The Journal of Systems and Software 85 (2012) 43–60

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatic testing environment for multi-core embedded software—ATEMES

Chorng-Shiuh Koong[a,*], Chihhsiong Shih[b], Pao-Ann Hsiung[c], Hung-Jui Lai[a], Chih-Hung Chang[d], William C. Chu[b], Nien-Lin Hsueh[e], Chao-Tung Yang[b]

[a] Dept. of Computer and Information Science, National Taichung University, Taichung, Taiwan
[b] Dept. of Computer Science and Information Engineering, Tunghai University, Taichung, Taiwan
[c] Dept. of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan
[d] Dept. of Information Management, Hsiuping Institute of Technology, Taichung, Taiwan
[e] Dept. of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 2 April 2010
Received in revised form 7 July 2011
Accepted 29 August 2011
Available online 10 September 2011

Keywords:
Automatic testing
Embedded software testing
Coverage testing
Unit testing
Cross-testing
Testing tool
Test case generation
Object testing
Multi-core embedded software testing
Parallelism degree testing
TBB testing

ABSTRACT

Software testing during the development process of embedded software is not only complex, but also the heart of quality control. Multi-core embedded software testing faces even more challenges. Major issues include: (1) how demanding efforts and repetitive tedious actions can be reduced; (2) how resource restraints of embedded system platform such as temporal and memory capacity can be tackled; (3) how embedded software parallelism degree can be controlled to empower multi-core CPU computing capacity; (4) how analysis is exercised to ensure sufficient coverage test of embedded software; (5) how to do data synchronization to address issues such as race conditions in the interrupt driven multi-core embedded system; (6) high level reliability testing to ensure customer satisfaction. To address these issues, this study develops an automatic testing environment for multi-core embedded software (ATEMES). Based on the automatic mechanism, the system can parse source code, instrument source code, generate testing programs for test case and test driver, support generating primitive, structure and object types of test input data, multi-round cross-testing, and visualize testing results. To both reduce test engineer's burden and enhance his efficiency when embedded software testing is in process, this system developed automatic testing functions including unit testing, coverage testing, multi-core performance monitoring. Moreover, ATEMES can perform automatic multi-round cross-testing benchmark testing on multi-core embedded platform for parallel programs adopting Intel TBB library to recommend optimized parallel parameters such as pipeline tokens. Using ATEMES on the ARM11 multi-core platform to conduct testing experiments, the results show that our constructed testing environment is effective, and can reduce burdens of test engineer, and can enhance efficiency of testing task.

1. Introduction

As computer software technology evolves to more sophistication, large scale of embedded software application and growing customized demands highlight the importance of embedded software quality control (Yin and Liu, 2009). Among the most discussed issues of software engineering, extensive attention is on improving software quality (Tasse, 2002). Software testing is one of the leading approaches (Hailpern and Santhanam, 2002) to ensure software quality since it enhances software quality and reliability (Kaner et al., 1999; Bertolino, 2007). However, software testing consumes massive manpower and efforts. Automatic or semi-automatic approaches are regarded as useful tools to economize testing time span and efforts.

Traditional unit testing tools mainly focus on workstation platform. Test case and test input data are generated from manual or automatic input. Current method for automatic test case generation still needs to be improved (Michael et al., 2002). Most unit testing tools are only capable of either automatically generating test case program framework, or merely supporting primitive type. Test engineer is obliged to manually write testing program segment and input test data under the generated program framework, or to generate test case manually (Sen et al., 2005). Meanwhile, enormous unwanted workloads are accompanied by repetitive actions during testing. Further, testing conducted by manually input test data is neither efficient nor capable of increasing test coverage (King, 1976).

Generally, embedded system confronted with more hardware and software resource constraints than desktop computer does (Broekman and Notenboom, 2002). Test engineer has to exhaust more efforts to test embedded software before improving quality of embedded software (Myers, 2004). For this reason, embedded

* Corresponding author. Fax: +886 4 22183580.
E-mail addresses: csko@mail.ntcu.edu.tw, shihc@go.thu.edu.tw (C.-S. Koong).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.08.030

software testing differs from conventional testing in that target program can first execute complicated operations on PC side for the required resource necessary to testing, and then execute testing on embedded hardware. This allows software to reduce testing efforts on embedded platform (Delamaro et al., 2006).

The use of multi-core embedded system has been prevalently considered to be better options than other alternatives. This also makes multi-thread program indispensable in improving performance (Hung et al., 2008). However, synchronizing defect such as race condition is an inevitable issue in parallel program. The factor of data synchronization, thus, demands special attention when executing parallel program. This factor coupled with the resource constraints problem faced with testing an embedded system as mentioned before makes the testing of multi-core embedded system especially challenging.

Parallel programs running on multi-core embedded systems usually adopt express coding library such as Intel TBB library (TBB) for scalability and performance. Intel TBB pipeline allows user to decide the extent of parallelism. Parameters such as pipeline token numbers and pipeline stage numbers can be modified as required. Experiences are essential in deciding the better pipeline token numbers and pipeline stage numbers. Parallelism degree can be limited if the token numbers selected are too small. On the contrary, resources can be consumed unnecessarily. For example, more buffer spaces may be needed if token numbers selected are too big. Traditionally, numerous manual modifications are required from programmer before better pipeline token number parameter can be found. To address the issue, this study enhances performance testing by expanding functionalities of automatic multi-round testing. To find better pipeline token number for target program, raw data returned from target-side is calculated automatically and analyzed during runtime. This technology not only frees programmer from consuming extra energy to find parallel pipeline parameter, but also elevates parallel program performance in more precision.

Summing up from the discussion, issues waiting for current multi-core embedded software testing to address include: (1) how demanding efforts and repetitive tedious actions can be reduced; (2) how resource restraints of embedded system platform such as temporal and memory capacity can be tackled; (3) how embedded software parallelism degree can be controlled to empower multi-core CPU computing capacity; (4) how analysis is exercised to ensure sufficient coverage test of embedded software; (5) how to do data synchronization to address issues such as race conditions in the interrupt driven multi-core embedded system; (6) high level reliability testing to ensure customer satisfaction.

This study developed an automatic testing tool to support cross-testing to both reduce target program overhead from performing testing functionality on embedded platform and decrease efforts for test engineer. The ATEMES can not only automatically generate test data with primitive type, structure type, object type and array type but also generate CppUnit-based test case and test driver. This addresses the first issue.

Moreover, ATEMES can execute automatic multi-round performance testing over multi-core embedded software adopting Intel TBB library. The system can support locating recommended value for better parallel parameter token number, which not only allows embedded software parallelism but also facilitates computing capacity of multi-core CPU to operate more efficiently. This feature addresses the issues of (2) and (3).

With the automatic multi-round mechanism, unit testing and coverage testing (Lyu et al., 1994) can be implemented to save test engineer from massive repetitive tasks. With the cross-testing technology between host-side (workstation) and target-side (embedded system), factors resulted from embedded system resource restraints can be reduced for testing. With the cross-testing technology, test case, test driver and target program can be cross-compiled automatically and uploaded to target-side for automatic implementation. Target-side test log data including runtime data, output result, and data of each core utilization during runtime from target-side CPU can be passed to host-side for runtime analysis, results of which can also be visually presented. This helps tackling the issues of (1) and (4).

To address issue (5), we instrument proper lock mechanism, such as mutex of C++, to source code segment of target program pertinent to performance testing. This way the share data can be shielded, and accurate testing results can be ensured. As for multi-core system performance monitor, to analyze overhead of each CPU core in the embedded system, ARM Linux system call is also requisite to effectively monitor CPU core number allocated by the task. Repetitive automatic multi-round testing, similarly, provides a vital scenario to analyze more precisely what performance the task is executing in embedded system, and to locate bottleneck to be tackled. We measure the response time of target program(parallel program) with different interrupt intervals using different core processor numbers on the ARM11 multi-core platform. The test data show that the response time generally decreases as number of cores increases. This provides evidences that our constructed testing environment can not only reduce burdens of test engineer, but also enhance efficiency of multi-core embedded testing task. This helps relieve both issues (3) and (5). Finally, a set of usability testing has been arranged to evaluate testing reliability for issue (6).

2. Related work

Delamaro et al. (2006) developed a coverage testing tool for mobile device software. The tool, named JaBUTi/ME, mainly supports java source code and can solve restrictions of mobile device performance and storage. Testing conducted on mobile device is difficult since issues such as memory limitations, persistent storage, and network connection availability have to be taken into consideration.

The tool not only can be implemented on emulators, but also can help testing on mobile device. With a desktop computer, test engineer can implement testing, instrument class, and generate test session on mobile devices. Communication between desktop computer and mobile device is exchanged through test server. Execution of test case can be monitored through the transfer of trace data when instrumented code is being executed on mobile device. This approach can reduce workloads from testing mobile device. Visualized interface of test result allows test engineer to be informed of what programs are to be executed. However, the tool does not support automatic testing. Failure of automatically generating test input data and test case results in more tasks from test engineer to edit test source code and manually generating test input data.

Ki et al. (2008) proposed an automated scheme of embedded software interface test based on the emulated target board called "Justitia". The setting of breakpoints allows test engineer to debug. However, the tool is efficient only to experienced test engineer who is skilled in the embedded system architecture.

The merit of Justitia is in its automatically detecting errors capability on program interface. Embedded software testing is engaged by combining the existing monitoring and debugging technology of emulator. By defining embedded software interface pattern, an automated scheme is created for locating source code interface. The tool can automatically generate test case, namely, interface test feature, location of interface, symbol to be monitored at the interface, input data, and expected output, and execute test case using emulation testing technology. Further, the system also supports memory test and interrupt test. After testing is finished, result of test coverage and interface error is presented on visualized interface. The system mainly supports single/unit testing rather on