

RESULTS AND DISCUSSION

In this section, both the parallel and sequential plurality voting algorithms introduced in previous sections were analyzed by computing the complexity of the algorithms and discussions on each step in order to highlight the efficiency of the new parallel algorithm. Hence, to do so, $T_s(n)$ is

defined as the function of executing time of the sequential plurality voting algorithm and $T_p(n)$ as the function of the executing time of the parallel voting algorithm in which p is the number of the processors. As mentioned in sub-section of Sequential Plurality Voting, the sequential plurality voter needs the time complexity $T_s(n)=O(n \log n)$, while parallel algorithm consumes the constant time of $O(1)$ to divide array X into p sub-arrays (*partitioning X*) having the maximal length of $(\log n)$ which has been indicated in step 1 of PPV.

In order to find the number of iterative elements in array X, i.e. the value of counter k in each partition, p, there is a need to time complexity of $O(\log n)$ in step 2. In step 3, if value k is equivalent to the number of inputs of array X, the algorithm will no longer have any result because it does not achieve plurality agreement. The time complexity of step 3.1 is $O(1)$, other than if the value k is opposite to n, the number of the countered elements is arranged, which are stored as matrix Z in step 2, into p partitions by using the approach of step 1. The time complexities of the procedures in steps 3.2, 3.3 and 3.4 are respectively $O(1)$, $O(1)$ and $O\left(\log \frac{k}{\log k}\right)$ or $O(\log k)$, which is smaller than or equal to $O(\log n)$.

The result of step 3 is matrix W in which row zero refers to the *value* of algorithm's input, and the first row represents the *tally* of each element in the partition. In step 4, the later row in several partitions should be compared to find the iterative elements (step 4.1). Finally, the plurality element in step 4.2 is possibly considered. The procedures in step 4 totally require $O(\log l)<O(\log k)<O(\log n)$, and therefore:

$$T_p(n)= O(1)+O(\log n)+ O(\log k)+ O(\log l). \quad (1)$$

It is concluded that the totally complexity of the parallel plurality voting for $1 \leq k \leq n$ is:

$$T_p(n)=O(\log n). \quad (2)$$

Since the execution time of the parallel plurality voter is $O(\log n)$, it is able to run faster than the sequential plurality voter with the complexity of $\Omega(n \log n)$. Furthermore, it can be obviously seen that the total number of the required processors in the parallel algorithm does not exceed $(n/\log n)$. Hence, taking into account the execution time and number of processors needed, the cost and time complexity of the proposed algorithm are optimal.

CONCLUSIONS

In this paper, an optimal parallel algorithm has been proposed to find the plurality agreement among the results of n redundant modules in the parallel shared-memory systems in the EREW model. As seen in results and discussion, the execution time of the sequential algorithm is $\Omega(n \log n)$, whereas it is $O(\log n)$ in the proposed parallel algorithm. Hence, the time complexity of the parallel plurality voter is less than its sequential peer.

The parallel plurality voter has also been shown to resolve the problem associated with sequential plurality voter in dealing with a large number of inputs. This parallel algorithm can be extended to unordered arrays that are implemented in future on Bus, Hyper Cube and Mesh typologies in the message passing systems.

REFERENCES

Blough, D. M., & Sullivan, G. F. (1990). *A Comparison of Voting Strategies for Fault-tolerant Distributed Systems.* Paper presented at the Ninth Symposium on Reliable Distributed Systems.

Brent, R. P. (1973). *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ.: Prentice-Hall.

Karimi, A., Zarafshan, F., Jantan, A. B., Ramli, A. R. B., & Saripan, M. I. B. (2010). *An Optimal Parallel Average Voting for Fault-tolerant Control Systems.* Paper presented at the 2010 International Conference onNetworking and Information Technology (ICNIT).

Latif-Shabgahi, G. (2004). A Novel Algorithm for Weighted Average Voting Used in Fault-Tolerant Computing Systems. *Microprocessors and Microsystems, 28*(7), 357-361.

Latif-Shabgahi, G., Bass, J. M., & Bennett, S. (2004). A Taxonomy for Software Voting Algorithms Used in Safety-Critical Systems. *IEEE Transactions on Reliability, 53*(3), 319- 328.

Latif-Shabgahi, G., Hirst, A. J., & Bennett, S. (2003). *A Novel Family of Weighted Average Voters for Fault Tolerant Computer Systems.* Paper presented at the Proceedings of ECC03: European Control Conference, Cambridge, UK.

Lei, C. L., & Liaw, H. T. (1993). Efficient Parallel Algorithms for Finding the Majority Element. *Journal of Information Sceince and Engineering, 9*, 319-334.

Lorczak , P. R., Caglayan, A. K., & Eckhardt, D. E. (1989). *A Theoretical Investigation of Generalized Voters for Redundant Systems.* Paper presented at the FTCS-19. Digest of Papers. Nineteenth International Symposium on Fault-Tolerant Computing, chicago, USA.

Parhami, B. (1992). *Optimal Algorithms for Exact, Inexact and Approval Voting*. Paper presented at the 22nd International Symposium on Fault-Tolearnt Computing (FTCS-22), Boston, N.A, USA.

Parhami, B. (1994). Voting Algorithms. *IEEE Transactions on Reliability, 43*, 617-629.

Parhami, B. (1996). Parallel Threshold Voting. *The Computer Journal, 39*(8), 692-700.

Tong, Z., & Kain, R. Y. (1991). Vote Assignments in Weighted Voting Mechanisms. *IEEE Transactions on Computers, 40*(5), 664-667.

Yacoub, S., Lin, X., & Burns, J. (2002). Analysis of the Reliability and Behavior of Majority and Plurality Voting Systems. Palo Alto: Hewlett-Packard Company.