ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

Assembly Language Program Evolution in Tierra-based On-Board Computer through Single Event Upset

Tomohiro Harada, Keiki Takadama

Abstract*—This paper explores the feasibility of Tierra-based OBC that evolve programs through a bit inversion caused by single event upset (SEU) in spacecraft. In particular, Tierra-based OBC is applied to the evolution of an assembly language program in order to extend its application region. This paper also proposes a novel method that can recover a correct program even if all programs are broken in Tierra-based OBC. Intensive experiments that simulate the SEU environment reveal that (1) Tierra-based OBC cannot only maintain the assembly language program through SEU but also can evolve it, i.e., minimize its program size, (2) the proposed recover method improves the robustness of Tierra-based OBC to high SEU rate in comparison with Tierra-based OBC without it, and (3) Tierra-based OBC with the proposed recover method can reliably recover a correct program by modifying broken programs even if a correct program is lost due to SEU.

***Index Terms*—single event upset, assembly program evolution, on-board computer, genetic programming.**

I. INTRODUCTION

The bit inversion called single event upset (SEU) [1] of semiconductor devices such as a memory or CPU occurs when semiconductor devices are exposed to space radiation. SEU causes software error of a spacecraft. SEU and multiple bit upsets (MBU) were, for example, observed in DRAM and SRAM on commercial semiconductor devices (CSD) on the mission demonstration test satellite-1 (MDS-1; TSUBASA) developed by JAXA[2]. To overcome this problem, conventional approaches employed (1) shielded devices, (2) multiplex logic circuit, or (3) CPUs with a tick process rule. A huge cost is, however, required in approaches (1) and (2) because they increase weight of a satellite or require much space. An approach (3), on the other hand, makes CPU calculation ability low, which cause to become difficult to conduct advanced missions that require much computational power. Unlike such hardware approaches, a checksum mechanism [3]is a typical software approach currently used. They are, however, hard to correct more than one bit inversions, i.e., MBU [1].

As a novel approach toward SEU, our previous researches proposed Tierra-based on-board computer (OBC)[4][5][6]. Tierra-based OBC is inspired by the idea of a biological simulator Tierra [7][8], where digital creatures implemented by a program are evolved through a mutation in a gene, and employs a technique of genetic programming (GP)[9]. Tierra-based OBC enables to maintain and evolve a computer program through the bit inversion caused by SEU regarding it as a mutation of a program. An advantage of Tierra-based OBC is to generate a small size program by the evolution. This means that Tierra-based OBC can generate a program that is robust to SEU since a small size of a program decreases the probability to be affected by SEU. Our previous researches showed that Tierra-based OBC can maintain and evolve a program that is written in several arithmetic instructions with two variable registers and executes simple calculation. It is, however, still simple and hard to apply to actual space missions. Toward actual applications on space missions, Tierra-based OBC has to evolve more complex programs such as written in an actual assembly language embedded on some kind of processors. Furthermore the current version Tierra-based OBC can maintain a correct program and can minimize its size, but it cannot recover a correct program when correct programs are broken caused by much SEU if a complexity of a program increases.

To tackle this problem, this paper aims at exploring the feasibility of Tierra-based OBC to evolve programs written in an actual assembly language by applying Tierra-based OBC to the evolution of assembly language programs for space mission, and proposing a novel method to reliably recover a correct program from a broken one in addition to maintain and minimize a program. Toward this aim, this paper employs an instruction set that can be used on a PIC microcontroller[10] developed by Microchip Technology Inc.. This is because the PIC microcontroller is carried on many kinds of spacecraft and it can execute programs for space missions. The PIC instruction set consists of 33 basic instructions composed of 12bits such as addition, subtraction, the Boolean logic,

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

bitwise operations, and branch instructions. 16 general purpose registers and one working register, which are all composed of 8 bits, can be used. The essential differences between the simple program we previously used and the PIC program are summarized as follows: (1) the previous program is sequentially executed from top to end, while the PIC program can execute branch structure such as a loop or a conditional branch; and (2) the previous program only uses one or a few registers, while the PIC program needs to handle with total 17 registers (16 general purpose registers and one working register), which increases the difficulty of a program and makes it difficult to evolve it. To investigate the feasibility of Tierra-based OBC to evolve the PIC program and the effectiveness of the proposed method to recover a correct program, this paper conducts the following experiments in Tierra-based OBC with and without the proposed recover method: (1) evolving the PIC program that execute numerical calculation and the Boolean calculation that supposed to be employed on actual space missions; and (2) recovering a correct program from the situation where correct programs are broken during the evolution process. The experiments simulate SEU with several occurrence probability and evaluate the feasibility of Tierra-based OBC from the viewpoint of whether it can maintain the correct program and how small and short execution program can be generated.

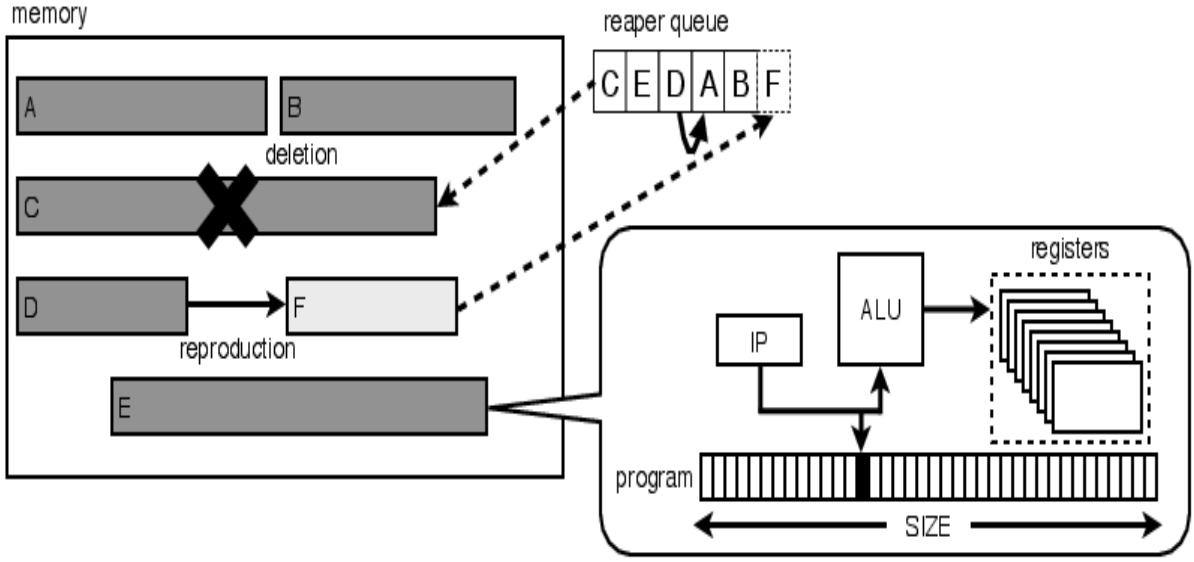


Fig 1An illustration of Tierra-based OBC

The remaining of the paper is organized as follows. Section II explains Tierra-based OBC we have proposed and Section III points its problems. Section IV describes the proposed recovering method. Section V explains test bed problems and experimental settings. Section VI shows experimental results and Section VII discusses on their results. And finally Section VIII concludes this paper and presents future works.

II. TIERRA-BASED ON-BOARD COMPUTER

A. Overview

Tierra-based On-Board Computer (OBC) is an OBC based on the idea of Tierra[7][8] that is a biological evolution simulator, where digital creatures are evolved through a cycle of a self-reproduction, a deletion and genetic operators such as a crossover, a mutation or an instruction insertion/deletion. To evolve a program that can solve any given tasks from the engineering point of view, e.g., orbit/altitude control, navigation, communication and optimization, unlike Tierra can only evolve programs that aim at reproduce themselves, Tierra-based OBC introduces *fitness* commonly used in evolutionary algorithms such as genetic algorithm (GA) [11] or genetic programming (GP) [9] to evaluate programs, i.e., programs are reproduced or deleted according to their *fitness* values. Mean square error or error ratio towards the target value is usually employed as the fitness evaluation.

Fig 1shows an illustration of Tierra-based OBC. Tierra-based OBC starts from programs that completely solve the given engineering task (aim). These programs consist of a sequence of instructions with some registers and they are stored in a limited memory space, i.e., a population in evolutionary algorithms. All programs are assigned in

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

reaper queue (named in Tierra) that controls their lifespan. A few instructions are executed in turn for a parallel execution of programs. When all instructions in one program are completely executed, its fitness is evaluated depending on its calculation results, and a program is asynchronously reproduced according to its fitness value.

Then its position in the reaper queue changes depending on whether it is reproduced or not. If it is reproduced, its position in the reaper queue moves to lower, while if not, its position gets upper. When the memory is filled with programs, programs located at the upper position in the reaper queue are deleted from the memory.

Algorithm 1The algorithm of Tierra-based OBC

1: $prog.f_{acc} \leftarrow prog.f_{acc} + prog.f$
2: **if** $prog.f_{acc} \geq f_m$**then**
3: $prog.f_{acc} \leftarrow prog.f_{acc} - f_{max}$
4: **repeat**
5: down position of pr in the reaper queue
6: **until** $rand(0,1) < P_{down}(prog.$
7: generate an offspring by mating pr with $pro\,g_p$ through the genetic operators
8: **if** $prog.f_{acc} = f_m$**then**
9: **if** pr is better than $elite_p$**then**
10: reproduce pr without any genetic operators
11: **else**
12: generate an offspring by mating pr with $pro\,g_p$ through the genetic operators
13: **end if**
14: $elite_{prev} \leftarrow prog$
15: **end if**
16: $pro\,g_{prev} \leftarrow prog$
17: **while** the number of programs exceeds the maximum number of the reaper queue **do**
18: delete a program located at upper position in the reaper queue
19: **end while**
20: **else**
21: **repeat**
22: up position of pr in the reaper queue
23: **until** $rand(0,1) < P_{up}(prog.$
24: **end if**

A. Algorithm

The algorithm of the current version of Tierra-based OBC is described in Algorithm 1. In this algorithm, pr indicates a program currently executed, and $prog$ and $prog.f$ indicate its fitness and its accumulated fitness respectively. $pro\,g_p$, indicates a program previously executed. $f_m$indicates the maximum value of the fitness, while $rand(0,$ indicates the random real value between 0 to 1. $P_{do}$and P are explained later. Tierra-based OBC can evolve programs for a given task through the following (1) selection and reaper queue control, (2) reproduction and (3) deletion procedures.

1) Selection and reaper queue control

When one program completes its execution, its fitness $prog.f$ is evaluated according to its calculation results and the calculated fitness is added to an accumulated fitness $prog.f_{acc}$ (line 1 in Algorithm 1). If an accumulated fitness of a program exceedsf_{max}, which is the maximum value of the fitness, it is selected as a parent program andf_{max} is subtracted from its accumulated fitness (lines 2 and 3). This selection mechanism selects a program having a high fitness as a parent program with a high probability because the accumulated fitness frequently exceeds f_{max}, while a program having a low fitness is hard to satisfy this condition.

After that, the position in the reaper queue of a program selected as a parent program become lower than the current position, i.e., its deletion probability decreases and it survives long (lines 4-6). While the position of a program not selected as a parent program becomes upper, i.e., its deletion probability increases andit becomes to

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

be easily removed (lines 20-22). The movement distance to lower/upper is determined by the probability P_{down} and P_{up} that are calculated as the following equation based on fitness of the program:

$$P_{down}(f) = \frac{f}{f_{max}} \times \frac{N-1}{N}, P_{up}(f) = \left(1 - \frac{f}{f_{max}}\right) \times \frac{N-1}{N},$$

where N is the maximum reaper size. For example, the fitness is calculated as f_{max}, P_{down} is calculated as $(N-1)/N$, which is the maximum probability. In this case a lot of down process is executed while $rand\,(0,1)$ produces smaller value than $P_{down} = (N-1)/N$ and its position in the reaper queue changes close to the lowest in the reaper queue.

2) Reproduction

Tierra-based OBC generates a new program by mating $prog$ with $prog_{\text{prev}}$, which is a program previously selected as a parent, through the genetic operators such as a crossover, a mutation, and an instruction insertion/deletion operators (line 7). The crossover operator combines $prog$ with $prog_{\text{prev}}$. The mutation operator changes one random instruction in $prog$ to other random instruction. The insertion operator inserts one random instruction into $prog$, while the deletion operator removes one instruction selected at random from it.

Additionally, to preserve programs that can completely execute the given task, the elite preserving strategy[12] is applied (lines 8-15). Concretely, if $prog$ is calculated as f_m, which means it completely execute the given task, it is selected as an eli. It is compared with the previous eli represented as $elite_{p_1}$. Then if pr is better than $elite_{p_1}$, it is reproduced (copied) as an elite program *without* the genetic operators (line 10). While if not, it generates an offspring mated with $prog_{p_1}$ *with* genetic operators to utilize a better program for the evolution (line 12).

3) Deletion

Tierra-based OBC conducts a deletion operator when a new program is generated and the number of programs in the reaper queue exceeds the maximum size of the reaper queue (lines 17-19). Concretely a program located at upper in the reaper queue is randomly selected and is deleted from a memory. Such program is commonly aged and having a low fitness. By this deletion mechanism depending on the reaper queue, Tierra-based OBC can delete programs that do not complete their execution due to some reasons, for example, a generation of the infinite loop caused by SEU.

B. Hardware architecture

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

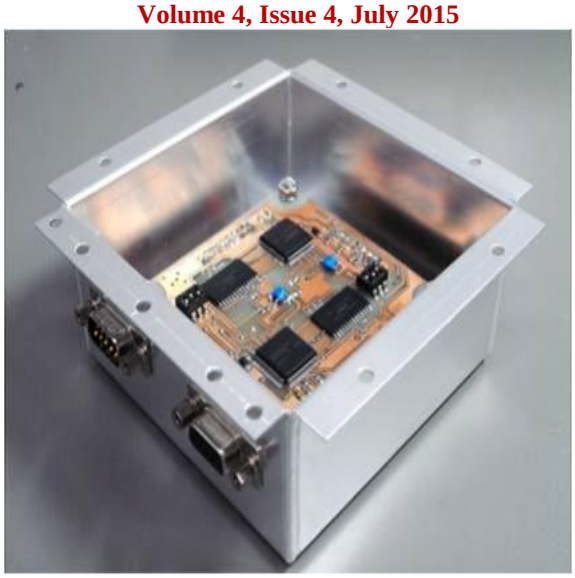


Fig 2 Flight model of Tierra-based OBC on SHIN-EN (UNITEC-1)

We constructed Tierra-based OBC in our previous research [6] with two H8 micro controller units (MCU) and two DRAMs as shown in Fig 2, and it is carried on SHIN-EN (UNITEC-1) [13] and launched in 2010. H8 is the microprocessor developed by Renesas Technology that is constructed on one chip with a CPU core, internal memory, timers, and I/O ports. Tierra-based OBC consist of two layers, the controller layer that executes the main Tierra-based OBC process and the communication layer that communicates with the mother OBC (MOBC). The details of components are shown in Table 1. Tierra-based OBC consists of two H8/3069s (25MHz), two DRAMs (16MBit), and 6 EEPROMs (256KBit). It weighs 197g. Evolved programs on the Tierra-based OBC are stored in DRAM, and H8 executes the evolution processes — processes that should not change. From this reason, H8/3069 is employed because it is robust against space radiation and many university satellites with H8/3069 successfully operate on actual space missions. Ordinary DRAM is used because programs are expected to be evolved by bit inversion. We confirmed that the evolution process is successfully executed on this OBC architecture.

Table 1 Components in Tierra-based OBC on SHIN-EN (UNITEC-1)

MCU	H8/3069 (25MHz) 2
RAM	DRAM (16Mbit) 2
ROM	EEPROM (256Kbit) 2
weight	197g

III. PROBLEMS OF CURRENT VERSION OF TIERRA-BASED OBC

Our previous researches [4][5][6] reported that the current version of Tierra-based OBC can evolve a simple program that includes simple four arithmetic operations, addition, subtraction, multiplication and division, and uses a few registers even if random bit inversions caused by SEU occur in a program memory space and register values during the evolution. However, the current version of Tierra-based OBC has the following two problems: (1) it is not investigated to evolve more complex programs like an assembly language program including branch structures and more variable registers; and (2) although it can maintain a correct program and can minimize its size, it is hard to recover a correct program when correct programs in a memory are all broken caused by SEU, in particular if a complexity of an evolved program increases.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

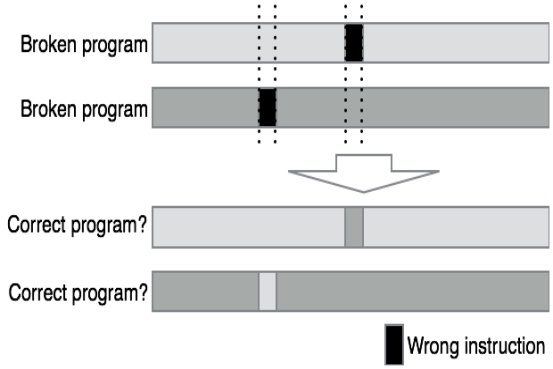


Fig 3An illustration of the proposed recover crossover

Regarding the first problem about the evolution of a complex program, to apply Tierra-based OBC to actual space missions, it is necessary to evolve a *machine code level* program, i.e., executable on an actual processor. Toward this aim, this paper explores the feasibility of Tierra-based OBC to evolve programs written in an actual machine language by applying Tierra-based OBC to the evolution of an actual machine language program for space mission.

In particular, we employ an assembly language embedded on a PIC microcontroller [10]. The PIC microcontroller unit is developed by Microchip Technology Inc.. The PIC instruction set consists of 33 instructions composed of 12bits, which includes addition, subtraction, the Boolean logic, bitwise operations, and branch instructions. A program can use any of 16 general purpose registers and one register named as *working register*. Each register consists of 32bits. Since the PIC microcontroller is usually used in actual space missions, it is possible to apply Tierra-based OBC to actual space missions if the PIC assembly programs can be evolved in Tierra-based OBC.

Regarding the second problem about recovering a correct program, it is desired for Tierra-based OBC not only to maintain and evolve a program but also recover a correct program even if correct programs are broken caused by SEU. It is, however, hard to reliably recover a correct program in such situation in the current version of Tierra-based OBC because it has to recover a correct one depending on the evolution process. For this reason, this research proposes a novel method to reliably recover the correct program from the broken programs and introduces it into Tierra-based OBC.

IV. PROPOSED METHOD: RECOVER CROSSOVER

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

A. Overview

To recover a correct program through the evolution process of Tierra-based OBC, this paper proposes a novel method named as *recover crossover (RX)*. It is assumed that, in the SEU environment, a broken program cannot execute a correct calculation caused by the change of a few instructions in it, but not caused by the change of most instructions in it. Furthermore it two or more programs are broken by SEU, it is also assumed that positions of the changed instructions in their programs differ from each other, i.e., different instructions may change. Hence a correct program can be recovered by recombining only correct instructions from broken programs. For this assumption, the proposed recover crossover try to generate a correct program by recombining two broken programs through the evolution process of Tierra-based OBC.

Fig 3 illustrates an image of the proposed recover crossover, where rectangles indicate programs in which wrong (bit inverted) instructions are shown as black rectangles, upper two rectangles indicate broken programs, while lower two rectangles indicate recovered ones. In Fig 3, two programs are broken caused by wrong instructions (black in Fig 3) but other instructions are correct. In this case, if a wrong instruction is exchanged to an instruction located at the same position in other program, a correct program can be recovered because other program has a correct instruction.

B. Algorithm of recover crossover

The recover crossover is applied to programs that are broken, i.e., have lower fitness than the maximum fitness (f_{max} in Algorithm 1). Concretely, the recover crossover is conducted when a program that is evaluated its fitness but its accumulated fitness does not exceed the threshold, which corresponds to line 20 in Algorithm 1. When conducting the recover crossover, a partner program that has similar size to a recovered crossover is selected from the memory. Modified algorithm of Tierra-based OBC with the recover crossover is described in Algorithm 2. Algorithm 2 only shows the description after line 20 in Algorithm 1. In Algorithm 2, $prog$ indicates a current program, while$prog_{par}$ indicates a selected partner that has similar size of $prog.similar_program$procedure gives a program that has similar size of a current broken program. Note that the recover crossover does not generate new offspring unlike the general crossover, and directly changes a current broken program.This paper proposes two type of the recover crossover named as *uniform recover crossover (URX)* and *differential recover crossover (DRX)*.

The uniform recover crossover, as shown in Fig , is completely the same way as the general uniform crossover, where randomly selected instructions in two programs are exchanged each other. In this method, if wrong

Algorithm 2The algorithm of Tierra-based OBC with the recover crossover (partially omitted from Algorithm 1)

1: $prog.f_{acc} \leftarrow prog.f_{acc} + prog.f$
2: **if** $prog.f_{acc} \geq f_m$**then**
...
20: **else**
21: $prog_{par} \leftarrow similar\_program(prog)$
22: conduct the recover crossover to pro by mating with $prog_p$
23: **repeat**
24: up position of pro in the reaper queue
25: **until** $rand(0,1) < P_{up}(prog.$
26: **end if**

1: **procedure** $similar\_program(pro$ **do**
2: $result \leftarrow reaper\_queue[0]$
3: **for all** **in** $reaper\_que$**do**
4: **if** $|prog.size - result.size| > |prog.size - p.siz$**then**
5: $result \leftarrow p$
6: **end if**
7: **end for**
8: **return** $resu$
9: **end procedure**

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

instructions are selected as exchanged ones, a correct program can be recovered.

On the other hand, the differential recover crossover, as shown in Fig 5, exchanges instructions that is different in two programs. From the assumption described above, since positions of the changed instructions in their programs differ from each other, exchanging different instructions in two programs has great potential to recover a correct program with minimum change of a program. To achieve the differential recover crossover, it is necessary to measure the difference between two programs. This paper employs the *Levenshte in distance* (or known as *edit distance*)[14]to measure the difference. The Levenshtein distance is originally developed to measure the difference of two strings or sequences by considering a single character insertion, deletion, or substitution as a single operator and counting the minimum number of operations from one string/sequence to another one. In the GP researches, especially in the Linear GP researches [15], which represents an evolved program as a linear structure, the Levenshtein distance is employed as the distance metric to measure the distance of two evolved program. By using the Levenshtein distance, we distinguish which instructions are inserted, deleted, or substituted from a current broken program to another selected one, and the differential recover crossover applies their distinguished operators to a current program with a certain probability, i.e., $1/(\text{the number of the different instructions})$.

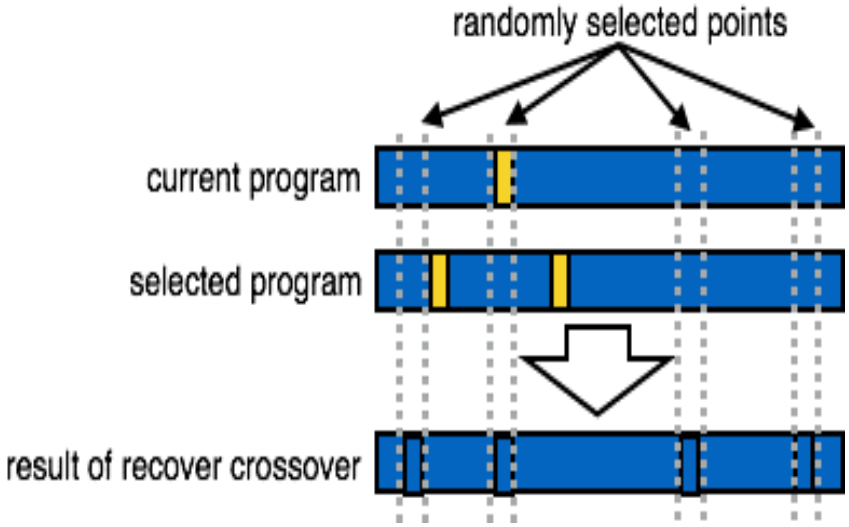


Fig 4An image of the uniform recover crossover. Yellow rectangles indicate wrong instructions, while gray lines indicate randomly selected crossover points.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

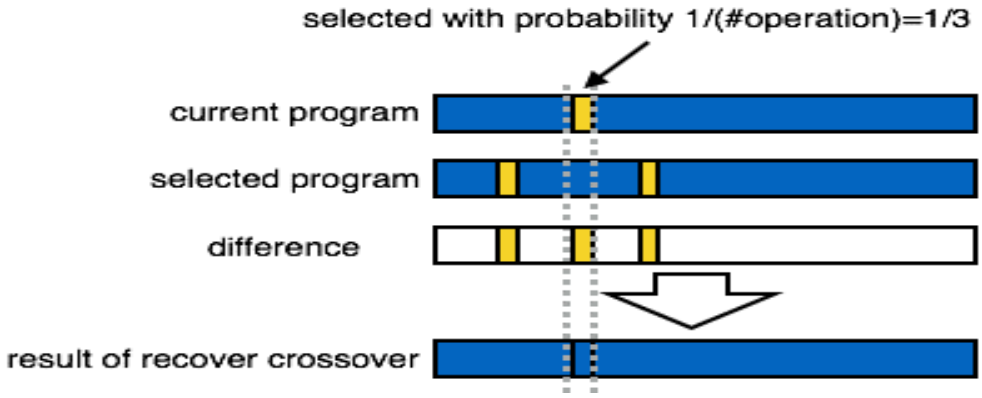


Fig 5An image of the differential recover crossover. Yellow rectangles indicate wrong instructions, "difference" rectangle indicates the difference between two programs from the Levenshtein distance, while grapy lines indicate a crossover point selected from "difference".

V. EXPERIMENTAL SETTINGS

A. PIC assembly language

This paper employs the PIC assembly language program as an actual machine-code program, and we employ an instruction set embedded on PIC10 developed by Microchip Technology Inc., which consists of 33 instructions composed of 12bits such as addition, subtraction, the Boolean logic, bitwise operations, and branch instructions. Each program uses 16 general purpose registers (R0-R15) and one special register named as *working register*(W). Each register consists of 32bits integer variable in this paper. In the PIC instructions, the value in a general purpose register cannot be directly stored to another general purpose register, and to do this, the value is stored to another one via the working register. Note that since the PIC instruction set does not include multiplication instructions, the multiplication has to be achieved by repeating additions and bitwise operations. For this reason, a program that aims to calculate multiplications has to include loop structures, which increases the complexity of a program.

B. Problem description

Test bed programs in our experiments are shown in Table 2. This paper employs four numerical calculations and four even-parity calculations. In the N1, N3 and N4 test beds, the input value x is initially stored to theR1 register and the value of the R0 register is finally used as the output value of a program. In the N2 test bed, the input values x and y are initially stored to the R1 and R2 registers and the value of the R0 register is finally used as the output value. In the even-parity programs, the registers of the number of the input values from R1 is used to store the input values, for example in 5bits even parity program, the R1 to R5 registers are used.

Table 2 Test bed problems in this paper

Numerical calculations				
ID	Function	The number of inputs	Initial size	Initial execution step
N1	$f(x) = x^4 + x^3 + x^2 + x$	16	62	715
N2	$f(x, y) = x^y$	25	25	708
N3	$f(x) = x^5 - 2x^3 + x$	16	61	714
N4	$f(x) = x^6 - 2x^6 + x^2$	16	61	714
Even-parities				
ID	Function	The number of inputs	Initial size	Initial execution step
E1	5bits-Parity	32(= 2^5)	15	15
E2	6bits-Parity	64(= 2^6)	17	17
E3	7bits-Parity	128(= 2^7)	19	19

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

E4	8bits-Parity	256($=2^8$)	21	21

For the fitness calculation of a program, the following fitness functions are employed for the numerical and the even-parity problems respectively;

$$f_N = \frac{f_{max}}{1 + \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i^*|}, \quad (1)$$

$$f_E = \frac{f_{max}}{1 + \frac{1}{n}\sum_{i=1}^{n} \delta(\hat{y}_i, y_i^*)}, \delta(x,y) = \begin{cases} 0 & x = y \\ 1 & x \neq y \end{cases} \quad (2)$$

where n indicates the number of inputs, $\hat{y}_i$ and y_i^* indicate the i[th] output value of a program the i[th] target value respectively. In both fitness functions, if the output values are all correct, i.e., $\hat{y}_i = y_i^*$ for all i, the fitness value becomes f_{max}, while if the difference between the output values and the target values increases, the fitness value gradually decreases.

C. Cases

To investigate the feasibility of Tierra-based OBC to evolve the PIC assembly program and to investigate the effectiveness of the proposed recover crossover, this paper conducts the following two experiments in Tierra-based OBC:

- **Case1** evolves the PIC assembly program that executes the numerical calculation and the even-parity calculation in various SEU rate to evaluate whether Tierra-based OBC with and without the proposed recover crossover can evolve the PIC assembly program in the SEU situation.
- **Case2** recovers the PIC assembly program in the situation where correct programs are all broken in the middle of the evolution process, i.e., all programs in Tierra-based OBC cannot calculate the correct results to evaluate whether Tierra-based OBC withand without the proposed recover crossover can reliably recover correct programs.

In both cases, the experiments start from an initial program that completely executes numerical calculations or the even-parity calculation shown in Table 2, and Tierra-based OBC with/without the recover crossover are compared. Afterword, Tierra-based OBC *without* the recover crossover is represented as TOBC-noRX, one *with* the uniform recover crossover is represented as TOBC-URX, and one *with* the differential recover crossover is represented as TOBC-DRX.In Case2 where all correct programs are broken in the middle of the evolution, one instruction of all programs simultaneously change to another random instruction and all programs cannot calculate the correct result at half of the maximum execution time (10×10^8 unit time).

D. Parameter settings

Table 3 summarizes parameter settings of Tierra-based OBC with and without the recover crossover used in this experiment. The population size, i.e., the reaper size is set as 20, which is smaller than a setting used in general EAs but is used because an actual memory resources on spacecraft will be limited. General settings of the crossover rate, the mutation rate, the insertion rate and the deletion rate are selected, i.e., high crossover rate and low mutation, insertion and deletion rate. The unit time is defined to be able to execute 100 instructions, while the various SEU rates are tested to investigate the robustness of Tierra-based OBC to high SEU rate. Note that SEU (the bit inversion) occurs not only to a program but also to register values of a program. In the proposed recover crossover, the uniform recover crossover exchanges each instruction with 50% of the probability, while the differential recover crossover exchanges different instructions with $1/(\text{the number of the different instructions})$ of the probability.

Table 3 Parameter settings of TOBC with/without the recover crossover

Population size	20
Crossover rate	0.7
Mutation rate	0.1
Insertion rate	0.1
Deletion rate	0.1
#executable instructions	100insts./unit time
SEU rate	$10^{\{-3,-4,-5\}}$/unit time
Execution time	2.0×10^8 unit time

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

Each experiment conducts 30 independent runs for each cases with each SEU rate. We evaluate (1) a success rate that is calculated from the number of the runs that successfully maintain the correct programs at the end of the evolution, and (2) the reduction ratio of the program size at the end of the evolution averaged from 30 runs. Note that the criterion (2) is given by the correct and the best program in each run.

VI. RESULTS

A. Case1

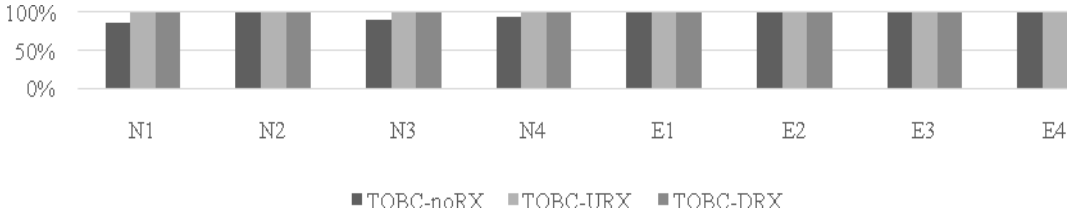
Figure 6 shows the result of the success ratio in Case1. In Figure 6, the horizontal axis indicates the type of the test bed problems, while the vertical axis indicates the success ratio in 30 runs. The difference of colors indicates the difference of the method of Tierra-based OBC, which indicates TOBC-no RX, TOBC-URX and TOBC-DRX from left to right. From Figure 6, it is indicated that Tierra-based OBC with and without the recover crossover achieve 100% success ratio in all test beds in the SEU ratio smaller than $10^{-4}/\text{unit time}$. This result indicates that Tierra-based OBC can maintain the correct PIC program under the SEU environment even though it is more complex than the program employed in our previous researches. On the other hand, in the success ratio in the high SEU ratio as $10^{-3}/\text{unit time}$, TOBC-no RX fails to maintain the correct PIC programs in some test beds, N1, N3, and N4, as shown in Figure 6(a). This result indicates that it is hard for the current version of Tierra-based OBC to evolve a complex program such as the PIC program in high SEU ratio. While as shown in Figures 6(b) and 6(c), Tierra-based OBC *with* the recover crossover, TOBC-URX and TOBC-DRX, achieves 100% of the success ratio even in the high SEU ratio as $10^{-3}/\text{unit time}$. Figure 7 shows the result of the reduction ratio of the program size averaged from 30 runs in each test bed. The horizontal and the vertical axes indicate the type of the test bed problems and the reduction ratio of the program size respectively. The positive value of the reduction ratio indicates the smaller size of the program is generated from the initial program, while the negative value indicates the larger size of the program is generated. The difference of colors indicates the difference of the method of Tierra-based OBC, which indicates TOBC-no RX, TOBC-URX and TOBC-DRX from left to right. This result indicates that Tierra-based OBC with and without the recover crossover can generate small size programs except the N4 test bed. This means that Tierra-based OBC cannot only evolve a simple program that is previously employed, but also can evolve a complex program like the PIC program under the SEU environment. Focusing on the influence of the recover crossover, all methods, TOBC-noRX, TOBC-URX and TOBC-DRX, show similar tendency in all test beds. This indicates that the proposed recover crossovers do not obstruct the evolution of the program. The reason why the negative reduction ratio is observed in the N2 testbed, i.e., why the program size increases, is that the initial program of the N2 test bed include double loop and it is easily expanded to decreases the execution step of a program. In fact, the reduction of the execution step in the N2 is confirmed from the evolved program. From these results, it is revealed that Tierra-based OBC can maintain and evolve a complex program like the PIC assembly program and the proposed recover crossover improves the robustness of the Tierra-based OBC to the bit inversion.

B. Case2

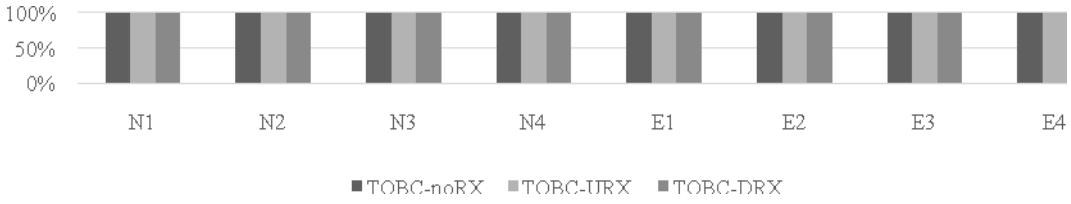
Figure 8 shows the result of the success ratio in Case2. Figure 8 has the same meaning as Figure 6 in Case1. From Figure 8, it is indicated that TOBC-noRX is hard to recover the correct PIC program after breaking all programs because the success ratio is almost 0% in all testbeds and all SEU rate situations. In contrast to this, TOBC-URX and TOBC-DRX both improve the success ratio than TOBC-noRX. This indicates that the proposed recover crossover contributes to recover the correct PIC program from the broken one in Tierra-based OBC. Comparing URX with DRX, DRX always has high success ratio than URX and achieves the success ratio larger than 85% in all testbeds, unlike URX achieves the success ratio between 60% and 90% in all testbeds. Figure 9 shows the result of the reduction ratio of the program size in Case2. In Figure 9, the horizontal and the vertical axes have the same meaning as Figure 6, and the results of TOBC-URX and TOBC-DRX are only shown but the result of TOBC-no RX is not shown in this figure because its success ratio is almost 0%. From this figure, it is indicated that the similar result as Case1 is achieved in Case2. Concretely, Tierra-based OBC with the recover crossover does not only recover the correct PIC program from a broken one but also continues to evolve programs after recovering the correct PIC program even in the situation that all programs are broken.

From these results, it is revealed that the recover crossover is effective to recover the correct PIC program even though all PIC programs are broken. In particular it is more effective to recombine only different instructions between two programs, i.e., the differential recover crossover, than to randomly recombine two programs, i.e., the uniform recover crossover.

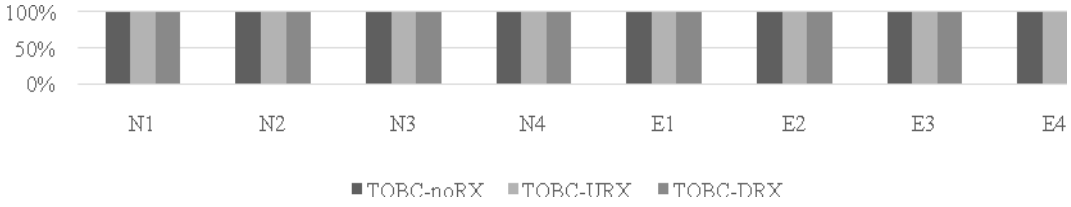
ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015



(a) SEU rate=10^{-3}

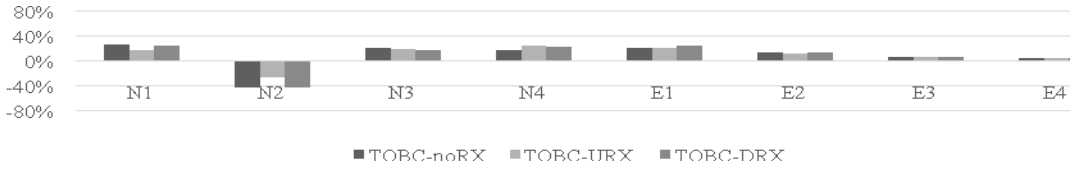


(b) SEU rate=10^{-4}

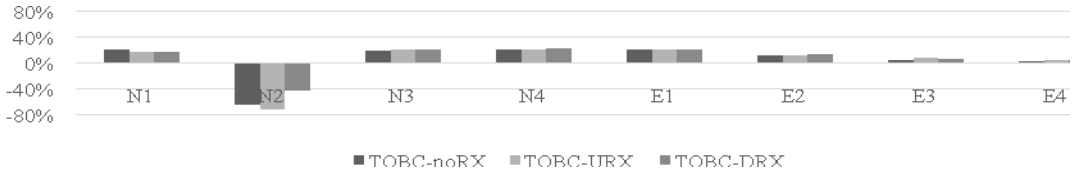


(c) SEU rate=10^{-5}

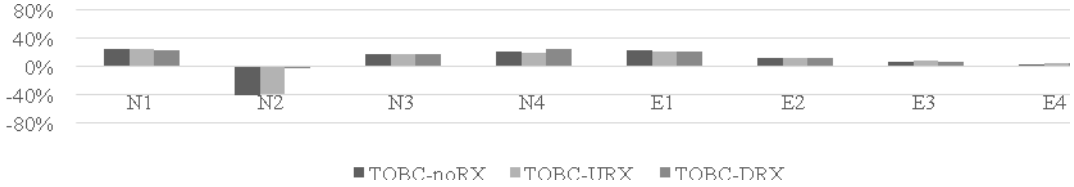
Fig 6 The success ratio in Case1.



(a) SEU rate=10^{-3}



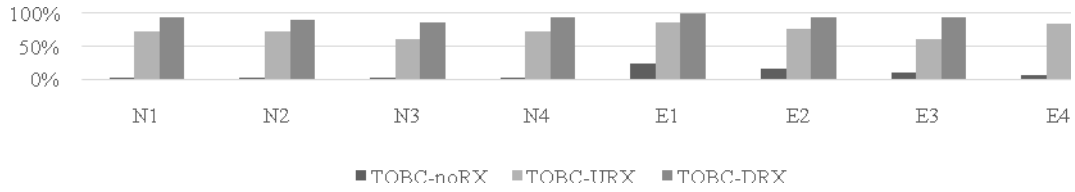
(b) SEU rate=10^{-4}



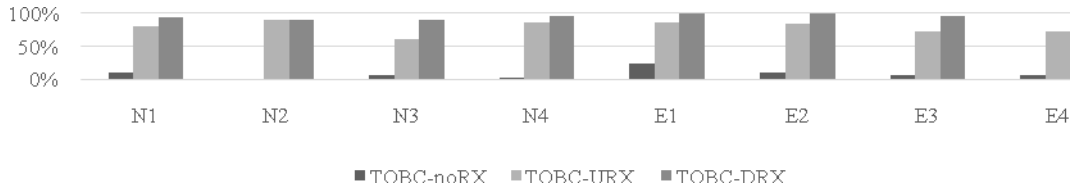
(c) SEU rate=10^{-5}

Fig 7 The reduction ratio of the program size in Case1.

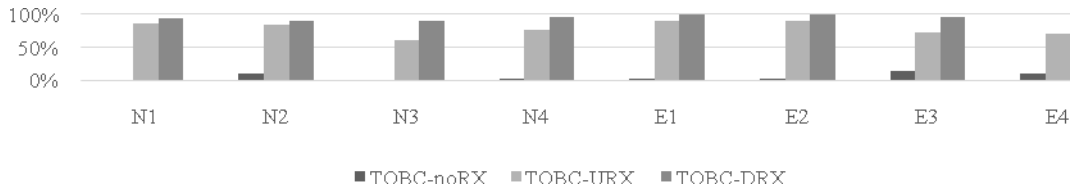
ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015



(a) SEU rate=10^{-3}

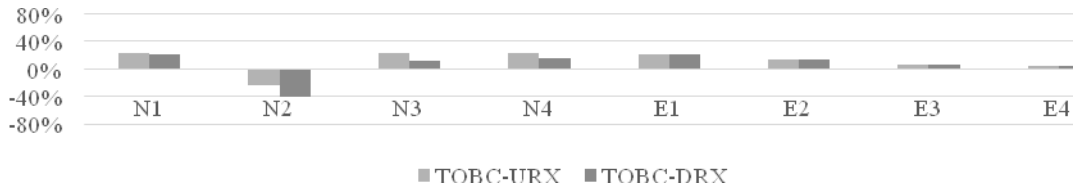


(b) SEU rate=10^{-4}

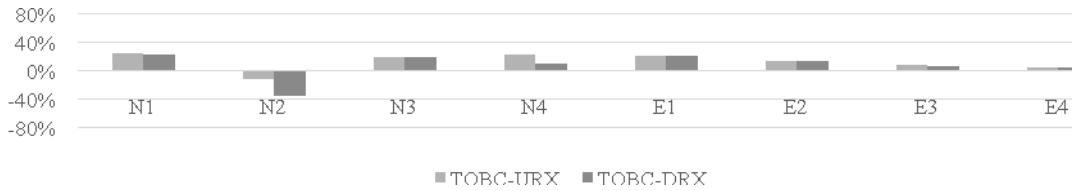


(c) SEU rate=10^{-5}

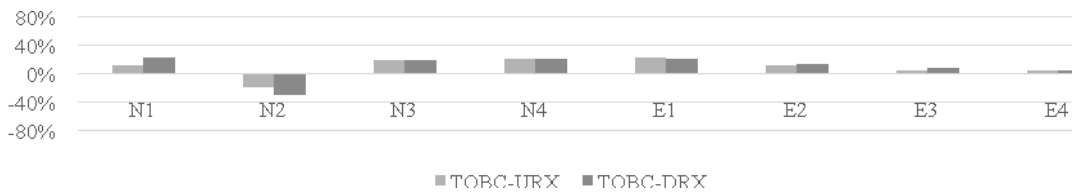
Fig 8 The success ratio in Case2.



(a) SEU rate=10^{-3}



(b) SEU rate=10^{-4}



(c) SEU rate=10^{-5}

Fig 9 The reduction ratio of the program size in Case2 (without TOBC-noRX).

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

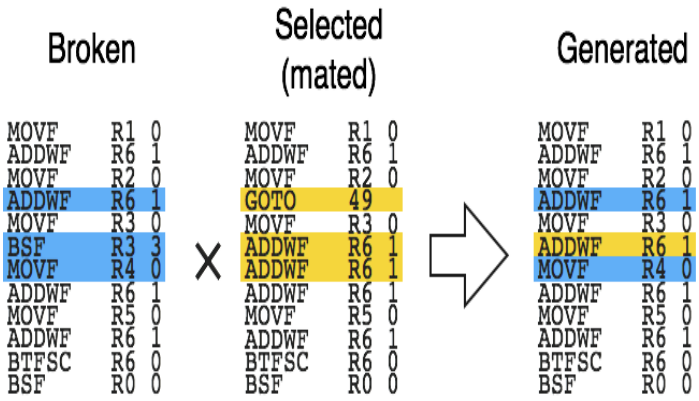


Fig 10An example of the recover process with the differential recover crossover

VII. DISCUSSION

To verify the effectiveness of the recover crossover, Figure 10 shows an example of the recover process that is actually performed by the differential recover crossover (TOBC-DRX) in the E1 testbed (5bits-Parity) in Case2. In Figure 10, the left, the center and the right figures are the PIC assembly program evolved in TOBC-DRX, and each line indicates one instruction, for example, "MOVF R1 0" instruction move the R1 value to the working register. The colored instructions differ between the left and the center programs. The left figure shows the broken program and the differential recover crossover is conducted to this program by mating with the center one. Note that these two programs are both broken and cannot calculate the correct result. The differential recover crossover randomly selects exchanged instruction(s) from the different instructions, i.e., the colored instructions, and in this example, the second colored instruction ("BSF R3 3" in the left program) is selected and replaced with the instruction at the same point in the mated one ("ADDWF R6 1" in the center program). Finally the program shown in the right figure is generated by the differential recover crossover, which becomes to be able to calculate the correct result.

In this example, the recovery succeeds only if the second instruction is exchanged. For this fact, the probability of the success of the differential recover crossover is $1/3 \times (1-1/3)^2 = 4/27$ because an exchanged instruction is selected with the probability of $1/(\text{the number of the different instructions})$. In contrast to this, the probability of the success of the uniform recover crossover is $1/2 \times (1-1/2)^2 = 1/8$ because an exchanged instruction is randomly selected and it is less probability than one of the differential recover crossover. In general, representing the number of the different instruction between mated two programs as D, and assuming that only one instruction has to be exchanged to recover a correct program, the success probability of the differential recover crossover is calculated as $(1/D)^1 \times (1-1/D)^{(D-1)} = (D-1)^{D-1}/D^D$, while one of the uniform recover crossover is $(1/2)^1 \times (1-1/2)^{(D-1)} = (1/2)^D$. The relationship between these two probabilities is shown in Figure 11, where the horizontal axis indicates the number of the different instructions between mated two programs while the vertical axis indicates the success probability of the recovery. The dark gray line indicates the probability of the differential recover crossover, while the light gray line indicates the one of the uniform recover crossover. From Figure 11, it is indicated that the probability of the differential recover crossover ($= (D-1)^{D-1}/D^D$) is always higher than the one of the uniform recover crossover ($= (1/2)^D$).

From this analysis, it is revealed that the differential recover crossover has high probability to recover a correct program than the uniform recover crossover if the influence of SEU is not too large. And this also indicates that the success probability of the recovery decreases as increasing the difference between two programs, i.e., the recovery gets difficult as increasing the number of wrong instructions in a broken program.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

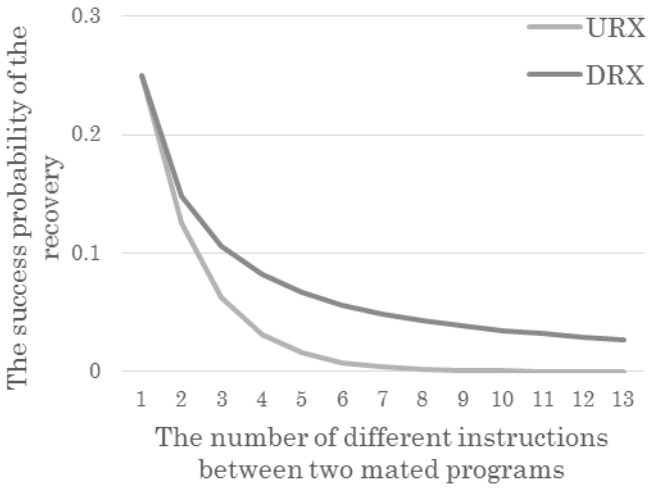


Fig 11The success probability of the recovery when only one instruction has to be exchanged between mated two programs.

VIII. CONCLUSION

This paper explored the feasibility of Tierra-based OBC to evolve the PIC assembly program and the effectiveness of the proposed method to recover a correct program from a broken one in Tierra-based OBC. The PIC assembly program is more complex than the program out previous researches used and can be applied to actual space missions. The proposed recover crossover modifies a broken program by combining it with other (broken) program. We conducted the simulation experiments that evolve the PIC assembly programs in the SEU environment with various SEU rate. The following implications have been revealed from the experimental results: (1) Tierra-based OBC cannot only maintain the PIC assembly program, but also evolve it even in the SEU environment; (2) the proposed recover crossover enhances the robustness of Tierra-based OBC to SEU. In particular Tierra-based OBC *with* the proposed recover crossover can maintain and evolve the correct program in high SEU rate in comparison with Tierra-based OBC *without* the recover crossover; and (3) even if the correct PIC programs are broken, Tierra-based OBC *with* the recover crossover can recover the correct program by modifying the broken one. In particular the differential recover crossover is high potential to recover the correct program, which only replaces different instructions between a broken program and a mated one.

What should be noticed here is that the experiment is conducted on the simulation, but not on the actual SEU environment. Therefore the effectiveness of Tierra-based OBC should be investigated on the actual space mission. For this issue, it is also important direction to construct Tierra-based OBC on hardware component and to verify the feasibility on it. In addition, the following future researches must be tackled: (1) an improvement of the recover crossover to recover a correct program more accurately; and (2) an exploration of the robustness to the bit inversion in a core process of Tierra-based OBC, for example, an error of the evaluation process or the reaper process.

ACKNOWLEDGMENT

This work was supported by Grant-in-Aid for JSPS Fellows Grant Number 249376.

REFERENCES

[1] Reed Business Information, “EDN Japan Feburary issue,” EDN Japan, 2005.

[2] N. Ikeda, H. Shindou, Y. Iide, H. Asai, S. Kuboyama, S. Matsuda, “Evaluation of the Errors of Commercial Semiconductor Devices in a Space Radiation Environment,” The transactions of the Institute of Electronics, Information and Communication Engineers. B, Vol. 88, No. 1, pp. 108-116, 2005.

[3] J. Justesen, T. Hoholdt, A Cource In Error-Correcting Codes, European Mathematical Society, 2004.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 4, July 2015

[4] K. Nonami, K. Takadama, "Tierra-based Space System for Robustness of Bit Inversion and Program Evolution," In SICE, 2007 Annual Conference, pp. 1155-1160, 2007.

[5] Tomohiro Harada, Masayuki Otani, Hiroyasu Matsushima, Kiyohiko Hattori, Hiroyuki Sato, Keiki Takadama, "Robustness to Bit Inversion in Registers and Acceleration of Program Evolution in On-Board Computer," Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 15, no. 8, pp. 1175-1185, 2011.

[6] Tomohiro Harada, Masayuki Otani, Hiroyasu Matsushima, Kiyohiko Hattori, Keiki Takadama, "Evolving Complex Programs in Tierra-based On-Board Computer on UNITEC-1," Proceedings on International Astronautical Congress (IAC), 2010 61st World Congress on, 2010.

[7] T. S. Ray, "An approach to the synthesis of life," Artificial Life II, Vol. XI, pp. 371-408, 1991.

[8] T. Kimezawa, T. S. Ray, "Artificial Life System Tierra," ATR Human Information Processing Research Laboratories, 1999.

[9] J. Koza, Genetic Programming On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

[10] Michrochip Technology Inc., "PIC10 F200/202/204/206 Data Sheet 6-Pin, 8-bit Flash Microcontrollers," Microchip Technology Inc., 2007.

[11] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Boston, Ma, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[12] D. Jong, K. Alan, "An Analysis of the Behavior of a Class of Genetic Adaptive System," PhD thesis, Department of Computer and Communications Sciences, University of Michigan, 1975.

[13] UNISEC, "UNITEC-1", http://www.unisec.jp/unitec-1/, 2009. (available 2015/06/25)

[14] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," Soviet Physics Doklady, Vol. 163, No. 4, pp. 845-848, 1965.

[15] M. F. Brameier, W. Banzhaf, Linear Genetic Programming, Springer, 2007.

AUTHOR BIOGRAPHY

Tomohiro Harada received his B.S. and M.E. degrees from the University of Electro-Communications, Japan, in 2010, and got Doctor of Engineering Degree from the University of Electro-Communications. He currently works at Department of Human and Computer Intelligence, College of Information Science and Engineering, Ritsumeikan University as an assistant professor from 2015.His research interests include evolutionary computation, multiagenet system, reinforcement learning, and learning classifier system. He is a member of the Japan Society for Evolutionary Computation.

Keiki Takadama received his M.E. degree from Kyoto University, Japan, in 1995 and got Doctor of Engineering Degree from the University of Tokyo, Japan, in 1998, respectively. He joined Advanced Telecommunications Research Institute (ATR) International from 1998 to 2002 as a visiting researcher and worked at Tokyo Institute of Technology from 2002 to 2006 as a lecturer. He moved to The University of Electro-Communications as associate professor in 2006 and is currently a professor from 2011. His research interests include multiagent system, distributed artificial intelligence, autonomous system, reinforcement learning, learning classifier system, and emergent computation. He is a member of IEEE and a member of major AI- and informatics-related academic societies in Japan.