第 1 章 引言

1.1 研究背景与意义

随着登纳德缩放定律和摩尔定律的终结，标准处理器性能提升的减速已成为了既定事实，新的体系结构黄金时代需要寻求领域独特的设计语言和指令架构[1]。在开源软件生态的驱动下，加州大学伯克利分校提出了一个免费、开源的指令集 RISC-V（RISC Five），并凭借其灵活的模块化设计广泛应用于各种领域独特的高性能微处理器设计中[2][3]。

随着 RISC-V 软硬件生态的日益完善，逐渐对 RISC-V 架构下软件的可重现性提出了要求。一方面，RISC-V 并行程序调试过程中需要循环执行程序并重现错误，以此提高程序的可靠性[4]；另一方面，RISC-V 平台上训练和推理机器学习模型的需求日益增加，结果的可重现性能够追踪模型性能变化趋势，帮助开发人员寻找性能变化的原因[5]。因此，以较低的额外开销实现程序的可重现性在 RISC-V 软件生态的发展中有着重要意义。

可重现性可以进一步分解为两个子属性，即确定性和可移植性[6]：确定性保证相同输入情况下，反复执行程序始终得到相同的结果；可移植性则保证程序无需过多修改即可在不同机器上部署、执行。在软件工程中，通常使用确定性重放（Deterministic Replay，或记录重放，Record and Replay, R&R）技术[7][8][9]实现程序的重现性。确定性重放技术分为记录和重放两个阶段，通过记录并重放程序执行过程，尽可能保证程序确定性的输出。确定性重放工具通过追踪、记录程序的执行踪迹（Execution Trace），在下一次运行该程序时按照记录的踪迹信息重现执行结果。按照重放系统范围，可将现有确定性重放工具分为程序重放和全系统重放。程序确定性重放技术的核心问题是查找并拦截程序执行过程中不确定性（Non- deterministic）的来源，如查找、拦截具有不确定性的系统调用和 CPU 指令，记录并重现它们返回的数据，设置周期性的进程检查点以实现在程序执行的任意时刻进行跳转[10]。除此之外，还存在确定性操作系统的解决方案，通过记录整个虚拟机[11][12]，或者修改系统内核[13][14]等方法来保证可重现性。

然而，目前的确定性重放工具并不能完全建立可重现抽象。程序确定性重放

工具需要在源程序基础上进行修改，增加部署和维护成本，产生额外的性能和复杂性开销，并且通用性差，无法记录所有程序[7]；全系统重放记录整个虚拟机的方式更加复杂，修改内核同样增加部署和维护成本，且要求在特定的指令集架构和操作系统环境下实现[11][12]。因此，现有的确定性重放工具并不能完全满足可重现性的要求。

基于上述背景，本文针对 RISC-V 上的程序执行的可重现性问题，设计实现一种容器化可重现方法：在程序执行期间，通过命名空间隔离用户进程和主机系统，通过进程追踪拦截并修改不确定性的系统调用，基于 QEMU 的二进制翻译工具修改不确定性的 CPU 指令，并在 RISC-V 平台上完成实验验证与分析。