Enhancing Efficiency of Software Fault Tolerance Techniques in Satellite Motion System

Hoda Banki
Department of Electrical and Computer Engineering, University of Kashan, Isfahan, Iran
banki@grad.kashanu.ac.ir
Seyed Morteza Babamir
Department of Electrical and Computer Engineering, University of Kashan, Isfahan, Iran
babamir@kashanu.ac.ir
Azam Farokh
Department of Electrical and Computer Engineering, University of Kashan, Isfahan, Iran
farokh@grad.kashanu.ac.ir
Mohammad Mehdi Morovati*
Department of Electrical and Computer Engineering, University of Kashan, Isfahan, Iran
morovati@grad.kashanu.ac.ir

Received: 02/Oct/2012 Revised: 07/Jun/2014 Accepted: 02/Aug/2014

Abstract
This research shows the influence of using multi-core architecture to reduce the execution time and thus increase performance of some software fault tolerance techniques. According to superiority of N-version Programming and Consensus Recovery Block techniques in comparison with other software fault tolerance techniques, implementations were performed based on these two methods. Finally, the comparison between the two methods listed above showed that the Consensus Recovery Block is more reliable. Therefore, in order to improve the performance of this technique, we propose a technique named Improved Consensus Recovery Block technique. In this research, satellite motion system which known as a scientific computing system is consider as a base for our experiments. Because of existing any error in calculation of system may result in defeat in system totally, it shouldn't contains any error. Also the execution time of system must be acceptable. In our proposed technique, not only performance is higher than the performance of consensus recovery block technique, but also the reliability of our proposed technique is equal to the reliability of consensus recovery block technique. The improvement of performance is based on multi-core architecture where each version of software key units is executed by one core. As a result, by parallel execution of versions, execution time is reduced and performance is improved.

Keywords: Software Fault Tolerance; Multi-core; Parallel Execution; Consensus Recovery Block; N-version Programing; Acceptance Test.

1. Introduction

Nowadays the influence of software on different domains such as economics, medicine, aerospace and so on is quite sensible. One of the main requirements of these systems is safety and reliability of software. According to the importance of software reliability, demand for using fault tolerance techniques in software development have increased significantly. Design diversity is one of the fault tolerance methods which needs to run multiple versions of the program [1]. software fault tolerance techniques increase software reliability, on the other hand by increasing number of versions of the program, execution time increases at the same time and this will reduce the performance. by taking advantages of distributed and parallel processing systems, the efficiency is increased and thus the cost of using these systems will be acceptable. Using the multi-core architecture is a good idea for taking advantage of parallel processing.

Based on the idea of software fault tolerance, for some software key units in a system, N versions can be developed separately with similar functionality [2]. The purpose of design diversity is constructing independent modules as many as possible and minimizing occurrence of identical errors in these modules [3]. All versions are executed with identical initial conditions and inputs. Output of all versions is given to a decision module and the decision module selects a unique result as a correct output.

The paper continues as follow: section 2 introduces N-version programming and recovery block and their derivative techniques. Section 3, introduces satellite motion system as a case study. Section 4, discusses the usage of multi-core architecture in fault tolerance techniques. Implementation results are reviewed in Section 5 .the proposed method is presented in Section 6 and finally in Section 7 conclusions are discussed.

2. Software Fault-Tolerance Techniques

In this section some fault tolerance techniques are introduced.

* Corresponding Author

2.1 N-version programing technique

Using different algorithms and designs, Most program functions can be performed in various ways. A program version denoting a separate implementation of a program function is called a variant. Each variant has a varying degree of efficiency in terms of memory management and utilization, execution time, reliability, and other criteria.

N-version programming (NVP) technique is one of the main techniques of software fault tolerance. In this technique, N different versions of a module are implemented and executed concurrently (simultaneously). Then the results will be presented to a decision module and this module selects the correct result [3]. The decision module examines the results and selects the “best” one if exists. There are other available alternative decision mechanisms. For example one decision mechanisms is majority voter. The NVP algorithm technique is shown in Fig. 1.

```
run Version 1, Version 2, …, Version n
if (Decision Mechanism (Result 1, Result 2, …, Result n)
        return Result
else failure exception
```

Fig. 1. N-version programming technique algorithm

Other augmentations, enhancements, and combinations have been made to the NVP techniques. These are typically given an entirely new name rather than being called an extension to the NVP technique. Some of these techniques are described in the following.

2.2 N-version programing-Tie broker technique

In order to improve the performance of NVP technique, N-version programming-Tie Broker (NVP-TB) technique has been developed whose strategy is to synchronize the versions. In this technique, assuming that three versions of software key unit are developed, when the results of two faster versions are produced, it does not wait for the slowest version anymore. In other words, when the two faster versions, complete their execution, their results will be compared and one of the results is returned as a correct result if they match, otherwise, it waits for the result of slowest version and then the correct result is determined by decision mechanism [4]. The algorithm of this technique is represented in Fig. 2.

```
run Version 1, Version 2, Version 3
if (Comparator (Fastest Result 1, Fastest Result 2) )
        return Result
else Wait ( Last Result)
        if (Voter (Fastest Result 1, Fastest Result 2, Last Result) )
            return Result
else error
```

Fig. 2. N-version programming-Tie broker technique algorithm

2.3 N-version programing-Acceptance test technique

To reduce the probability of selecting an incorrect result, Tai and his colleagues added an acceptance test to the NVP technique. In this technique, after the decision mechanism selects one of the results as the correct one, this result is passed to the acceptance test for checking its correctness in order to increase the reliability [4]. The N-version programing-Acceptance test technique is represented in Fig. 3.

```
run Version 1, Version 2, Version 3
if (Voter (Result 1, Result 2, Result 3) )
        If (Acceptance Test (Result))
                return Result
else error
```

Fig. 3. N-version programming-Acceptance test technique algorithm

2.4 N-version programing-Tie broker- Acceptance test technique

Because the two modified NVP techniques are complementary, N-version programming-Tie Broker-acceptance test (NVP-TB-AT) technique has been developed to concentrate on both reliability and performance. Actually, this technique is a combination of NVP-TB technique and acceptance test. Acceptance test is used to increase the reliability which will cause the execution time to increase and thus the performance will be reduced. But by using the Tie-broker technique, reduction of performance is compensated. As a result, not only this technique has higher performance than NVP-AT, but also has reliability equal to NVP-AT[5]. The N-version programing-Tie broker Acceptance test is explained in Fig. 4.

```
run Version 1, Version 2, Version 3
if (Comparator (Fastest Result 1, Fastest Result 2) )
        return Result
else Wait ( Last Result)
        if (Voter (Fastest Result 1, Fastest Result 2, Last Result) )
            If (Acceptance Test (Result))
                return Result
else error
```

Fig. 4. N-version programming-Tie broker-Acceptance test technique algorithm

2.5 Recovery block technique

Recovery block (RcB) technique is one of the main techniques of software fault tolerance. This technique works in a way that different versions are prioritized in order of their importance; then they is run in order of their preferences. In other words, RcB incorporates these variants such that the most efficient module is located first in the series, and is called the primary alternate or primary try block. Acceptance or rejection of each version is identified by acceptance test module. At first, the overall situation of system is stored. if no versions can successfully pass the acceptance test, the system is returned to the saved state and then the next module will run [3]. If no alternates are successful, an error occurs. The algorithm of RcB technique is shown in Fig. 5.

```
ensure              Acceptance Test
by                  Primary Alternate
else by             Alternate 2
else by             Alternate 3
...
else by             Alternate n
else failure exception
```

Fig. 5. Recovery block technique algorithm

2.6 Distributed recovery block technique

Distributed recovery block (DRB) technique, is the distributed version of RcB technique in which several recovery blocks are implemented in several systems. the only difference between these blocks is the priority of modules [6].

The basic DRB technique consists of a primary node and a shadow node, each cooperating with each other and running an RcB scheme. In DRB, the recovery blocks are concurrently executed on both nodes. The initial primary node executes the primary algorithm and the initial shadow node executes the alternate alternative one. First, the technique attempts to ensure that the primary algorithm on node 1's results passes the AT (i.e., produces a result which passes the test. If this result fails the AT, then the DRB tries the result from the alternate algorithm on node 2. If neither passes the AT, then backward recovery is used to execute the alternate on Node 1 and the primary on Node 2. The results of these executions are checked to ensure the AT. If neither of these results passes the AT, then an error occurs. If any of the results are successful, the result is passed on to the successor computing station.

2.7 Consensus recovery block technique

The consensus recovery block (CRB) technique is a combination of NVP and RcB., at first NVP runs and if it fails to produce the correct result, recovery Block runs and produces the correct result[3]. The consensus recovery block technique is represented in Fig. 6.

```
Run Ranked Version 1, Ranked Version 2, ..., Ranked Version n
If (Decision Mechanism (Result 1, Result 2, ..., Result n ) )
        return Result
else
        ensure              Acceptance Test
        by                  Ranked Version 1 [Result]
        else by             Ranked Version 2 [Result]
        ...
        else by             Ranked Version n [Result]
        else raise failure exception
return Result
```

Fig. 6. Consensuse Recovery block technique algorithm

When two or more correct answers exist for the same problem and the same input, we have multiple correct results (MCR). NVP in general and voting-type decision algorithms in particular, are not appropriate for situations in which MCR may occur. It is claimed that the CRB technique reduces the importance of the AT used in the RcB. CRB is Also able to handle cases in which NVP would not be appropriate because of MCR.

3. Acceptance Test

Acceptance Test (AT) is the most basic approach to self-checking software (Fig. 7), which typically is used with the RcB, CRB and DRB techniques. The AT is used to verify the acceptance of the systems behavior based on the assertion on the anticipated system state.

As shown in Fig. 7, a value of TRUE or FALSE is returned. The AT needs to be simple, effective, and highly reliable in order to: (1) decrease the chance of additional design faults, (2) keep run-time overhead reasonable, (3) ensure detection of the anticipated faults and (4) ensure that a non-faulty behavior would not incorrectly be detected.

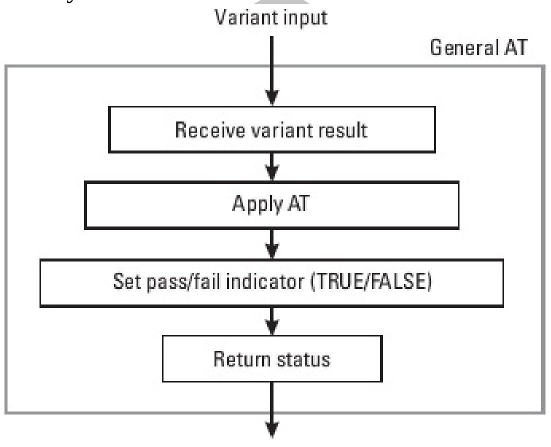

Fig. 7. Acceptance test functionality.

ATs can thus be difficult in development depending on their specifications. Also, the form of an AT depends on its application. The coverage of an AT is an indicator of its complexity, where an increase in coverage generally requires a more complicated implementation of the test. Increasing the complexity leads to increasing the time of programs execution and fault manifestations [3,7].

4. Satellite Motion System

In this section the satellite motion system, which is used in scientific computing, is introduced as a case study. The calculation of satellite motion is the most critical part of the satellite control system; so, errors in this part lead to failure of entire system. The geodetic satellites have two major missions: (1) positioning in geodesy or (2) to be used as a sensor for measuring the external gravity field of the Earth. In order to increase the reliability of this part, the fault-tolerant software techniques were utilized. Satellite motion equation is represent in Eq.(1)[8]. The analytical solution of this differential equation leads to the Kepler orbit [9].

$$\ddot{\vec{r}} = -\frac{GM}{\left|\vec{r}^{\,3}\right|}\vec{r} + K \qquad (1)$$

The satellite motion equation is a second order vector differential equation; therefore it has to be converted to a first order differential equation that is represent in Eq. (2) [8].

$$\ddot{\vec{r}} = -\frac{GM}{\left|\vec{r}^{3}\right|}\vec{r} + K \rightarrow \begin{cases} \ddot{x} = -\dfrac{GM}{\left|\vec{r}\right|^{3}}\vec{x} + K_x \\ \ddot{y} = -\dfrac{GM}{\left|\vec{r}\right|^{3}}\vec{y} + K_y \\ \ddot{z} = -\dfrac{GM}{\left|\vec{r}\right|^{3}}\vec{z} + K_z \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} v_x = \dot{x} \\ v_y = \dot{y} \\ v_z = \dot{z} \\ \dot{v}_x = -\dfrac{GM}{\left|\vec{r}\right|^{3}}x + K_x \\ \dot{v}_y = -\dfrac{GM}{\left|\vec{r}\right|^{3}}y + K_y \\ \dot{v}_z = -\dfrac{GM}{\left|\vec{r}\right|^{3}}z + K_z \end{cases} \qquad (2)$$

Where, r is the position vector, GM is the product of gravitational constant and Earth's mass, k is the effects of all the perturbing forces on a satellite. Since the equation is a second order three-dimensional differential equation, it could be solved numerically using methods such as Runge-Kutta, Adams-Bashforth and Adams-Moulton. In this paper, various implementations of these methods are used as different versions of fault-tolerant techniques.

Ruge-Kutta, Adams-Bashforth and Adams-Moulton are the most common methods for solving first order differential equations numerically. Runge-Kutta (Eq .(3),(4) and (5)) solves these equations in single-phase, while Adams-Bashforth (Eq. (6) and (7)) and Adams-Moulton (Eq. (6) and (8)) solve it in multi-phase.

$$y' = f(x, y) \qquad (3)$$

$$\begin{cases} k1 = hf(x_n, y_n) \\ k2 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k1}{2}) \\ k3 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k2}{2}) \\ k4 = hf(x_n + h, y_n + k3) \end{cases} \qquad (4)$$

$$y_{n+1} = y_n + \frac{h}{6}(k1 + 2k2 + 2k3 + k4) \qquad (5)$$

$$\begin{cases} y'(t) = f(y, t) \\ y(t_0) = y_0 \end{cases} \qquad (6)$$

$$y_n = y_{n+1} + h\sum_{i=0}^{n-1}\xi_i^p f\,(t - (1+i)h, y_i \qquad (7)$$

$$y_n^c = y_{n-1} + h\sum_{i=0}^{n-1}\xi_i^c f\,(t - ih, y_{i+1})a \qquad (8)$$

5. Multi-Core Architecture Usage

In a single-core platform, only one thread is running at a certain time point. But In a multi-core platform, there can be several threads which are running on different cores at the same time. So in the multi-core architecture, threads which are created to run the program, really run in parallel on a multi-core platform. Therefore synchronization issues and the cost for communication among cores are discussed. If the extra cost is quite considerable compared to the normal single core execution cost , such applications are not suitable for the multi-core architecture [9].

A software system is composed of a series of software key and non-key units (Fig. 8). Each software system includes critical and important parts in which occurrence of error leads to the system failure whose cost cannot be compensated. These critical and important parts are called software key units and other sections are non-key units[2].

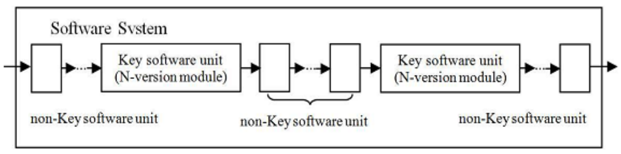

Fig. 8. non-Key software unit and key software unit

One way to increase fault tolerance is using different versions and deployment of fault tolerance techniques. But since the development of different versions of the entire system is very costly, several different versions that have different implementations are developed only for software key units. Since the key units have several versions which lead to increase of the execution time, we use multi-core architecture features to reduce time and run the versions on different cores in parallel. This approach reduces execution time and thus increases the performance. In comparison with the high cost of the sequential program, the cost of synchronization and communication between the cores is negligible [2].

6. Implementation and Results of Multi-Core Usage

The effect of multi-core architecture on increasing performance of the NVP technique has been discussed by Yang et al [10]. In this paper we discuss the effect of multi-core architecture on techniques derived from the NVP, DRB, CRB and improved consensus recovery block. In this paper, fault-tolerance techniques have been used to increase reliability; so, different implementations of numerical methods for solving differential equations of the satellite motion were used as different versions in fault tolerance techniques. Accordingly, Runge-Kutta, Adams-Bashforth and Adams-Moulton methods are implemented as different versions.

In other words, in each technique we execute different versions on single and multi-core architecture and then compare execution times on the single core with the multi-core. Finally, we offer a new technique to reach a higher performance where the execution times of techniques are significantly decreased using the multi-core architecture. As shown in Fig. 9, the speedup rate of the NVP technique for dual and quad core processors is 1.73 and 2.42 respectively. Because the reliability on this technique is low the NVP-TB-AT Technique is used instead. The speedup rate of this technique is 1.70 and 2.06 for dual and quad core processors respectively. The effect of multi-core architecture on performance of the RcB technique is shown in Fig. 10.

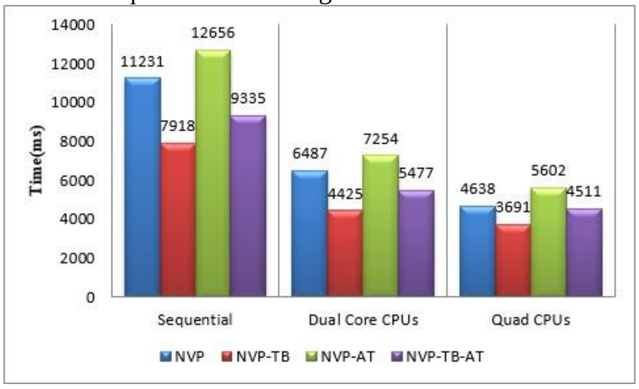

Fig. 9. Execution time of NVP technique and derived technique

The RcB technique execution time on single core, dual core and quad core are 11266, 6413 and 4718 respectively. In single core architecture, all versions are executed sequentially; so the execution time is longer than other ones. For example, in our implementation the execution time of each version is equal to 1945, 2356 and 1872 respectively. This means that the execution time of RcB technique on single core is about sum of all these times. In order to apply advantage of parallelism, we can use distributed version of this technique named Distributed Recovery Block (DRB). The DRB technique has 1.76 and 2.39 speedup rate using dual and quad core processors. Shown in Fig. 10, the execution time improvement for quad core architecture is more than dual core architecture in the case of parallelism. In other words by increasing the number of cores, an improvement of the performance is expected.

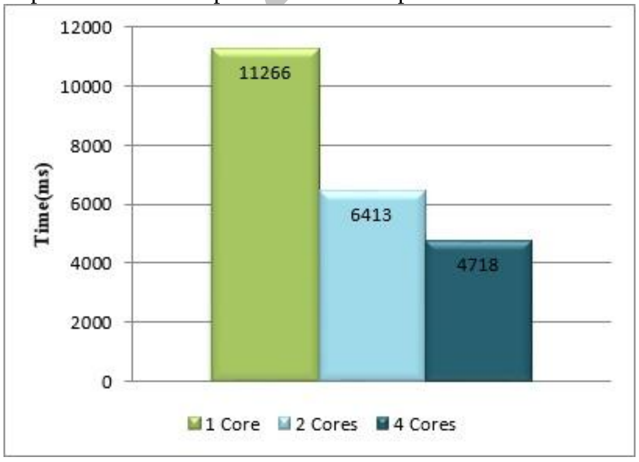

Fig. 10. Efect of multi-core architecture on performance of recovery block technique

7. Suggested Technique (Improved Consensus Recovery Block)

while using NVP-TB-AT, if the result of two faster versions were equal, one of them would be announced as the correct result and no acceptance test is performed on the results [5]. So if there is an error in the system that causes the result of two faster versions be similar and wrong, probability of the overall system failure increases using this technique. Thus this technique is less reliable than RcB technique, because in RcB technique the result goes to the acceptance test module in any conditions to be returned as a correct result. Also, if a program had several correct answers, the NVP-TB-AT technique might face failure. If both faster versions produce correct but different results, the voter waits for the slowest version and judges between results of two faster versions and result of slowest version using the decision mechanism. If the lowest version has a correct but a different result than results of faster versions, the voter cannot decide and system will face failure. But, if the RcB technique is used and the program has several correct results, system does not fail because the AT is applied to every version and so the correct result will be determined. Order of this technique is shown by Eq. (9) and (10).

$$\begin{cases} Versions = V_1, V_2, \ldots, V_n \\ V = Nmber\ of\ Versions \\ C = Number\ of\ Cores \\ Order\ of\ Versions = f(V_1), f(V_2), \ldots, f(V_n) = F_1, F_2, \ldots, F_n \\ f(V) = The\ Slowest\ Version \\ S = \dfrac{V}{C} \\ Q(V) = F_1, F_2, \ldots, F_n \end{cases} \quad (9)$$

$$RcB - Order = O(Q(V))$$

$$NVP - Order = \begin{cases} O(\dfrac{Q(V)}{C}) & if\ C \le V \\ O(\dfrac{Q(V)}{V}) & if\ C > V \end{cases}$$

$$DRB - Order = O(S \times f(V))$$

$$CRB - Order = O((NVP - Order) + (RcB - Order))$$

$$ICRB - Order = O((NVP - Order) + (CRB - Order)) \quad (10)$$

As mentioned in Section 2, different versions of RcB technique are executed consecutively. Accordingly, the RcB technique order is calculated by sum of all versions time order. In the NVP technique, time order is related to the number of versions and available cores because of running versions simultaneously. In other words, if available cores are more than the number of versions, increasing the number of cores will be ineffective on decreasing time order. On the other hand, while the available cores are equal to or less than the number of versions, increasing the number of cores leads to decrease of time order.

In the DRB technique, the execution steps of versions are computed based on relationship between the number of versions and nodes (primary and shadow nodes). This means that the arrangement for running versions considers that all versions can be performed by minimum steps. Moreover, according to concurrent execution, time order of this technique always depends on the slowest version. So, the DRB technique time order is determined by product of the number of cases in which all versions are executed and time order of the slowest version.

Since the CRB technique is a merger of the NVP and RcB techniques, the proposed technique is a combination of NVP and DRB techniques and the time order of these two techniques are computed using sum of constituent techniques time order.

On the other hand, the performance of the RcB technique is largely dependent on the performance of acceptance test. While in many cases, creation of the acceptance test module is very difficult, the CRB technique decreases the importance of acceptance test more than the RcB one. Also, the NVP technique will not be able to produce the final result when the problem has several correct answers. So, the RcB and NVP techniques have drawbacks in some cases which the CRB has resolved by combining two techniques discussed above.

According to superiority of CRB technique over other techniques, we concentrate on it and in order to improve its performance, we have proposed a technique which is similar to CRB technique and called Improved Consensus Recovery Block. In execution of CRB, first the NVP section tries to produce the correct result. If decision module was able to produce the result, the technique terminates. Otherwise, the second section namely recovery block will execute to produce the correct result. Since the execution of recovery block is sequentially, the execution time is increased. The recovery block does not use multi-core facility and therefore does not take advantage of parallel processing. In this paper, in order to take full advantage of multi-core facilities and reduce the execution time of the CRB technique, we try to use Distributed Recovery Block (DRB) instead of RcB.

Fig. 11 shows the proposed algorithm where the first versions are executed simultaneously through NVP technique and their result is given to a voter. If the voter can produce a correct result, it returns the result. Otherwise, different versions are executed through DRB technique.

```
Run Ranked Version 1, Ranked Version 2, …, Ranked Version n
If (Decision Mechanism (Result 1, Result 2, …, Result n ) )
            return Result
else
            run         RB1 on Node 1 (Initial Primary) ,
                                 RB2 on Node 2 (Initial Shadow)
            ensure      AT on Node 1 or Node 2
            by          Primary on Node 1 or Alternate on Node 2
            else by     Alternate on Node 1 or Primary on Node 2
else failure exception
```

Fig. 11. Improved consensus recovery block technique algorithm

Influence of the multi-core architecture on performance of the CRB technique is shown in Fig. 12. Different implementations of numerical methods for solving differential equations of satellite motion were used as different versions which are required in CRB technique. As Fig. 12 shows, the CRB execution time on dual-core and quad-core architectures is 19106 and 16044 respectively, while Improved Consensus Recovery Block execution time on dual-core and quad-core architectures is 12637 and 10124 respectively. So Improved CRB decreases total execution time. In other words, the speedup rate of Improved CRB in comparison with CRB for dual-core and quad-core architectures is 1.51 and 1.58 respectively. Execution of the NVP section is same in CRB and Improved CRB techniques but the difference is in the recovery block section because the CRB executes the recovery block section sequentially.

$$Speed-up = \frac{T(1)}{T(P)} \qquad (11)$$

Also, the Improved CRB technique Speed-up for 2, 4 and 8 Cores cases are represented in the Table 1, calculated using Eq.(11)[11] (Prefers to the number of cores and T(P) is the execution time using P cores).

Table 1. Speed-up of Improved Consensus Recovery Block Technique

Statuses	Speed up
2 Cores	1.78
4 Cores	2.22
8 Cores	2.29

Important point of this technique is the close relation between speed-up and both the number of versions and available cores., if the number of available cores is greater than the number of versions, increasing the number of cores will be ineffective on Speed-up improvement. Otherwise, increasing the number of cores is effective on speed-up.

In the worst case, namely the case in which last version performs the acceptance test successfully, the execution time will be equal to the total time of running all versions. However, in the Improved CRB, the recovery block section is executed distributedly and so its execution time is equal to execution time of the longest version.

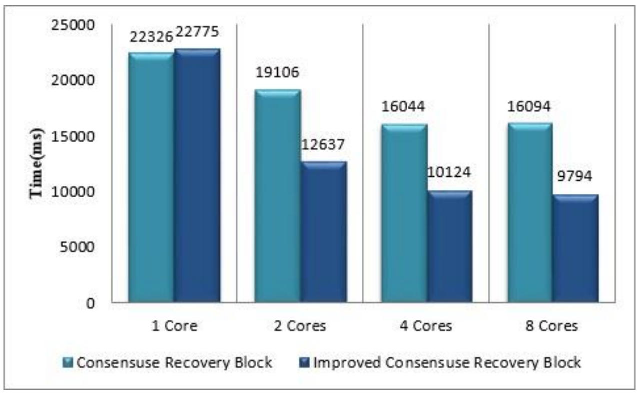

Fig. 12. Influence of multi-core architecture on performance of consensus recovery block technique

According to Fig. 10, execution time of the DRB technique in quad core architecture is less than execution time of CRB technique. But since the CRB does not have problems of the DRB technique, it is more suitable in many cases. In this paper, we showed that the CRB execution time can also be decreased.

8. Conclusions

Among different software fault tolerance techniques the Consensus Recovery Block (CRB) has more reliability over other ones in some cases and also it does not have problems of other techniques. To increase performance of this technique, we proposed one technique which is called Improved CRB technique in which the reliability is like the CRB and because of using distribution concepts, it has more performance. According to capability of multi-core architecture for supporting parallel processing, this architecture has been used to decrease the execution time and thus increasing performance of the fault-tolerance techniques. As a result, we showed that the Improved CRB technique is more suitable over other techniques from view of the reliability and performance properties.

Because the satellite motion computation is the most critical part of the system, in this paper we have used this subsystem as a case study and software fault tolerance techniques were used to solve the numerical differential equation of satellite motion in order to increase the reliability. To this end, different implementations of the numerical differential equation of the satellite motion methods were employed as different versions which are required in software fault tolerance techniques. Then, to determine the increase rate of the performance, we compared the execution time for single core architecture in the sequential mode and for multi-core one in the concurrent mode in different fault tolerance techniques.

The NVP-TB-AT technique, which has more performance and reliability over other derived NVP techniques, the execution time in case of sequential mode at single core architecture was 9335 while the execution time in case of the parallel mode at dual-core and quad-core architecture was 5477 and 4511 respectively. So, the speedup rate for dual-core and quad-core architectures is 1.70 and 2.06 respectively. Moreover, the execution time of recovery block technique on single-core, dual-core and quad-core is 22.04, 16.14 and 12.14 respectively.

Since high reliability is critical in the satellite motion computation system, we use the Consensus Recovery Block technique which has high reliability but its problem is high execution time. This problem was solved by proposing an Improved Consensus Recovery Block technique.

According to our experiments, the best execution time of Improved CRB is at quad-core architecture and it is equal to 10124, while the execution time of CRB is 16044 at quad core architecture. These two techniques have similar reliability but their performance rate is different. In other words, Consensus Recovery Block does not use distribution and concurrency mechanisms, therefore it cannot use advantages of concurrency in multi core architecture. The proposed technique has high performance because of taking advantage of distribution mechanism and using concurrency in multi core architecture.

Therefore according to the obtained results, using Improved Recovery Block technique and the multi core architecture simultaneously increases the reliability and performance in a fault tolerant software.

References

[1] A. Avizienis and J. P. J. Kelly, "Fault tolerance by design diversity- Concepts and experiments," *IEEE Computer*, vol. 17, pp. 67-80, 1984.

[2] L. Yang, L. Yu, J. Tang, L. Wang, J. Zhao, and X. Li, "McC++/Java: Enabling Multi-core Based Monitoring and Fault Tolerance in C++/Java," in *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2010, pp. 255-256.

[3] L. L. Pullum, *Software fault tolerance techniques and implementation*: Artech House Publishers, 2001.

[4] A. T. Tai, J. F. Meyer, and A. Avizienis, "Performability enhancement of fault-tolerant software," *Reliability, IEEE Transactions on*, vol. 42, pp. 227-237, 1993.

[5] A. T. Tai, J. F. Meyer, and A. Avizienis, *Software performability: From concepts to applications*: Kluwer Academic Publishers, 1996.

[6] K. H. Kim, "The Distributed Recovery Block Scheme," *M. R. Lyu(ed.), Software Fault Tolerance*, pp. 192-198, 1995.

[7] I. Koren and C. M. Krishna, *Fault-tolerant systems:* Morgan Kaufmann, 2007.

[8] M. Eshagh and M. Najafi Alamdari, "Comparison of numerical Integration methods in orbit determination of low earth orbiting satellites," *Journal of The Earth and Space Physics*, vol. 32, pp. 41-57, 2006.

[9] S. Akhter and J. Roberts, *Multi-Core Programming* vol. 33: Intel Press, 2006.

[10] L. Yang, Z. Cui, and X. Li, "A case study for fault tolerance oriented programming in multi-core architecture," in *11th IEEE International Conference on High Performance Computing and Communications, HPCC '09.* , 2009, pp. 630-635.

[11] B. Parhami, *Introduction to parallel processing: algorithms and architectures* vol. 1: Springer, 1999.

Hoda Banki is M.Sc. student at Kashan University in software engineering. She received B.Sc. degree in software engineering from Islamic Azad University Central Tehran Branch (IAUCTB) in 2008. Her main research interests are high performance computing, distributed systems, quantitative evaluation of software architecture and design based on software architecture styles.

Seyed Morteza Babamir received BS degree in Software Engineering from Ferdowsi University of Meshhad and MS and PhD degrees in Software Engineering from Tarbiat Modares University in 2002 and 2007 respectively. He was a researcher at Iran Aircraft Industries, Tehran City, Iran, from 1987 to 1993, head of Computer Center in University of Kashan, Kashan, Iran, from 1997 to 1999 and haed of Computer Engineering Department in University of Kashan from 2002 to 2005. Since 2007, he has been an associate professor of Department of Computer Engineering in University of Kashan, Kashan, Iran. He authored one book in Software Testing, four book chapters, fourteen journal papers and more than forty international and internal conference papers (http://ce.kashanu.ac.ir/babamir/Publication.htm). He is managing director of Soft Computing Journal published by supporting University of Kashan, Kashan, Iran. He is a member of the ACM.

Azam Farokh is M.S. student at Kashan University in software engineering. She received B.S. degree in software engineering from Arak University in 2009. Her research interests are including Software Engineering, high performance computing, Distributed Systems, Fault Tolernat Systems and software architecture. Her current research project is Evaluation of the appropriate style for Adaptive software architecture.

Mohammad Mehdi Morovati is M.Sc. student at Kashan University in software engineering. He received B.Sc. degree in software engineering in 2008 from college of Bahonar. His research interests include the field of Software Engineering and Artificial Intelligence and to date his focus has spanned the areas of Distributed Software Systems, Fault Tolerance Software Systems, High Performance Computing and Self-adaptive Software Systems.