Comparison of two retargetable compilers: GCC and SoarGen

Zheng Zhiwen, Minwook Ahn, Jonghee M. Youn, Yongjoo Kim, Yongin Kwon, Yunheung Paek

Seoul National University

jmjung@optimizer.snu.ac.kr, mwahn@optimizer.snu.ac.kr, jhyoon@optimizer.snu.ac.kr, yjkim@optimizer.snu.ac.kr, yikwon@optimizer.snu.ac.kr, ypaek@snu.ac.kr,

Abstraction

This paper shows our empirical comparison result between two retargetable compilers, GCC and SoarGen. SoarGen is our retargetable compiler. According to our experimental result, using SoarGen for targeting ODALRISC is proved to be easier and faster than using GCC. The average retarget time of the SoarGen is much less than the retarget time of the GCC.

1. Introduction

Porting GCC to new processor is an involved task, and it is difficult to learn for GCC beginner since there is no easy guide for the machine description that can serve porting endeavors. [2] For reducing the time to market, another retargetable compiler called SoarGen [4] had been invented by our research group in order to retarget a new processor faster and easier. For testing whether SoarGen is better than GCC, we did some investigations about porting GCC and SoarGen to ODALRISC [1] respectively.

2. Comparison between GCC and SoarGen

As GCC is free software, it have been extended and developed many times by open communities. GCC is considered as a developer retargetable compiler. [3] Developer retargetable is a way to handle machine specific optimizations that go beyond code generation by permitting the compiler developer to modify the compiler to target the given architecture.

SoarGen uses GCC as its frontend to utilize GCC existing optimization routines, and performs tree/DAG-based code generation. It also offers an architecture description language (ADL) (we call it SoarDL) for target architecture description automaticlly. So, SoarGen is not just a compiler, but a compiler-compiler that automatically generates a compiler from SoarDL. And the most important thing is SoarGen can be learned fast and easily since there is a particular manual with it. However, nothing is perfect, the cycle accurate information cannot be transferred since low level information like pipeline and data path is omitted in SoarGen.

3. Experimental result

For investigating the retargetabilities in the above two compilers, we do some analysis on our porting experiment. First, in order to measure the time required for retargeting GCC and SoarGen by people who know nothing about it, we did a test that porting to ODALRISC processor by GCC and SoarGen respectively. The test was done by five researchers in our research group. The results of the test are displayed in Figure 1.

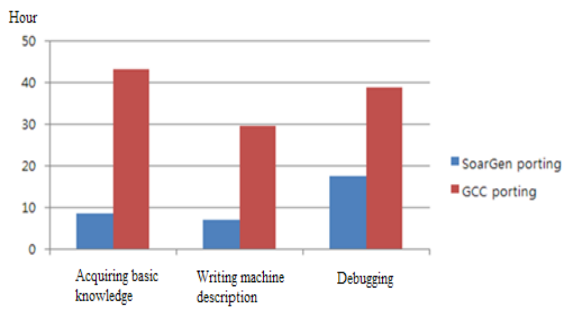

Figure 1. the time consumed for porting using SoarGen and GCC

They all successfully ported SoarGen to ODALRISC, but they didn't complete GCC porting in this test. They almost finished the modification of target description macro file (machine.h) such as implementing the storage layout or basic characteristics of registers and register class information. However, they got into troubles when they were defining machine instructions. In average, they complete about 60% of the whole work of GCC porting since they did not complete the machine instruction definition and the last debugging and verification. Figure 1 shows how much time they spent during each porting.

After we collected the porting result from the researchers, we found the several differences between GCC and SoarGen. In the case of SoarGen, it enables users to describe the target machine at a high level. Machine description file with the top-down approach is intuitive and systematic. Furthermore, it enables us using less grammar to describe the architecture information formally. So it is not much difficult to retarget SoarGen if we got sufficient knowledge about it. Finally, the source code of SoarGen is shorter than GCC's, and it is easier for debugging since the error message is so comprehensive.

However, in the case of GCC, it is difficult to debug poring because of the tremendous changes in the long period of the development time, the huge size of source code and the unkindness error message. Furthermore, as open source software, the document of GCC is not so detailed since so many engineers developed it. Finally, the serious lack of the knowledge about GCC increases the difficulty of porting.

4. Conclusion

From our experiment, we conclude that SoarGen is more intuitive and easier to port a new processor although it does not support the cycle accurate description of the target processor. We will include several new features for representing a target processor in more detailed level including this in SoarGen soon.

5. References

[1]. Imyong Lee, Dongwook Lee and Kiyoung Choi, ODALRISC: A Small, Low Power, and Configurable 32-bit RISC Processor, ISOCC, 2008

[2]. http://gcc.gnu.org/onlinedocs/gccint/, 9.17.2009

[3]. Sejong Oh and Yunheung Paek, A Quantitative Comparison of Two Retargetable Compilation Approaches, ICPP, 2003

[4]. Minwook Ahn, Yunheung Paek and Junghun Cho, Using a H/W ADL-based compiler for fixed point audio codec optimization thru application specific instructions, KIPS Transactions. Part A. vola13, 4, 2006, pp.275-288

6. acknowledge

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST)/ Korea Science and Engineering Foundation(KOSEF). (R11-2008-007-01001-0) , ETRI SoC Industry Promotion Center, Human Resource Development Project for ITSoC Architect, Seoul R&BD Program(10560), Korea Science and Engineering Foundation(KOSEF) NRL Program grant funded by the Korea government(MEST) (No.2009-0083190)