Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 17 (2011) 497 – 504

Procedia Engineering

www.elsevier.com/locate/procedia

The 2nd International Symposium on Aircraft Airworthiness (ISAA 2011)

A Study on Compiler Selection in Safety-critical Redundant System based on Airworthiness Requirement

CHANG Wei, BAO Xiaohong, ZHAO Tingdi

School of Reliability and System Engineering, Beihang University, Beijing, P.R.China, 100191

Abstract

The dependability of compiler would directly affect the quality of software because it can directly produce object code. At the same time, compiler diversity is an important part of software diversity design in a redundant system, which could not only help avoid common defects from compilers but also to find defects in source code. This paper proposes a method for compiler selection in safety-critical embedded redundant system based on airworthiness requirement and the principle of software diversity. A case on compiler selection in tri-redundancy FCS (flight control system) is given in the end.

Keywords: airworthiness requirement; redundant system; compiler dependability; compiler diversity; compiler selection

1. Introduction

The complex and ubiquitous software supports the architecture of global economy today, high-level language and compiler are just the cornerstone of the software. Software is required for higher reliability and safety with the growing proportion of function realized by software in safety-critical field such as aerospace. High-quality software requires strict control for every stage in software life cycle. Usually, a compiler is required to "translate" source code to executable code for high-level language. So the dependability of compiler is crucial for the safety of software system.

Present compiler research has focused on the compiler optimization [1], compiler formal verification [2] [3] and other related designs. People usually tend to pay much attention to a certain compiler feature such as the compatibility for hardware platform, whether a specific type of high-level language is supported, compiler efficiency and so on when selecting compiler. There are few systematic approaches for compiler

* Corresponding author. Tel.: 010-82339821; fax: 010-82339821.
E-mail address: changweibuaa@126.com.

1877-7058 © 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2011.10.060

selection. Software Engineering Research Centre of Carnegie Mellon University gave research on compiler selection and evaluation for Ada language [4], the significance, guidelines, criteria and related methods of Ada compiler selection are given. [5] [6] gave research for factors to be considered and related technology when selecting the Ada compiler. But there is a lack of research for compiler selection of other languages. The method for compiler selection in safety-critical redundant system based on airworthiness requirement has not been seen.

Redundant design is usually adopted to improve system safety, the corresponding software should also be dissimilar. Airworthiness standard DO178B [7] points out that the development of software development environment such as compiler should meet the DO178B development process, but the process for compilers in redundant system can be adjusted according to the former certification process for each compiler when dissimilarity could be ensured. But research on how to select compilers in safety-critical redundant system based on DO178B requirement has not been seen.

This paper proposes several ways for assuring the dependability of compiler according to the relevant airworthiness requirement from standard DO178B and gives related study on software diversity design and principle of compiling, the basis for compiler diversity evaluation and corresponding verification ideas are put forward. A relative explicit compiler selection method for safety-critical embedded redundant system is proposed based on airworthiness requirement through the above analysis. Finally, a case study on compiler selection for tri-redundancy FCS is given according to the above research.

2. Basic Theories

2.1. Some Discussions On Compiler

Compiler is the programme translating “high-level language” into “machine language (low-level languages)”. The main work flow of a modern compiler is as follows: source code → pre-processor →compiler →assembler →object code →linker executable code.

This compiling process can be divided into front end and back end. The front end which mainly depends on the source language consists of lexical analysis, syntax analysis, semantic analysis, intermediate code generation and some optimization operations. This part is where the analysis strategies of compiler are included. The back end which depends on the target machine consists of object code generation, some related error handling measures and symbol tables. This part is where the optimization strategies different from other compilers are included [8]. There may be the following defects based on the above: First, strict "semantic equivalence" for every process in the front end is difficult to achieve, which may lead to error in the output. Secondly, it is necessary to take fully into account the characteristics of the target machine in the back end. Program initialization, error detection and machine exception handling could not be traced to the source code from object code and difficult to detect. These factors may cause further defects from compiler. In addition, the error handling design in compiler will determine whether the compiler could fully find the defects in source code.

Nowadays people usually focus on testing and verification of source code, but the relevant test and verification work would not be so meaningful if the correctness of the compiler could not be proved. What’s more, the error handling design of compiler has a great impact on the test. There are representative researches on verification of compiler as follows: CompCert compiles Clight (subset of C language) to PowerPC assembly code, using the Coq auxiliary proof tools to write the compiler and complete its proof of correctness [2]. Writing a target machine simulation and a compiler in ACL2 gives the opportunity to prove the correctness with the support of automatic reasoning [3]. But these methods are mainly at theory stage and could not be widely used in engineering for the moment.

2.2. Principle Of Software Diversity

It is proved that the reliability of redundant system depends not only on the software reliability of each version, but also the dissimilarity between them. Index for measuring dissimilarity is the degree of software diversity, which is the probability of different software versions causing different failure modes [9]. Professor Avizienis and his team gave the following Table 1 by combining the experimental study with other people's research.

Table 1 Qualitative analysis of dissimilarity [10]

	Implementers	Languages	Tools	Algorithms	Methodologies
spec	higher	higher	lower-	higher+	lower
design	higher+	lower	lower	higher+	higher
coding	higher+	higher+	lower	higher	higher
testing	lower	lower-	higher	lower	higher+

As can be seen from the table: Different developers, different languages, different software algorithms all have a great impact on software diversity. Different tools or methods in test stage could result higher dissimilarity. Compiler diversity in redundant system could help avoid common defects caused by compilers and software diversity to be better assured.

3. Method For Compiler Selection In Safety-Critical Embedded Redundant System

3.1. Airworthiness Requirement

DO178B points out that "The software development environment is a significant factor in the production of high quality software. During the software planning process, the software development environment should be chosen to minimize its potential risk to the final airborne software."

According to Qualification Criteria for Software Development Tools in DO178B, "If a software development tool is to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of airborne software. If multiple-version dissimilar software is used, the tool qualification process may be modified, if evidence is available that the multiple software development tools are dissimilar. This depends on the demonstration of equivalent software verification process activity in the development of the multiple software versions using dissimilar software development tools". The applicant should show that [7]:

- Each compiler was obtained from a different developer.
- Each compiler has a dissimilar design.

The following chapter will give the compiler selection method when source language, development platform and target platform requirements have been satisfied.

3.2. Principle For Compiler Selection In Safety-critical Embedded Redundant System

There are two aspects for compiler selection requirement in safety-critical embedded redundant system according to DO178B. Firstly, each compiler should be dependability in some degree in case not to introduce defects and the defects in source code could also be found as fully as possible. Secondly, compilers should be dissimilar to assure the software diversity and avoid common defects caused by compilers. This paper proposes the following principles according to the above requirement:

1) For the selection of each compiler, there should be sufficient evidence or instructions to ensure the dependability of compilers has achieved the consistence level of dependability with the software product.

2) For compiler selection in redundant system, dissimilarity should be ensured after the first principle has been met, and there should be related analysis and verification for dissimilarity.

3.3. Means To Assure The Dependability Of Compiler

1) Formal Verification

Formal verification is the most powerful method to prove the correctness of compiler. The present researches for formal verification method are mainly at theoretical stage, but it would be the first choice for compiler verification once this method could be maturely used. It is also the recommended verification method in DO178B.

2) Authority and Standard Certification

Qualification Criteria for Software Development Tools in DO178 points out: "If a software development tool is to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of airborne software." That means the compiler development process should meet the process defined in the standard. Dependability of compiler could be guaranteed in some degree if the development process achieves that required in related standard based on the above thought, the standards should be derived from the relevant field compiler applied.

3) Product Service History

DO178B points out that "If equivalent safety for the software can be demonstrated by the use of the software's product service history, some certification credit may be granted." This method is not only based on the service times of the product but also the application field, configuration management and so on. The specifications are as follows according to DO178B and related research [11]:

(1) There should be a large number of known users using the product with known frequency considering the stability and maturity of the software.
(2) The consequences should be detailed analyzed when there is change for applied field.
(3) The consistency of the product historical environment should be evaluated.
(4) There should be required enforcement mechanisms for error reporting and logging considering the experimental error rate and the history of product.
(5) All the expected deviation must be documented and can be checked considering the validity of the report on activities.

The above requirement could be summed up as two aspects: First, the compiler should have been widely used in similar field and there should be enough documents about the usage. Second, the change for the compiler should be strictly controlled and evaluated.

3.3.1. Evaluation Of Compiler Diversity

Compiler diversity could not only improve the dissimilarity between different software versions and avoid common defects from compilers but also could help fully find defects in source code. So to ensure the dissimilarity between compilers is important in redundant system.

According to the software diversity principle in charpter2.2, software diversity mainly comes from Implementers, Languages, Tools, Algorithms and Methodologies. Usually, development institution would decide the Implementers, Languages, Algorithms and Methodologies. So the different institutions have a really important effect on compiler diversity. We should investigate the development institutions and main designs when evaluating the compiler diversity.

The compiler selection method in safety-critical redundant system is shown in table 2 as follows based on the above discussion.

Table 2 Compiler selection method in safety-critical redundant system

	Dependability	Compiler Diversity	Priority
Formal Verification	Proving the correctness of compiler by formal verification.	1. Compilers are developed by different institutions. 2. Each tool has a dissimilar design	**First**
Authority and Standard Certification	The compiler should have been certified by authority and standard.		**Second**
Product Service History	1. Investigate the usage of the compiler. 2. Check whether the conditions could be met.		**Third**

3.4. Verification And Analysis

Because most source code of compilers are not or not wholly open to the public. Compiler diversity verification and analysis are mainly by "black box" method in which the same source code is compiled by different compilers and the results are compared and analyzed. This paper gives the following verification ideas:

1) Based on the statements in the source code

Selecting the safety-critical and frequently used statements according to the actual situation to compile. Safety-critical refers to statements which would be used in safety-critical functions. Frequently used statements refer to statements which would be frequently used according to the designer. Each statement could be compiled when conditions permit. Analyze the result and compare the binary code generated in instruction sequence, instruction length, registers allocation and then sum up the dissimilarity. Debug level and optimization options could be set to fully analyze the dissimilarity of different compilers.

2) Based on function module

Further analysis against key program module would be taken to further define the compiler diversity. Means are similar with the first part. In this process, we could still set debug level, compiler optimization if necessary to fully analyze the dissimilarity of different compilers.

4. A Case Study On Compiler Selection Of Tri-Redundancy Fcs

Redundancy technology is very important for improving the reliability and fault tolerance ability of flight control system. System reliability index, increased costs and volume by redundancy and basic system reliability should be considered and weighed to determine the number of redundancy. The structure of tri-redundancy FCS is shown in Diagram 1:

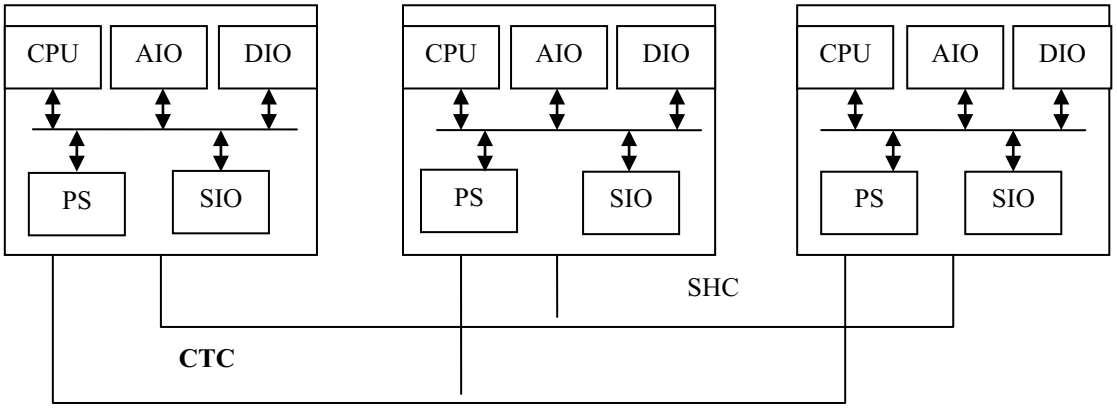

Diagram 1: Structure of tri-redundancy FCS[12]

SHC: synchronization handshake circuit
CTC: cross transmission circuit

The three channels are exactly the same and each channel includes the central processing unit (CPU), input output interface (DIO), analogy processing module (AIO), serial transceiver module (SIO), power supply module (PS) and other functional modules [12]. Every channel would receive external data when FCS is working and the most satisfying one would be selected as the control computer through voting mechanism.

The source code in this paper comes from a safety subset of c language. Compilers to be selected are: C optimizing compiler, Diab Compiler and GCC. The investigation for the three compilers is in the following Table 3.

Table 3 Investigation for compilers to be selected [13][14]

Compiler name	Compiler dependability analysis		Compiler diversity analysis	
	Authority and standard certification	Product service	Optimization strategy	Development institution
C optimizing compiler	Conforming fully to ANSI Standard C (ISO/IEC 9899 and FIPS PUB 160).	Products have been used in BAE Systems, Boeing Company, CMC Electric, EADS, General Electric and so on.	✓ The Target-Specific Optimizer applies additional optimizations such as peephole optimizations and multiple issue instruction pipeline scheduling. ✓ Green Hills Software is uniquely qualified to support the largest number of MISRA C rules through its optimizing compilers and Advanced Run-time Error Detection.	Greenhills，who has been the leader in embedded optimizing compilers for over 25 years.
Diab Compiler	Certified for POSIX PSE52. Provides compliance with the latest ANSI and ISO standards.	Products have been used in automotive, aerospace, weapon system and so on.	✓ Whole program optimization. ✓ Profile-driven optimizations. ✓ Multiple debugging options ✓ Position-independent code (PIC) and data (PID). ✓ Absolute addressing from C and assembly. ✓ Run-time error checker. ✓ Instruction-set simulation.	Wind River
GCC	Issued by the GPL and LGPL licenses.	GCC could compile programs for variety of hardware platforms, its efficiency is generally 20% to 30% higher than the average.	GCC uses the majority of machine-independent compiler optimizations such as common sub expression elimination, constant merging. It also uses some optimizations based on machine description such as instruction combining, instruction scheduling and so on.	Andy Tanenbaum

We give the analysis for the three compilers in Table 4 based on the method in charpter3.

Table 4 Analysis of consistency with selection method

Compiler name	Verification of compiler dependability		Compiler diversity analysis	
	Product Service History	Authority and Standard Certification	Development Institution Diversity	Design Diversity
C optimizing compiler	√ √ √	√ √	√ √	√ √
Diab Compiler	√ √ √	√ √		
GCC	√	√		

Note: √ indicates compliance level, it is more compliant when there are more √. The compliance with requirements for "Product Service History" need to be further analyzed by related users. There should also be further analysis of dissimilarity with ideas proposed by the paper to get the qualitative and quantitative result. We can see C optimizing compiler and Diab Compiler have both been widely used in safety critical areas and have been verified by different authorities or standards. Dependability of them could be assured in some degree. GCC has not been widely used in safety critical areas or verified by related standards, so its suitability for flight control system should be further analyzed. The three compilers are developed by three different institutions and there optimization strategies and design ideas are also different. So the dissimilarity could be assured in some degree. The specific dissimilarity need to be further analyzed by users according to ideas in the paper.

5. Conclusion

This paper puts forward an executable compiler selection and verification method against safety-critical embedded redundant system through combining guidelines in airworthiness standard with compiling theory, compiler verification technology and software diversity principle. The method could be the explicit base for compiler selection in engineering application. A case study for compiler selection in tri-redundancy FCS is given according to the above discussion. The case would be further analyzed and verified in the following work.

Acknowledgement

The authors thank Professor Jin for his meaningful advising and Shaojun Li for his positive involvement of the discussion.

References

[1]. Timothy M. Jones, Michael F.P. O'Boyle. Evaluating the Effects of Compiler Optimizations on AVF. University of Edinburgh, UK.

[2]. Martin Strecker. Compiler Verification for C0. Universit´e Paul Sabatier, Toulouse, France.

[3]. Thomas Würthinger, Linz, Juli.. Formal Compiler Verification with ACL2. Institute for Formal Models and Verification, 2006.

[4]. Nelson H. Weiderman. Ada Adoption Handbook: Compiler Evaluation and Selection. Carnegie Mellon University, Software Engineering Institute, 1989.

[5]. Robert E. Davis. The Selection of an Ada Compiler for a Real-time Embedded Avionics Application. ACM New York, NY, USA, 1989.

[6]. A. Tetewsky, R. Racine. Ada Compiler Selection for Embedded Targets. ACM New York, NY, USA, 1987.

[7]. SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION. RTCA, Inc.

1140 Connecticut Avenue, Northwest, Suite 1020 Washington, D. C. 20036-4001 U.S.A, 1992.12.1.

[8]. Suqin Zhang, Yingzhi Lv, Weidu Jiang, Guilan Dai. Principle of compiling. Beijing: Tsinghua University Press, 2005.

[9]. Wenbin Yao, Xuguo Zong, Xiaozong Yang. Fault-tolerant Software Development Based on Fuzzy Reusable Libraby. Journal of computer research and development, 2002.12, 39(12).

[10]. Avizienis, A, L. Chen. On the Implementation of N-version Programming for Software Fault Tolerance during Execution. In *Proc. IEEE COMPSAC 77*, pages 149–155, November 1977.

[11]. Richard Barry. How to Verify Your Compiler for Use in IEC 61508 Safety-critical Applications, 10/30/2007 11:53 PM EDT.

[12]. Jihui Pan[1, 2], Xiaolin Zhang[1]. Design and Realization of Treble-Redundancy Management Method of Flight Control System. Computer measurement & control, 2010, 18(2).

[13]. http://www.ghs.com/products/optimizingC++EC++Compilers.html Greenhill official website

[14]. http://www.windriver.com/products/development_suite/wind_river_compiler/ wind rive official website.

[15]. Yanxiang He, Tao Liu, Wei Wu. Research on the Key Technology of Trusted Compiler. Computer Engineering and Science, 2010, 32 (8).

[16]. John Hannan, Department of Computer Science University of Copenhagen. Frank Pfenning, Carnegie Mellon University. Compiler Verification in LF.

[17]. Xavier Leroy. Formal Verification of a Realistic Compiler. Communications of the ACM, 2009, 52(7):107-115.

[18]. NASA Software Safety Guidebook. National Aeronautics and Space Administration.

[19]. CPU Source Code Analysis and Chip Design and LINUX Transplantation. Jili Ni, Xi Chen, Hui Li. Electronic Industry Press, 2007.

[20]. Mustafa E. A. Ibrahim, Markus Rupp. Compiler-Based Optimizations Impact on Embedded Software Power Consumption.

[21]. Han Lee. Understanding the Behavior of Compiler Optimizations. Department of Computer Science, University of Colorado, Boulder, CO 80309, U.S.A.