

Design of software error detection system based on SPARC V8 and research on the key technology

Chunmei Huang^{1*}, Chunmao Jiang¹, MingCheng Qu²

¹*School of Computer Science Technology and Information Engineering, Harbin Normal University, Harbin, Heilongjiang 150025*

²*School of Computer Science and Technology, Harbin Institute of Technology*

Received 12 June 2014, www.tsi.lv

Abstract

This paper analyses the key issues confronted when SIHFT is implemented on the SPARC V8 platform, gives the algorithm to solve the problem and the corresponding technology solutions. Software error detection technology system was designed based on SPARC V8, and software signature control flow error detection technology was implemented, the system is based on the architecture of SPARC V8, uses software signature control flow error detection technology and copy instruction error detection technology as the prototype, it's a software system which detects transient faults induced by space radiation and was developed through research, analysis and transformation, with availability, modifiability, portability, maintainability, readability, scalability and other features. The error detection coverage rate of software error detection technology suitable for target platform was tested through simulation experiments. The result data of experiments conducted in the emulator TSIM shows that on the basis of given average performance overhead, the system had high error detection coverage rate when brought in register injected fault and memory injection fault. This proved the SIHFT technology is feasible and effective.

Keywords: SIHFT, architecture, SPARC V8

1 Introduction

SIHFT (Software Implemented Hardware Fault Tolerance) technology is a Software fault-tolerant technology, which is developed by the Stanford university centre for reliability calculation for ARGOS projects, through the method of software to detect and correct hardware transient fault caused by radiation. It is comprised of a software implemented EDAC technology, copy instruction error detection technology (EDDI), Control Flow Checking by Software Signatures (CFCSS) and Watchdog timer. Due to the majority of storage has implemented hardware EDAC at present, while watchdog timer belongs to the category of hardware research, so in this article, we are dedicated to research two core technologies which are Control Flow Checking By Software Signatures(CFCSS) and copy instruction error detection (EDDI).

SIHFT is still in the preparatory stage in domestic, has not yet entered the stage of practical development [1, 2]. In foreign countries, space research institutions and academia carried on the thorough research on this question and some research results were obtained [3, 4]. The United States ARGOS (the Advanced Research and Global Observations Satellite) Satellite did an in-orbit experiment about the main method of the commercial device resist radiation and the SIHFT error detection coverage reached more than 99% [5].

Bound by design technology and production technology and other aspects, there is a large gap between the domestic research on microprocessor and application level with foreign. Generally speaking, the microprocessor performance is low and relies on import leading to cannot satisfy the spacecraft to achieve more control autonomously and data processing requirements. What's more, in the field of high technology our countries are blockaded by foreign countries. Therefore, it is very difficult to obtain radiation-hardened hardware from abroad, so the software fault-tolerant technology has a special significance in the development of China's space industry. This paper analyses the key issues confronted when SIHFT is implemented on the SPARC V8 platform, gives the algorithm to solve the problem and the corresponding technology solutions. And on this basis, design a software error detection technology system based on SPARC V8.

2 Fundamental Concepts

2.1 SOFTWARE SIGNATURE CONTROL FLOW ERROR DETECTION TECHNOLOGY

Software signature control flow error detection technology is a kind of do not need a watchdog processor or other hardware accessory pure software control flow error detection technology, used to test the execution of a program control flow errors which caused by the

* *Corresponding author* e-mail: hsdrose@126.com

radiation. The primary idea is to divide program into basic blocks, and next construct the program flow chart, for each basic block gives a unique signature in advance, which called a compile time signature, and compute the value which is the result of function f while the input is the signature of the father's basic block and the son's, and later save the value into the basic block of the son, this information is compiled into the program at compile time. When the program executed, the control flow for each run to a new node will generate a runtime signature G according to the information, which is compiled into the program at compile time. Subsequently compare the runtime signature G and compile time signature, if they are equal, there are no program control flow errors occurred; if unequal, there are the program control flow errors have taken place, then turn into fault handler. When multiple control flow inflow the same node, runtime adjusted signature D is required to test control flow detection.

Software signature control flow error detection technology used $V = \{v_1, v_2, \dots, v_n\}$ represents the basic block of a collection of nodes, used $E = \{b_{rij} \mid b_{rij} \text{ is the edge from node } v_i \text{ to another node } v_j\}$ represent a collection of edges of control flow between the basic blocks, so a program can be expressed by a program flow chart map $P = \{V, E\}$. Node v_j is included by the succeed collection $suc(v_i)$ of node v_i , if and only if $b_{rij} \in E$. Node v_i is included by the precursor collection $pred(v_j)$ of node v_j , if and only if $b_{rij} \in E$. If during the program executes, the edge b_{rij} exist but b_{rij} is not in E , says edge b_{rij} is illegal. Illegal side shows that the control flow error happened. If one node v received more than two control flow, referred to as the fan-in node, which has more than one node in the $pred(v)$.

2.2 COPY INSTRUCTION ERROR DETECTION TECHNOLOGY

Copy instruction error detection technology refers to the repeated calculation by "copy" instructions to error detection. The basic idea is to use different registers and instruction of the variable "copy" assembly language source program. The original instruction of the assembly language source program is called main instructions; the "copy" instructions added to the source program is called shadow instruction. General registers and memory units are divided into two groups to correspond to the main instructions and shadow instructions. The homologous registers and memory units should always have the same value, if the value in a pair of register corresponding to the main instructions and the shadow instructions becomes not equal because of the instantaneous error, therefore, compare values in this two register can detect the errors. In the copy instruction error detection technology, introducing comparison order to compare the value of the corresponding register, if they are an unequal then call error handler.

3 System Design

Software error detection technology system to after modification of the GCC cross compiler generates a fixed part of registers the assembly language source as an input, and to the error detection function of an assembly language program as an output. As shown in figure 1, it is the top chart of the system data flow graph, depicts that the developed system data exchange relations with the surrounding environment.

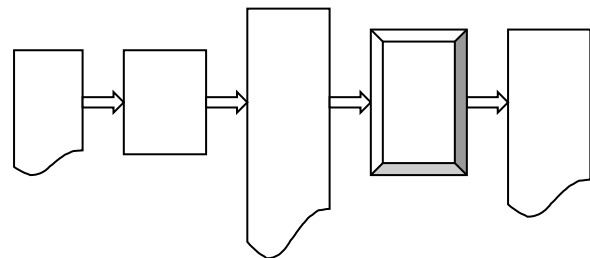


FIGURE 1 The system model of software error detection system

By analysing software signature control flow error detection technology and copy instruction error detection technology, the processing steps of the system should be, first of all, needed to divide the assembly language source program into basic blocks, based on basic block generate the program flow chart, and next according to the structure of the program flow chart in each basic block's head to add software signature check instructions. For each basic block, no stores basic block is divided at first, subsequently generates shadow instruction and constructs a dependency graph, finally carries on the instruction scheduling. Therefore, we will functions divided into basic blocks; software signature technology processing and copy instructions deal with three parts, as shown in figure 2.

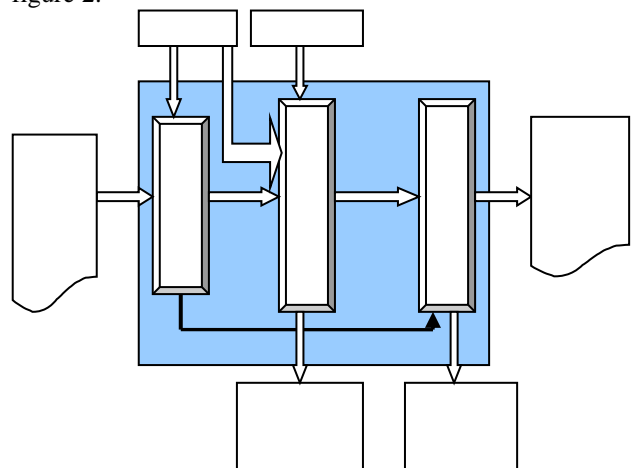


FIGURE 2 The function of software error detection system

System has three input files, which are the assembly language source program; instruction opcode grouped information file and register division method file. Considering the modifiability and scalability of the system, will be mainly related to the specific platform information about an input to the form of a configuration

file. Instruction opcode grouped information file storage system related instruction operation code and the information about the operation code group. Register division method file storage SPARC V8 registers the use method includes the corresponding relation of main registers and shadow registers; software signature control flow error detection technology used registers, the matching relation of status register and a certain register. After the assembly language source program divided into basic blocks, with basic block's information and the configuration file input to the software signature part processing, therefore, obtained the assembly language program, which has the function of control flow check. Finally, delivery the signature check instructions and assembly language source program of each basic block to copy instruction's parts processing, obtained the assembly language program which has the function of copy instructions check and control flow check.

After dividing the basic block, the data flow passed to copy instruction's error detection technology is the information of basic block. After software signature error detection technology processing, the data flow passed to copy instruction technology is with signature check instructions information of basic block. These two types of information transmitted is intended to enable the two technologies can be used independently; the system is easy to cut.

In summary, the software error detection technology based on SPARC V8 system architecture is shown in figure 3. System consists of three modules. Basic block partition module called label mapping, function return statement processing, determine the basic block entry and determine the basic block exports. Software signature technology processing function subdivided into control flow graph generation; add software signature check instructions and optimization. Copy instruction's technology processing called no store basic block division, produce the shadow's instructions, and construct a dependency graph and instruction scheduling. Command recognition module almost is called by all modules (not noted on the picture). Each module implements must follow high cohesion and low coupling principles. System according to the level gradually refined until the bottom of the module.

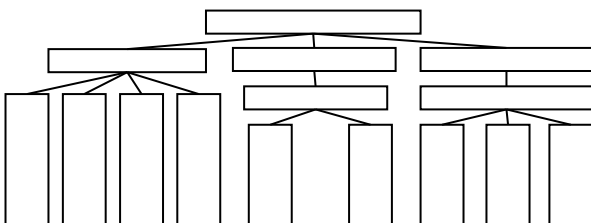


FIGURE 3 The structure of software error detection system

4 The key technology research

4.1 INSTRUCTION RECOGNITION

Function of the instruction recognition modules is given an instruction; identify the opcode of this instruction. In basic block division, shadow instruction generation and other process modules, all need to be able to correct recognition instruction, thus instruction recognition module is called by the upper most modules.

In order to improve the efficiency of instruction recognition, all the opcode saved into the hash table in alphabetical order, hash function: hash address = first address of the hash table + opcode field first character ASCII value - 97. During the recognition, first calculate the hash address according to the opcode field first character, second followed by matching opcode of the overflow table, if the overflow table has a prefix of the encoding, depending on the opcode field length matching recognition.

All the instruction opcode related to system and grouping information are stored in the instruction opcode grouping information files, the instructions grouped by category (data processing, data transmission and control flow), syntax format and the processing method of shadow instruction.

4.2 BASIC BLOCK DIVISION

Basic block is the largest collection of the statement's sequence for the program, which can be executed sequentially; it has only one entry statement and export. Basic block is an important concept in the control flow and data flow analysis to the compiler theory. Basic block division is the preamble step to construct a program flow graph. Program flow graph is the program's control flow semantic representation; each node represents a basic block, and edge represents control flow between basic blocks. The division of basic block in compiler theory and based on basic block division code optimization is carried out on the intermediate code, while the software error detection technology basic block division was conducted on the assembly code. The abstract intermediate code of a basic block division algorithm is applied to the assembly language on basic block division, and the handling of the specific issues related to an instruction set is needed for basic block division problem. The intermediate code of the basic block division algorithm from here, assembly language basic block division algorithm is given.

4.2.1 Basic block division algorithm of assembly language program

Input: assembly language source program;
Output: The information table of Basic block
(1) Label mapping

Establish one-to-one relationship between the label and its line number.

(2) Sub functions return statement processing

Set up corresponding relationship between function return statements of assembly language program and the statement to jump.

(3) Determine the entry of basic block

After program entry, the line numbers in the assembly instructions to fill entrance set.

For the part of operation code, which is B/CALL/TA/FBA/CBA or B + A unconditional jump instruction, the label will have to jump map to line number, fill the assembly instruction line number which after the line number just mentioned to the entrance set .

For the function return statements, fill the line number followed by the line number of assembly instruction, to which is function return statement jumped to the entrance set.

For conditional transfer instruction, assembly instruction line number after this instruction fills to the entrance set. Then let the conditional transfer instruction label mapping line number, fill the assembly instruction line number followed by this line number to the entrance set.

(4) Determine the export of basic block

Sort the entrance set by the line number ascending, S_1, S_2, \dots, S_n . For $S_i, i = 1, 2, \dots, n - 1$, will be the first transfer instruction's line number after S_i into export sets, if there is no transfer instruction, will be the last one before entrance S_{i+1} assembly instruction line number fill in the exports set. For S_n , will be the first transfer instruction's line number after S_n into export sets

4.3 CFCSS INSTRUCTION GENERATION

4.3.1 Generation algorithm of signature error detect instruction

Input: Program flow graph

Output: The information table is comprised of error detect instruction

Explanation: Algorithm is to define the composition of each node in the flow diagram error detect instructions. The error detect instructions are divided into instruction related to G as "G=G \oplus dj", "br G \neq sj error", "G=G \oplus D" and related to D such as "D=0000" or "D=S $_i$ \oplus S $_m$ ". There are all or part of the instructions included by one node. For one certain node, the position of G_FLAG represents the composition of error detect instruction related to G has been defined; the position of D_FLAG represents whether error detect instruction related to D has been defined in the error detect instructions. When it detects a situation that is beyond the scope of processing a message will be sent out and quit.

Algorithmic process:

(1) Initialization, count the data which the subsequent steps of the algorithm needed. Including, the

number of each node's precursors, if the number of precursor is greater than 1, mark for the fan-in node; the number of each node's successor and the number of fan-in node in the successor, mark whether each successor is fan-in node.

(2) Grants each node of the program flow graph a unique signature.

(3) Sort all the nodes by the number of fan-in node in successor descending.

(4) For each node i in the list after sorting

If (no precursor node) //head node

{ Assembly code is "mov GSR, Si"; location is G_FLAG;

If there is no successor node, there is no instruction of "D=Si \oplus Sj" in the mark error detection instruction.

}

else

//both have precursor and successor node

{ if (successor node j is non-fan-in node)

{ calculate dj=Si \oplus Sj, there is instruction of "G=G \oplus dj" and "br G \neq sj error" in the mark error detection instruction, there is no instruction of "G=G \oplus D", location is G_FLAG.

If there is no fan-in node in the successor of node, there is no instruction of "D=Si \oplus Sj", location is G_FLAG.

}

else if(successor node j is fan-in node)

{ if(i's D_FLAG has set)

{ if(there is a fan-in node in it's successor node G_FLAG has not set)

tip exit;

else go to(4)(handle next node)

}

else

{ if(there is a D_FLAG had positioned pioneer node ,which is node i's successor's fan-in) tip exit;

else

for each fan-in node j: calculate dj=Si \oplus Sj, there is instruction of "G=G \oplus dj", "br G \neq sj error" and "G=G \oplus D" in the mark error detection instruction, location is G_FLAG.For i, instruction of "D=0000" in the mark error detection instruction ,location is D_FLAG. For each non- precursor node of i, there is instruction "D=Si \oplus Sm" in the mark error detection instruction, location is D_FLAG.

}

}

}

4.3.2 The optimization of the signature error detect instructions

The function of optimization of the signature error detected instructions is to optimize the signature error detect instruction, to reduce the cost of add signature error detect instructions. According to software signature

error detection technology optimization algorithm, under the need in some basic blocks first do not conduct the compare of error detection. If there is an error of control flow, error will be according to the control flow being transferred to the back, until detected in once compared. The advantage is to reduce the comparison instruction, improves the application performance; the shortcoming is to accumulate the error backward, once found errors, code needs to be executed again whose length gets greater during restoration.

4.4 INSTRUCTION GENERATION OF EDDI

Copy instruction error detection technology through the "copy" instruction to conduct repeated to calculate to detect transient faults. In this technique, general-purpose registers are divided to two groups, called primary registers group and shadow registers a group. The values of the primary register with the corresponding shadow register always are same. Under the SPARC V8 architecture, some registers do not have enough corresponding shadow registers, so that bringing shadow instruction generation method difference with the original technology method. Learned by the ideas of SWIFT algorithm, in the implementation, not have the copy on storage instructions and memory, which does not implement the memory test.

4.5 INSTRUCTION SCHEDULING

Instruction scheduling is the method of a sort for main instruction and shadow instruction to improve the error detection coverage rate and reduce the execution time in the instruction error detection technology. In order to ensure the instruction sequence after scheduled and without scheduling instruction sequence on the semantic equivalence, instruction scheduling be based on instruction dependency relationship graph. In copy instruction error detection technology, the dependency relationship graph is instructions as the vertex and the dependency relationship as to the edge of the directed graph.

The dependency relationship between instructions is must satisfy the constraint conditions during instruction scheduling. Reference [6] is pointed out shortcoming of the definition [5] of the original dependency relationship, according to the principle of compilation techniques [7, 8], dependency relationship between instructions was redefined as: the instruction j after the instruction i , instructions j depend on the i when the instructions i and j is really relevant / anti-related or output-related. In the no storage basic block when the last instruction is store instructions or transfer instructions, not to participate in instruction scheduling. Thus, in no storage basic block, instruction which participated in scheduling could only happen in the register data-related. In establishing instruction dependency graph, only need to according to

the register data-related to establish dependency relationship.

In the copy command error detection technology, goal of scheduling instruction is to maximize the error detection coverage and minimize the execution time. Under the superscalar architecture, instruction scheduling can reduce the program execution time [9, 10]. Assembly line of SPARC V8 processor is five levels; it has no correlation [11] in every stage, so the instruction scheduling will not reduce the instruction execution time. Thus, the goal of instruction scheduling is how to maximize the error detection coverage. According to the instruction scheduling algorithm, on the premise of meet the dependency graph, required to execute commands to the i , number of primary instructions and shadow not equal. On the premise of meet the dependency graph, priority scheduling primary instruction, when perform to a certain instruction, the number of primary instructions and shadow are greatest, therefore, the error detection coverage is the largest. In no stored basic block does not participate in instruction scheduling is storage instruction located on the last of no stored basic block and transfer instruction and so on. Therefore, before the instruction scheduling, it first determines the boundaries of scheduling and then establishes a dependency graph, finally perform scheduling.

Instruction Scheduling Algorithm:

input: Instruction list of the no stored basic block

output: Instruction list after scheduling

Algorithmic process:

- (1) Determine the boundaries of scheduling
- (2) Establish dependency graph
- (3) Perform scheduling according to dependency graph:

The node which precursor is 0 arranged in two lists by the main command and shadow command.

while (number of main command in the list > 0 || number of shadow command > 0)

{if (number of main command in the list > 0)

{Select the line number the smallest primary instruction scheduling. In the original list to delete the node, added to the end of the list of instructions to be scheduled, number of main command in the list reduce one; Subsequent penetration of this command reduce one, add the nodes which penetration is zero to the end of the list of main and shadow command by main command and shadow command. Calculate the number of list command of main command and shadow command. }

else {Steps to deal with the main command list and shadow command list is the same. }

}

5 Simulation and result analysis

In our experiment, we use a simulator is tsim-eval-2.0.7a [12]. Experimental test code using the QuickSort, InsertSort, Fibonacci, MatrixMul these four commonly used test programs, using Gaisler Research compile BBC cross compiler developed by LEON2 and LEON3 to assembly file, and then use software error detection

technology system processing into assembly code with fault-tolerant code, using the BCC [13] compile a connection to generate the ELF file and execute in the TSIM.

The experiment process is to inject a fault into register and memory code runtime, count the experimental data. Limited space, Table 1 and Fig.4 only show the final experimental results.

TABLE 1 Result of fault injection

index	Fibonacci	QuiciSort	InsertSort	MatrixMul	The average detected coverage of SIHFT	Source average accuracy
PC test	88.0%	82.1%	86.7%	92.1%	87.2%	54.5%
NPC test	75.9%	76.3%	88.6%	70.8%	77.9%	45.0%
R test	98.0%	96.1%	90.4%	90.0%	93.6%	75.7%
FP test	76.0%	72.9%	99.0%	92.1%	85.0%	28.1%
MEM test	82.4%	83.2%	75.7%	78.0%	79.8%	37.5%

Table of Real-time data contrast

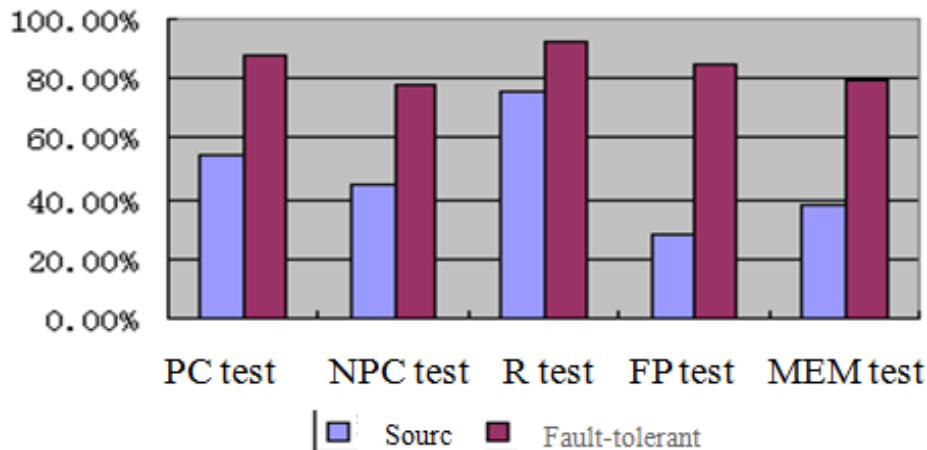


FIGURE 4 Comparison chart of source code and SIHFT error detection rate

According to calculation of 32 general-purpose registers/ PC/ NPC and four status register, statistics of Fibonacci,

etc. four test cases used r register, through calculation, error detection coverage over the last as shown in table 2.

TABLE 2 Result of fault injection

Test procedure	Source Code incorrect results rate	SIHFT undetected rate	SIHFT detected rate
Fibonacci	10.3%	4.8%	95.2%
QuickSort	11.0%	4.0%	96.0%
InsertSort	10.4%	5.3%	94.7%
MatrixMul	11.4%	4.8%	95.2%
average value	10.8%	4.7%	95.3%

It can be seen from table 2 source code register failure not detection rate was 10.8% (system crash + wrong result+ Infinite loop) and after joining fault-tolerant code, register failure not detection rate fell to 4.7%, namely the register error detection coverage rate is 95.3%.

In the results from the experiments on the simulators, TSIM data indicate that assuming the given performance overhead, register to inject faults, error detection coverage rate is 95.3%; Memory Injects faults, error detection coverage of 79.8%, this suggesting that SIHFT technique is effective and feasible.

6 Conclusions

This paper, according to the actual requirements of project, analysed the key problems that are realized on SPARC V8 platform to SIHFT technology, and gave the algorithm to solve the problems and the corresponding technical solution, on this basis, designed a software error detection technology system based on SPARC V8, realized the software signature control flow error detection techniques, tested the error detection coverage that is suitable for the target platform software error detection technology by the simulation test. The results




from the experiments on the simulators TSIM data show that, on the basis of the average performance overhead has been given, the coverage of error detection is higher when register and storage injection failure has been introduced, it shows that SIHFT technique is effective and feasible.

References

- [1] Keyan Pan, Changlong Wang 1998 Radiation-hardened Technology *Onboard Digital Electronic Devices* (3) 67-8
- [2] Qiongying Ren, Jinrong Cai, Guangxuan Luo 2012 *Single Particle Effects of Board Computer and Protection and Reinforcement of it's Software* Guizhou University (Natural Science Edition) 15(3) 178-80
- [3] Oh N, Shirvani P P, McCluskey E J 2002 Control-flow checking by software signatures *IEEE Transactions on Reliability* 51(1) 111-22
- [4] Oh N, Shirvani P P, McCluskey E J 2002 Error detection by duplicated instructions in super-scalar processors *IEEE Transactions on Reliability* 51(1) 63-75
- [5] Shirvani P P, Saxena N, Oh N, Mitra S, Yu Shu-Yi, Huang Wei-Je, Fernandez-Gomez S, Toubia N A, McCluskey E J 2013 *Fault-Tolerance Projects at Stanford CRC* CRC Technical Report
- [6] Zhenyuan Huang 2006 *Research and Implementation of an Onboard Computer Software Error Detection Technology* Harbin Institute of Technology Master Thesis 2006 50-7
- [7] Muchnick S S 2005 *Advanced Compiler Design and Implementation* Mogran Kaufman Kejia Zhao and Zhiyu Shen. Machinery Industry Press 195-9, 381-90
- [8] Shen J P, Lipasti M H 2004 *Modern Processor Design - Basis of Superscalar Processor* Chengyi Zhang, Yu Deng, Lei Wang. Electronic Industry Press 44-7, 8586
- [9] Boyin Lu, Baolin Yin 2001 A Scheduling Optimization Algorithm Based on DAG Graph Instruction *Computer Engineering and Applications* 2001(12) 121-4
- [10] Shuxin Yang, Zhaoqing Zhang 2004 Global Instruction Scheduling Summary *Computer Engineering and Applications* 2004(21) 24
- [11] *The SPARC Architecture Manual Version 8* SPARC International, Inc. Revision SAV080SI9308. 1992 1~57
- [12] *TSIM2 Simulator User's Manual for Version 2.0.7* Gaisler Research AB. January 2007 6-11
- [13] *BCC-Bare-C Cross-Compiler User's Manual. Version 1.0.29* Gaisler Research. February 2007 3-18

Acknowledgment

This work was supported by Harbin reserve Talent project (2014RFQXJ073).

Authors	
	<p>Chunmei Huang</p> <p>Current position, grades: Lecturer of informatics at Computer & Informatics Department, Haerbin Normal University. University studies: M.Sc. in Mathematics (2011) from Harbin Engineer University. Research interests: different aspects of Cloud Computing and Embedded Computing.</p>
	<p>Chunmao Jiang</p> <p>Current position, grades: professor of informatics at Computer & Informatics Department, Haerbin Normal University University studies: he received his M.Sc. in Mathematics (2004) and PhD in Information Sciences (2013) from Harbin industry University. Research interests: different aspects of Cloud Computing and Distributed Systems.</p>
	<p>Mingcheng Qu, born in 1980</p> <p>Current position, grades: Ph.D. in school of computer science and technology of Harbin Institute of Technology. Research interests: embedded computing and P2P etc.</p>