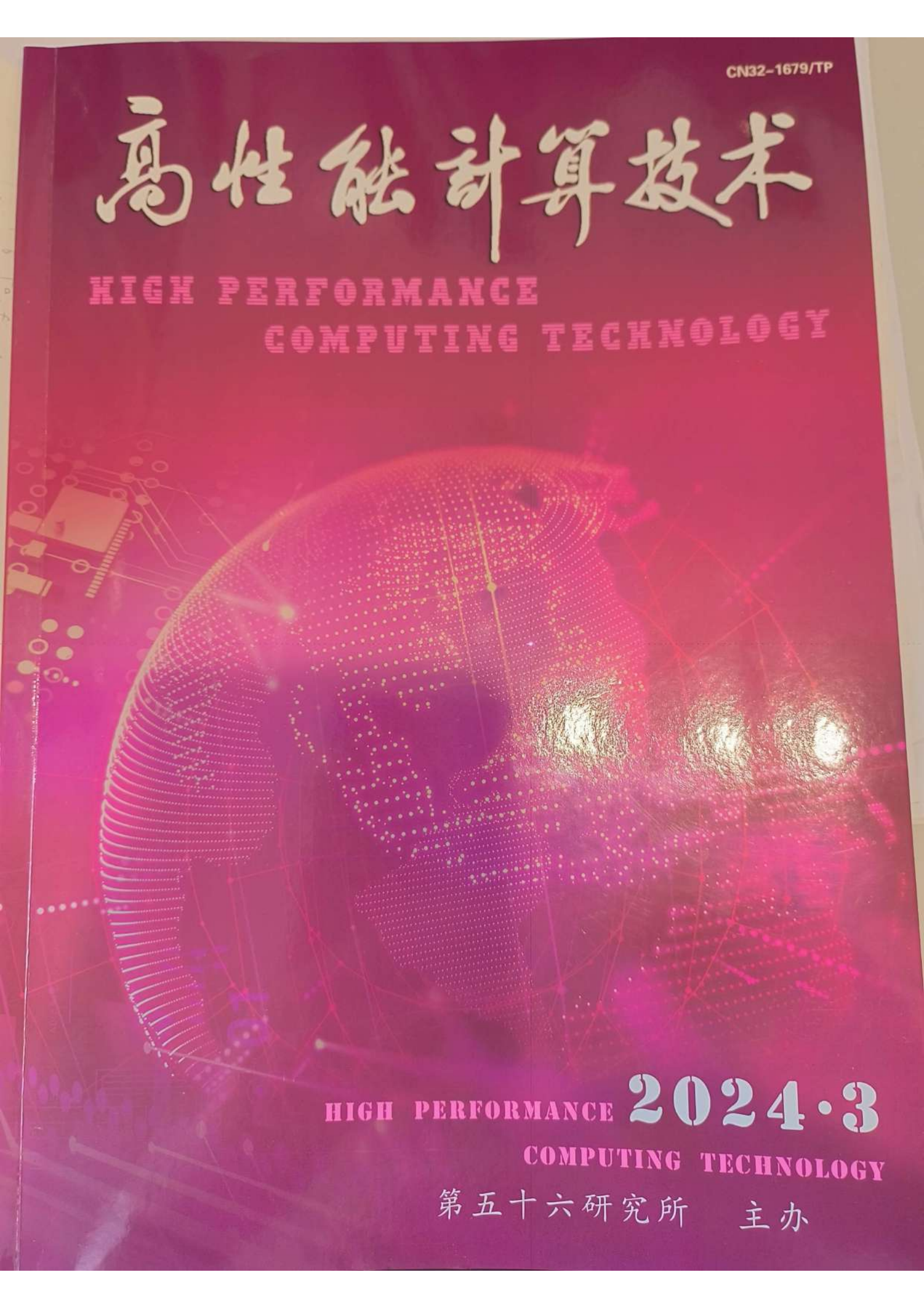
CN32-1679/TP

高性能计算技术

HIGH PERFORMANCE COMPUTING TECHNOLOGY



HIGH PERFORMANCE 2024·3
COMPUTING TECHNOLOGY
第五十六研究所 主办

高性能计算技术

High Performance Computing Technology

1974 年创刊　双月刊
2024 年第 3 期（总第 288 期）

电　话　（0510）85155322
E-mail　hpcbjb@163.com

军内发行　不得外传

目　次

高性能服务器芯片基础生态关键技术

信息动态

面向申威平台的反向调试技术研究

刘汉旭 谢汶兵 尚博文 李佳梅 张杨阳

摘 要 当前国产化适配过程中，随着应用程序的规模和复杂度上升，错误隐匿性增强导致排查难度增大，通常需要反复重启调试来追查原因，极大地消耗了开发人员的精力和时间。反向调试是一种高级调试技术，无需重启即可实现程序指令流的反向执行，从而快速定位到一些复杂性、偶发性程序错误。本文提出一种反向调试技术实现方案，并基于申威国产平台对反向调试功能进行设计和实现。首先根据国产指令集特征，精确识别和提取出寄存器或内存操作；然后记录指令流执行效果和轨迹并保存到日志；最后通过状态重构和状态回放实现对程序历史状态的回溯，并开展 Testsuite 测试验证了反向功能正确性和完整性。测试结果表明，本文所提方案能支持常用反向调试命令，满足反向调试场景需求，从而大幅提升国产平台下软件调试效率。

关键字 申威平台 反向调试 状态记录 历史回放 指令解析
中图法分类号 TP319

1 引言

近年来，国产处理器和国产操作系统发展迅速，国产软硬件平台日益成熟，然而与之配套的调试工具发展相对较慢，严重制约着国产平台下的软件调试效率，影响各类生态软件国产化迁移适配进度[1]。此外，随着应用程序的规模和复杂度上升，程序中错误的隐匿性逐步增强，从而对调试器依赖度越来越高，这就要求开发与处理器相配套的高级调试工具来适应国产化适配进程。

在程序调试的过程中，根据程序的执行方向可以分为正向调试和反向调试。开发人员通常都是采用正向调试的方法跟踪程序执行过程，排查程序中存在的错误。然而，大多数情况下程序中的错误呈现位置和错误根源位置都有一定的距离，开发人员在调试过程中很容易跳过错误根源位置，待察觉后只能重新启动调试，如此往复多次，逐步排查和逼近错误根源。可见该调试过程耗时且低效，尤其针对大型应用程序，重启调试甚至可能需要等待十几分钟。针对该问题，反向调试技术应运而生[2]，开发人员从此可以实现在程序调试过程中“任意穿梭”，回到历史的调试状态。从错误呈现位置开始反向调试，能够很快追溯到错误根源，调试效率得到大幅提升。

反向调试的便捷性和高效性是显而易见的，然而反向调试技术实现却并非易事。反向调试技术的实现与处理器架构紧密相关，目前针对 X86、Arm 平台支持较成熟，对于国产平台支持较弱，尤其针对自主处理器架构，反向调试技术尚未实现。由于该技术的架构强相关性，需根据架构特征进行设计和实现，使得该部分工作量变得庞大且复杂，存在许多技术难点仍待攻克[3]。

本文基于申威国产平台展开反向调试技术研究，根据自主架构特征设计和实现反向调试功能，并通过测试集对该部分进行功

作者简介：刘汉旭（1995-），男，硕士，主要研究方向为国产平台软件调试与优化分析技术。

能正确性和完整性验证。

2 反向调试技术

2.1 概念介绍

通常情况下程序指令流是正向执行的，即从程序第一条指令开始执行，直到最后一条。反向调试则是打破这种执行规则，任意改变程序控制流，既能够使得程序指令流往相反的方向执行，达到一种程序指令流倒退执行的效果；也能使程序任意倒退或前进到特定指令位置，实现在程序指令流中“任意穿梭”的效果。

2.2 实现方式

目前针对反向调试技术的实现方式主要分为三种[4][5]：

（1）基于事件的反向调试

在调试过程中，将程序中发生的外部事件进行分类和记录，对程序运行状态的关键点进行快照，比如发生系统调用时、进行信号交互时[6]。在反向调试时，根据需要在快照中前进与后退，并读取快照存储下来的各类事件，从而恢复出程序运行状态。像 Visual Studio 企业版中的 IntelliTrace 功能便是基于事件的反向调试功能实现[7]。然而，该实现方式导致调试粒度较大且不够灵活，无法进行单步反向以查看单条指令或单行源码执行前后的程序状态，只能跳跃式地在离散的快照之间穿梭。

（2）基于时间的反向调试

在程序被调试过程中，根据时间间隔对程序执行过程中的状态进行保存，通过缩小时间间隔以“模拟纳秒”的模式设置检查点，反向时可以根据时间来恢复到先前某一时刻的运行状态。通过该方式实现反向功能的调试器有 UndoDB[8]。该实现方式虽然在一定程度上保障调试灵活度，却增加了性能开销，此外还不能处理程序中的系统调用[4]。

（3）基于指令的反向调试

记录程序执行过程中每一条指令的状态修改，并保存到日志中。反向调试时先读取日志文件，再根据日志文件的描述复现程序的执行状态，即对程序的历史状态进行回放，从而恢复程序在任意位置的运行状态，保证了最高的历史调试精准度，支持更丰富的反向调试命令，还可以修改程序执行流程，代价便是带来一定的程序性能开销。GDB 中的反向调试功能便是基于该方式实现[9]。

2.3 难点和挑战

通过对比上述三种实现方式，发现基于指令的反向调试提供了最高的调试精准度，同时其方式实现难度也更大，存在诸多挑战性问题需要攻克：

（1）指令集架构差异大。不同架构指令集种类、数量和功能也各有不同，需要根据特定架构特征实现对程序指令的精准解析，比如某条指令将会修改哪些寄存器或读写哪块内存。此外，不同架构系统调用和信号处理方式也有不同，也需要额外处理。

（2）指令状态保存开销大。程序执行期间需要额外保存每条指令的执行状态，软件里包含指令流数量庞大，导致大量的内存空间消耗，需要设计高效的指令状态存储方式降低指令状态的保存开销。

（3）历史状态重构难度大。读取到保存的指令状态后，如何重构数据以完整复现出指定位置的程序状态，以及如何快速处理历史状态与未来状态的切换，都是一些挑战性问题，该过程复杂且实现难度大。

鉴于程序调试场景对性能要求并非关键，反而更加聚焦调试精准度，本文选取基于指令的方式在国产申威平台进行反向功

能设计实现，并针对上述三个方面的挑战进行探索研究。

3 国产平台下反向调试方案设计

下面将基于指令的方式进行国产平台下反向调试方案设计，针对指令集架构差异、状态保存开销、历史状态重构三个方面存在的难点进行模块设计，解决其问题。

3.1 指令解析模块设计

若想记录指令的执行状态，首先需要对指令进行精确解析。不同国产处理器的指令编码方式也不同，尤其针对自主处理器架构，需要设计一套专用、高效、精确的指令解析模块。

根据国产平台指令编码格式，解析指令里包含的操作码和操作数，得到程序中每条指令的执行状态，即指令所修改的寄存器和内存。例如，图 1 所示为某条指令编码格式，其中 bit[0, x]用来编码操作码，bit[x+1, y]用来编码第 1 个寄存器，bit[y+1, z]用来编码第 2 个寄存器，bit[z+1, 31]用来编码偏移量。基于该指令的编码规则，通过提取操作，可精准解析出寄存器和偏移量，并进一步计算出该指令所修改的寄存器或内存。

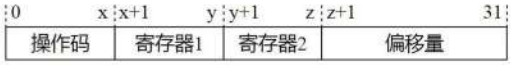


图 1 指令编码示意图

3.2 指令状态记录模块设计

通过指令解析模块获取到每条指令的执行效果，进行记录便可以得到程序的执行轨迹。由于每条指令目的操作数可能会修改寄存器或内存，也有可能不做任何修改，因此使用固定数组来存储历史状态势必会造成内存浪费，故选用循环分配双向链表节点，只对执行过程中被修改的寄存器或内存记录信息并分配双向节点，极大地节省了内存，也保证了反向调试的效率。指令状态记录链表结构如图 2 所示，每个链表节点对应一条指令，每个节点保存了修改的寄存器和内存内容和结束信号。其中 record(n)表示已记录的第 n 条指令状态，reg 表示该指令修改的寄存器，mem 表示该指令修改的内存，end 表示当前指令状态的结束标志。

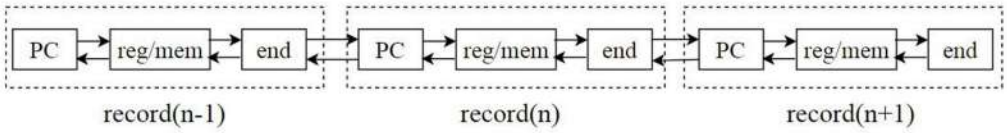


图 2 指令状态存储链表

3.3 历史状态回放模块设计

指令解析并记录完成后，会形成一个记录着寄存器和内存变化情况的 record 链表，即保存程序指令流执行轨迹的日志。当进行反向调试时，需将 record 链表中的对应历史状态进行有序回放。通过把第 n-1 条指令历史状态 record(n-1)替换成当前程序状态，覆写寄存器和内存的值，实现历史状态回放。若想在反向调试过程中修改程序执行流程，需要判断目的指令状态是否被记录，并根据判断结果进行指令状态交换或状态执行。

4 申威平台反向调试功能实现与验证

4.1 实现过程

文中申威平台为64位字长的RISC架构，采用 32 位定长格式自主指令集。反向调试

功能基于开源调试器 GDB 进行实现，根据本文所述方案，反向调试实现流程如图 3 所示，其具体实现过程描述如下：

（1）指令解析

针对申威平台指令格式对第 n 条指令进行译码，对操作码和操作数进行正确提取，精确计算出每条指令所修改的寄存器或内存，并记录到双向链表。

（2）调试方向判断

首先标记当前程序执行状态为 PC_state，然后通过 PC_state 中 flag 变量取值来判断当前程序调试方向。

（3）反向调试时进行历史状态回放

若程序调试方向为反向，则先将所要跳转的第 n-1 条目标指令的历史状态 record(n-1)与当前程序执行状态 PC_state 进行交换，并将程序指针 PC 指向所要跳转的第 n-1 条指令位置，最后将 record(n-1)中包含的历史状态进行回放，即将寄存器或内存值进行覆盖，从而实现“程序倒着执行”的效果。

（4）正向调试时进行状态记录或回放

若程序调试方向为正向，则先将 PC 指向第 n+1 条指令的位置，并通过查找双向链表判断第 n+1 条指令的历史状态 record(n+1)是否已被记录。若 record(n+1)已被记录，则将当前程序执行状态 PC_state 与 record(n+1)进行交换，实现状态回放，无需指令执行开销；若 record(n+1)未被记录，则需要解析第 n+1 条指令并交予 CPU 执行，再将指令状态 record(n+1)添加至双向链表。这样便可实现“程序来回穿梭着执行”的效果。

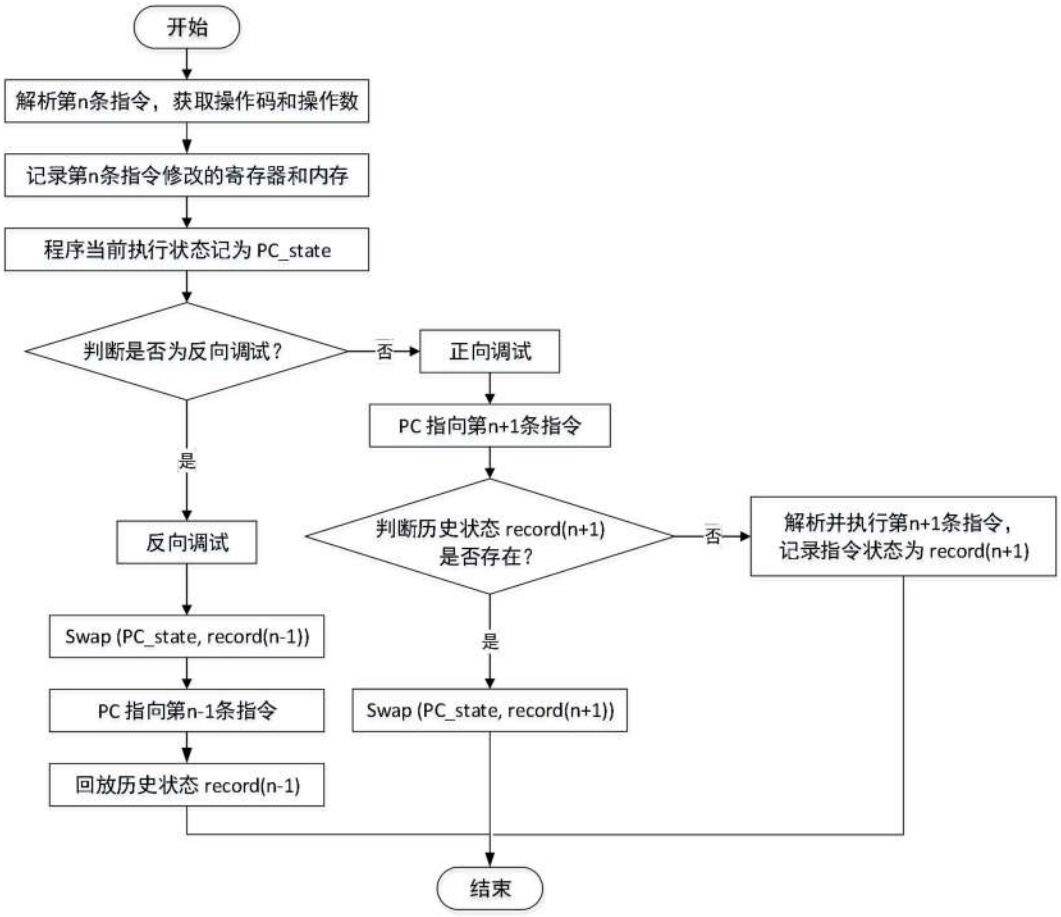


图 3 基于申威平台反向调试实现流程图

基于申威平台实现的反向调试命令种类见表 1，能够满足开发人员日常的反向调试需求[10]。

表 1 反向调试命令实现列表

反向调试命令	功能介绍
record [stop]	状态记录启动[停止]
reverse-step	源码级单步反向(步入)
reverse-stepi	指令级单步反向(步入)
reverse-next	源码级单步反向(步出)
reverse-nexti	指令级单步反向(步出)
reverse-continue	反向到上个断点
reverse-finish	反向到函数调用位置

4.2 测试集验证结果

为了验证本文所实现反向调试功能的正确性和完整性，接下来基于申威 6432 平台，选取 GDB9.1 官方测试集 Testsuite 开展实验验证。Testsuite 里包含 gdb 各功能模块的测试用例，这里选取了针对反向调试的功能测试用例[11]。

实验中选取飞腾 S2500 进行对比测试，飞腾平台反向调试功能在 GDB 里具有官方原生支持[12]，因此选取飞腾平台测试结果作为参照，能充分反映新架构中对反向调试功能支持情况。

验证结果如表 2 所示，可以看出本文基于申威 6432 平台提出的反向调试方案测试结果与飞腾平台一致，即针对反向调试每个测试用例里通过的测试项（expected passes）数目一致。该结果表明本文基于申威 6432 平台设计和实现的反向调试功能，具备与主流架构同等的正确性和完整性，能够满足软件调试场景中的反向调试需求。

表 2 各测试用例通过的测试项（expected passes）数量

测试用例	飞腾平台	申威平台
insn-reverse.exp	1	1
finish-reverse.exp	32	32
break-reverse.exp	15	15
until-reverse.exp	12	12
pipe-reverse.exp	6	6
watch-reverse.exp	32	32
fstatat-reverse.exp	6	6

5 结束语

国产软硬件平台发展日益成熟，而与之配套的调试工具发展缓慢，严重制约着国产平台下的软件调试效率。对此，本文面向申威国产平台提出一种反向调试技术方案，并基于 GDB 调试器进行设计和实现，最后通过 Testsuite 测试集验证反向调试功能的正确性和完整性。测试结果和飞腾保持一致，表明该方案能够满足反向调试场景需求，有助于提升国产平台软件调试效率。

参考文献

[1]丁吕繁,陈玮彤,王俊翔,等.信息创新环境下的嵌入式软件交叉调试技术[J].电子元器件与信息技术,2022,6(02):165-167.DOI:10.19772/j.cnki.2096-4455.2022.2.064.

[2]Engblom J. A review of reverse debugging[C]//Proceedings of the 2012 System, Software, SoC and Silicon Debug Conference. IEEE, 2012: 1-6.

[3]张大方,胡先浪,王立杰.基于国产平台的智能跟踪调试技术[J].计算机系统应用,2019,28(08):101-108.DOI:10.15888/j.cnki.csa.007032.

[4]徐建波,舒辉,康绯.反向调试技术研究综述[J].计算机科学,2021,48(05):9-15.

[5]Dmitriev D S, Dovgalyuk P M, Makarov V A. What is reverse debugging? Classification of reverse debugging methods[C]//Journal of Physics: Conference Series. IOP Publishing, 2019, 1352(1): 012010.

[6]乔文利,孔祥营.面向国产平台的反向调试技术[J].传感器世界,2019,25(11):23-27.DOI:10.16204/j.cnki.sw.2019.11.004.

[7]IntelliTrace for Visual Studio Enterprise [EB/OL].https://learn.microsoft.com/en-us/visualstudio/debugger/intellitrace?view=vs-2022

[8]UDB Time Travel Debugging for C/C++

[EB/OL].https://undo.io/products/udb
[9]Process Record and Reply[EB/OL].https://sourceware.org/gdb/wiki/ProcessRecord
[10]Running programs backward[EB/OL].https://sourceware.org/gdb/current/onlinedocs/gdb.html/Reverse-Execution.html#Reverse-Execution
[11]Testing[EB/OL].https://sourceware.org/gdb/wiki/ProcessRecord#Testing
[12]Supported Targets[EB/OL].https://sourceware.org/gdb/wiki/ProcessRecord#Supported_Targets

Research on Reverse Debugging Technology base on SW64 Platform

LIU Han-xu， XIE Wen-bing， SHANG Bo-wen， LI Jia-mei， ZHANG Yang-yang

Abstract: In the current domestic software adaptation, as the scale and complexity of applications increases, the hiddenness of errors increases, making troubleshooting more difficult. It is usually to restart the debugging repeatedly to trace errors, which greatly employs the energy and time cost overheads. Reverse debugging is an advanced debugging technology that can realize the reverse execution of program instruction steams without restarting, so as to quickly locate some complex and occasional program errors. In this paper, we proposed a reverse debugging technology implementation scheme, designed and implemented the reverse debugging function based on SW64 platform. Firstly, according to the characteristics of the domestic architecture instruction set, the register or memory operations is accurately identified and extracted. Then the execution effect and trajectory of the instruction stream are recorded and saved to the log. Finally, the program historical state is traced back through state reconstruction and state replay, and the Testsuite is used to verify the correctness and completeness of the reverse function. The results show that the proposed approach can support commonly reverse commands and meet the requirements of reverse debugging scenarios, and greatly improve the software debugging efficiency under the domestic platform.

Keys: SW64 platform， reverse debugging， state record， state replay， instruction parse