分类号： 密级：

UDC： 学号：

南 昌 大 学 硕 士 研 究 生

学 位 论 文

**基于RISC-V架构的容器化可重现方法研究**

**Research of Reproducible Container Based on**

**RISC-V Architecture**

崔傲

培养单位（院、系）： 数学与计算机学院

指导教师姓名、职称： 徐子晨 教授

申请学位的学科门类： 工学

学科专业名称： 计算机科学与技术

论文答辩日期：

答辩委员会主席：

评阅人：

年 月 日

书脊(提交论文电子版时，此页请删除)

|  |
| --- |
| 5cm左右  **中部崛起视野中的中东部互动发展与经济合作研究**  **姓名**  **南昌大学**  5cm左右 |

**一、学位论文独创性声明**

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南昌大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名（手写）： 签字日期： 年 月 日

**二、学位论文版权使用授权书**

本学位论文作者完全了解南昌大学有关保留、使用学位论文的规定，同意学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权南昌大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。同时授权北京万方数据股份有限公司和中国学术期刊（光盘版）电子杂志社将本学位论文收录到《中国学位论文全文数据库》和《中国优秀博硕士学位论文全文数据库》中全文发表，并通过网络向社会公众提供信息服务，同意按“章程”规定享受相关权益。

学位论文作者签名（手写）： 导师签名（手写）：

签字日期： 年 月 日 签字日期： 年 月 日

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 论文题目 |  | | | | |
| 姓 名 |  | 学号 |  | 论文级别 | 博士□ 硕士□ |
| 院/系/所 |  | | 专业 |  | |
| E\_mail |  | | | | |
| 备注： | | | | | |

□公开 □保密（向校学位办申请获批准为“保密”， 年 月后公开）

摘 要

在进行程序调试、分布式系统构建及机器学习模型训练等应用场景中，往往需要确保程序在不同机器间反复执行能够得到可重现的结果。现有的确定性重放工作难以同时满足确定性和可移植性的要求：一是保证输出的确定性需要对程序进行的额外修改来维护系统时间或标识符的唯一性；二是可移植性差，难以跨机器执行。针对上述问题，本文对基于RISC-V架构的容器化可重现方法展开研究，主要工作如下：

第一，本文对Linux系统下的不确定性进行研究，确定不确定性来源，并就确定性重放问题的国内外研究现状展开调查与分析。

第二，基于上述问题与内容，本文在x86架构下实现一种容器化可重现方法，建立可重现容器抽象，保证容器内程序强制以可重现的方式运行，且无需对源程序进行修改，并向RISC-V平台进行移植优化。该方法通过命名空间等隔离用户进程，利用ptrace对标识符、系统调用、信号等中的非确定因素进行追踪、拦截，实现可重现的软件输出。本文还通过模拟器将可重现容器抽象移植到RISC-V架构下运行。

第三，本文使用Gem5对RISC-V硬件平台进行全系统仿真，并进行容器化可重现方法的实验验证和性能评估。评估结果表明，与xxxx相比，…………。

**关键词：**确定性；RISC-V；记录重放；容器化

ABSTRACT

……

**Key Words:**

目 录

[第1章 引言 1](#_Toc97973316)

[1.1 研究背景与意义 1](#_Toc97973317)

[1.2 国内外研究现状 2](#_Toc97973318)

[1.2.1 确定性重放技术研究现状 2](#_Toc97973319)

[1.2.2 RISC-V研究现状 2](#_Toc97973320)

[1.3 主要研究内容 3](#_Toc97973321)

[1.4 组织结构 3](#_Toc97973322)

[第2章 RISC-V指令集与可重现性研究 1](#_Toc97973323)

[2.1 RISC-V指令集 1](#_Toc97973324)

[2.1.1 模块化设计 1](#_Toc97973325)

[2.1.2 函数调用规范 1](#_Toc97973326)

[2.1.3 RV32/64特权架构 1](#_Toc97973327)

[2.2 可重现性研究 1](#_Toc97973328)

[2.2.1 确定性重放 1](#_Toc97973329)

[2.2.2 确定性模型 2](#_Toc97973330)

[2.2.3 确定性重放实现方法 2](#_Toc97973331)

[2.2.4 虚拟化与容器 2](#_Toc97973332)

[第3章 容器化可重现方法 3](#_Toc97973333)

[3.1 基于ptrace的系统调用拦截 3](#_Toc97973334)

[3.2 可重现方法工作流程 3](#_Toc97973335)

[3.3 不确定性来源 3](#_Toc97973336)

[3.3.1 用户进程编号 3](#_Toc97973337)

[3.3.2 随机函数 4](#_Toc97973338)

[3.3.3 时间 4](#_Toc97973339)

[3.3.4 信号 4](#_Toc97973340)

[3.3.5 文件和目录 4](#_Toc97973341)

[3.4 容器中的不确定性来源 4](#_Toc97973342)

[第4章 基于RISC-V架构的容器化方法 5](#_Toc97973343)

[4.1 基于Gem5的RISC-V全系统仿真 5](#_Toc97973344)

[4.1.1 Gem5全系统模拟器 5](#_Toc97973345)

[4.2 RISC-V容器化方法 5](#_Toc97973346)

[4.2.1 基于命名空间的容器架构 5](#_Toc97973347)

[4.2.2 基于QEMU的模拟器 5](#_Toc97973348)

[4.2.3 动态二进制指令翻译 5](#_Toc97973349)

[4.3 RISC-V容器化方法中的不确定性来源 5](#_Toc97973350)

[4.4 RISC-V特权级切换 5](#_Toc97973351)

[第5章 实验设计与分析 6](#_Toc97973352)

[5.1 软硬件平台 6](#_Toc97973353)

[5.2实验环境配置 6](#_Toc97973354)

[5.3 可重现性方法功能验证 6](#_Toc97973355)

[5.4 可重现方法性能损耗分析 6](#_Toc97973356)

[第6章 结论与展望 7](#_Toc97973357)

[致 谢 8](#_Toc97973358)

[参考文献 9](#_Toc97973359)

[附录A ××××× 10](#_Toc97973360)

[攻读学位期间的研究成果 11](#_Toc97973361)

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，标准微处理器性能提升的减速已成为了既定事实，体系结构在新的黄金时代需要寻求新的前进方向[1]。加州大学伯克利分校提出了RISC-V（RISC Five），即第五代RISC架构。RISC-V并非是精简指令集简单的版本迭代，和前代相比它最大的优势在于开源和模块化，允许用户基于特定需求添加定制化拓展指令集。RISC-V由于其高度的灵活性在工业界和学术界均受到广泛关注，推出了一系列支持乱序执行的微处理器，如BROOM等[2]，将会应用在可穿戴设备、智能家居、机器人、自动驾驶及工业装置等领域的计算设备中，在边缘微设备的应用中具有广阔的前景。

随着RISC-V软件生态的日益完善，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；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，可重现性能够确保模型结果正确，帮助开发人员寻找模型性能变化的原因。因此，以较低的额外开销确保程序的可重现性在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[10]：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）技术实现程序的重现性。确定性重放技术通过记录并重现程序错误，尽可能避免不确定性因素对程序的影响。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是查找并拦截不确定性（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，记录并重现它们的调用，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[4][5][6]。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[7][8]，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[3]；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。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，设计实现一种轻量级的容器化可重现方法，在程序执行期间通过容器隔离不确定性的CPU指令和系统调用，同时满足对确定性和可移植性的要求，并在RISC-V架构下完成该方法的实验与分析。

1.2 国内外研究现状

国内外研究人员对可重现性进行了深入的研究，根据研究方向、内容和方法大致可以划分为三个方向，分别是基于程序的可重现性研究、基于执行环境的可重现性研究以及基于虚拟化的可重现性研究，本节从这三个方向对可重现性研究的国内外现状进行综述。

1.2.1 基于程序的可重现性研究——确定性执行

可重现性的需求在19世纪80年代后期就引起了研究人员的注意，并基于确定性重放技术设计并实现了多种先进的记录和重放工具，最早应用于并行程序调试过程中

。

在软件调试过程中，顺序程序可以通过循环执行的方式，发现和纠正程序执行中的错误。但是对并行程序的2次执行可能会产生不同的结果。1987年，LeBlanc等人提出了Instant Replay[1]，作为一种重现并行程序执行行为的一般解决方案。Instant Replay通过记录重点事件发生的时序，保存程序重放所需的信息。Instant Replay后来成为了多项确定性重放技术的前身，乃至发展出了分布式内存上的可重现模型。

1988年，Pan等人提出了Recap，结合检查点和数据重放记录方法，在程序执行期间记录系统调用、共享内存读取结果及异步事件（信号）发生的时间。相较于前者，Recap实现了从某一个检查点重放的功能，不必从程序头重新执行。但是Recap使用日志机制保存事件信息，增加了巨大的额外存储和性能开销。此外，Bacon等人通过记录指令计数器保存共享内存访问的总顺序，实现了基于硬件辅助机制的确定性重放技术，

确定性重放工具Bugnet[3]能够记录外部I/O事件并重放并行程序，但Bugnet只支持特定API；

Flashback[4]能够记录和重放更多的系统调用和CPU指令，但Flashback必须作为操作系统拓展使用，修改操作系统内核，部署和维护困难，且只支持单线程程序的确定性重放；

Jockey[5]可以实现与Flashback相似的功能，可作为程序运行中的动态链接库拦截系统调用和CPU指令。但是Jockey需要作为目标进程的一部分运行，增加了部署和维护成本，并且不支持很多程序。

1.2.2 基于环境的可重现性研究——确定性操作系统

Rosse等人

此外，Aviram等人[7]、Hunt等人[8]、Bergan等人[9]均提出了操作系统级别的可重现抽象。

Aviram 等人2010年提出了Determinator，这是一个围绕确定性设计的操作系统抽象。Determinator实现了操作系统级别的确定性，强制单个进程，甚至交互进程组确定性执行。Determinator通过拒绝用户代码直接访问硬件资源来强制执行确定性，包括实时时钟、周期计数器和可写共享内存等。

DDOS专注于

1.2.3 基于虚拟化的可重现性研究

容器技术起源于虚拟化。虚拟化技术是在一台主机上运行多个进程，将硬件资源抽象为虚拟逻辑对象的技术，包括计算机的硬件资源、存储设备和网络资源的虚拟等。虚拟化技术包括平台虚拟化、硬件虚拟化、应用程序虚拟化等，平台虚拟化技术允许在宿主机设备中运行多个异构的体系结构应用，通过虚拟机监视器（Virtual Machine Monitor，VMM，或称为Hypervisor）为用户提供抽象、虚拟的硬件环境。Popek和Goldberg等人1974年的论文[8] 为将系统软件视为VMM确立了三个基本特征：（1）保真。 VMM上的软件的执行与硬件上的执行相同，除非定时影响；（2）性能。 绝大多数来宾指令由硬件执行，而无需VMM的干预；（3）安全。 VMM管理所有硬件资源。VMM通过内核代码的二进制翻译实现虚拟化，在宿主机和虚拟机之间添加一层中间层，将宿主机处理器的指令代码转换、翻译成目标处理器的指令集，捕获文件执行时所需的系统调用。VMware ® Workstation、Virtual PC、 QEMU等均是采用的这种方法实现硬件的虚拟化。Adams等人[9]对基于x86架构下的软硬件虚拟化技术进行了比较，得出结论，硬件VMM的性能通常比纯软件VMM低。硬件虚拟化技术不具备性能优势的原因主要有2个：（1）它不支持MMU虚拟化。 （2）它无法与用于MMU虚拟化的现有软件技术共存。Shuja等人[10]根据针对ARM架构下移动虚拟化的硬件支持的最新进展，调查了基于软件和硬件的移动虚拟化技术，并介绍了CPU，内存，I / O，中断和网络接口的在移动设备中虚拟化面临的挑战和问题。他们的研究最后提出，在资源受限的移动设备上实施基于CPU的虚拟化解决会消耗CPU周期和内存空间，实现该方案的成本总是很高，而使用静态二进制转换实现虚拟化的解决方案开销更低。针对资源有限的边缘设备必须使用资源有效的技术来解决上述问题。Bernstein等人[11] 介绍了Docker和Kubernetes，前者是一个开源项目，可以自动化Linux应用程序的快速部署，后者是一个用于Docker容器的开源集群管理器。

1.3 主要研究内容

1.4 组织结构

第2章 RISC-V指令集与可重现性研究

本章介绍RISC-V指令集与可重现性方法的基础理论与相关知识背景。

2.1 RISC-V指令集

RISC-V作为近年来最火热的开源指令集架构，被广泛应用于各个特定领域的微处理器，特别是机器学习领域的模块化定制。

2.1.1 模块化设计

计算机体系结构的传统方法是增量ISA，新处理器不仅必须实现新的ISA扩展，还必须实现过去的所有扩展。目的是为了保持向后的二进制兼容性，这样几十年前程序的二进制版本仍然可以在最新的处理器上正确运行。这一要求与来自于同时发布新指令和新处理器的营销上的诱惑共同导致了ISA的体量随时间大幅增长。例如，图1.2显示了当今主导ISA80x86的指令数量增长过程。这个指令集架构的历史可以追溯到1978年，在它的漫长生涯中，它平均每个月增加了大约三条指令。

这个传统意味着x86-32（我们用它表示32位地址版本的x86）的每个实现必须实现过去的扩展中的错误设计，即便它们不再有意义。例如，图1.3描述了x86的ASCII Adjust afterAddition（aaa）指令，该指令早已失效。

作为一个类比，假设一家餐馆只提供固定价格的餐点，最初只是一顿包含汉堡和奶昔的小餐。随着时间的推移，它会加入薯条，然后是冰淇淋圣代，然后是沙拉，馅饼，葡萄酒，素食意大利面，牛排，啤酒，无穷无尽，直到它成为一顿大餐。食客可以在那家餐厅找到他们过去吃过的东西，尽管总的来说这样做可能没什么意义。这样做的坏处是，用餐者为每次晚餐支付的宴会费用不断增加。

RISC-V的不同寻常之处，除了在于它是最近诞生的和开源的以外，还在于：和几乎所有以往的ISA不同，它是模块化的。它的核心是一个名为RV32I的基础ISA，运行一个完整的软件栈。RV32I是固定的，永远不会改变。这为编译器编写者，操作系统开发人员和汇编语言程序员提供了稳定的目标。模块化来源于可选的标准扩展，根据应用程序的需要，硬件可以包含或不包含这些扩展。这种模块化特性使得RISC-V具有了袖珍化、低能耗的特点，而这对于嵌入式应用可能至关重要。RISC-V编译器得知当前硬件包含哪些扩展后，便可以生成当前硬件条件下的最佳代码。惯例是把代表扩展的字母附加到指令集名称之后作为指示。例如，RV32IMFD将乘法（RV32M），单精度浮点（RV32F）和双精度浮点（RV32D）的扩展添加到了基础指令集（RV32I）中。

RISC-V 指令集包括几个可以互相替换的基本指令集，以及额外可以选择的扩充指令集。基本指令集包括 RV32I，RV64I 等，实现不同位数的，最基础的整数加减的运算；而扩充指令集包括了

M（整数乘除法扩充）

A（不可中断指令扩充）

F（单精度浮点运算扩充）

D（双精度浮点运算扩充）

等等，基本可以实现我们熟悉的指令集的所有功能。

2.1.2 函数调用规范

2.1.3 RV32/64特权架构

除用户模式（U-mode）以外，RISC-V设计了两种具有更高权限的模式：机器模式（Machine Mode，缩写为M-mode）以及监管者模式（Supervisor Mode，缩写为S-mode）。M-mode被设计用来拦截和处理异常。中断和异常是不确定性的主要来源。S-mode的核心是使用基于页面的虚拟内存实现内存保护。这是一种用于更复杂RISC-V处理器上的可选模式。S-mode的权限基于U-mode和M-mode之间，不能使用M-mode下的CSR和指令。在类Unix系统，如

机器模式（缩写为 M 模式，M-mode）是 RISC-V 中 hart（hardware thread，硬件线程）可以执行的最高权限模式。在 M 模式下运行的 hart 对内存，I/O 和一些对于启动和配置系统来说必要的底层功能有着完全的使用权。因此它是唯一所有标准 RISC-V 处理器都必须实现的权限模式。实际上简单的 RISC-V 微控制器仅支持 M 模式。这类系统是本节的重点。

机器模式最重要的特性是拦截和处理异常（不寻常的运行时事件）的能力。RISC-V 将异常分为两类。一类是同步异常，这类异常在指令执行期间产生，如访问了无效的存储器地址或执行了具有无效操作码的指令时。另一类是中断，它是与指令流异步的外部事件，比如鼠标的单击。RISC-V 中实现精确例外：保证异常之前的所有指令都完整地执行了，而后续的指令都没有开始执行（或等同于没有执行）。图 10.3 列出了触发标准例外的原因。

在 M 模式运行期间可能发生的同步例外有五种：

⚫ 访问错误异常 当物理内存的地址不支持访问类型时发生（例如尝试写入 ROM）。

⚫ 断点异常 在执行 ebreak 指令，或者地址或数据与调试触发器匹配时发生。

⚫ 环境调用异常 在执行 ecall 指令时发生。

⚫ 非法指令异常 在译码阶段发现无效操作码时发生。

⚫ 非对齐地址异常 在有效地址不能被访问大小整除时发生，例如地址为 0x12 的amoadd.w。

有三种标准的中断源：软件、时钟和外部来源。软件中断通过向内存映射寄存器中存数来触发，并通常用于由一个 hart 中断另一个 hart（在其他架构中称为处理器间中断机制）。当 hart 的时间比较器（一个名为 mtimecmp 的内存映射寄存器）大于实时计数器mtime 时，会触发时钟中断。外部中断由平台级中断控制器（大多数外部设备连接到这个中断控制器）引发。不同的硬件平台具有不同的内存映射并且需要中断控制器的不同特性，因此用于发出和消除这些中断的机制因平台而异。

八个控制状态寄存器（CSR）是机器模式下异常处理的必要部分：

⚫ mtvec（Machine Trap Vector）它保存发生异常时处理器需要跳转到的地址。

⚫ mepc（Machine Exception PC）它指向发生异常的指令。

⚫ mcause（Machine Exception Cause）它指示发生异常的种类。

⚫ mie（Machine Interrupt Enable）它指出处理器目前能处理和必须忽略的中断。

⚫ mip（Machine Interrupt Pending）它列出目前正准备处理的中断。

⚫ mtval（Machine Trap Value）它保存了陷入（trap）的附加信息：地址例外中出错

的地址、发生非法指令例外的指令本身，对于其他异常，它的值为 0。

⚫ mscratch（Machine Scratch）它暂时存放一个字大小的数据。

⚫ mstatus（Machine Status）它保存全局中断使能，以及许多其他的状态，如图

10.4 所示。

2.2 可重现性技术

关于可重现性的研究体现在以下几种应用场景中：（一）在并行程序调试过程中，循环执行并行程序往往具有随机性，无法得到确定的结果。开发人员使用确定性重放技术在用户系统上记录程序的执行踪迹（Execution Trace），并在开发系统中重现程序崩溃前的状态[3]，为开发人员进行软件漏洞分析提供依据；（二）在分布式系统中，需要可重现性确保副本行为相同，满足分布式一致性的需求；（三）在机器学习、科学计算、大数据分析等计算任务中，模型权重训练过程存在随机性。确定性重放技术可以记录程序执行过程中的执行踪迹，反应模型训练过程中性能变化趋势，帮助研究人员寻找性能变化原因。

分解为2部分：确定性及可移植性。

2.2.1 确定性重放

确定性重放也常被成为记录重放（Record and Replay），即在时间或空间上重复执行一个程序，执行的副本作为原程序的镜像，在相同的输入下应当产生与之相同的输出。数值计算型程序多次执行结果必然相同，但是涉及到系统时间、文件输入、缓冲区等因素影响的程序，执行结果会受到多种因素的影响。因此需要通过记录重放的方法使程序反复执行得出相同的结果。

2.2.2 确定性模型

分析程序执行过程中的不确定性来源：1）具有不确定性的指令；2）中断与异常；3）具有随机性的系统调用等；4）线程间资源竞争；5）输入输出操作；6）线程调度等。

2.2.3 确定性重放实现方法

确定性重放可分别通过硬件支持方法和纯软件实现。

硬件方法：修改体系结构、设计专用芯片等。成本高、增加功耗、灵活性低。

纯软件方法：

系统级别

进程级别

2.2.4 虚拟化与容器

第3章 容器化可重现方法设计与实现

在x86平台上，设计容器化可重现方法。使用纯软件用户空间的方式隔离容器中的程序，以系统调用getpid（获取进程编号PID）为例，使用ptrace监视并拦截容器中运行中程序（称为用户进程）的系统调用。

3.1 基于ptrace的系统调用拦截

定义：

ptrace()系统调用函数提供了一个进程（the “tracer”）监察和控制另一个进程（the “tracee”）的方法。并且可以检查和改变“tracee”进程的内存和寄存器里的数据。它可以用来实现断点调试和系统调用跟踪。

3.2 可重现方法工作流程

通过用户空间隔离进程，使用追踪进程tracer可以拦截被追踪用户进程tracee的系统调用，读取和写入进程B的内存与寄存器。其中可重现的元素被允许通过，不可重现的元素被进行可重现的包装或者被禁止进入容器空间。

3.3 不确定性来源

3.3.1 用户进程编号

通过命名空间隔离进程，获取惟一PID等。

3.3.2 随机函数

拦截具有随机性的系统调用函数，用简单伪随机函数替换。

3.3.3 时间

返回时间信息的系统调用，如，替换为用户进程执行的时间调用的计数。

3.3.4 信号

异步信号。

部分信号是天然可重现的：, 和。

3.3.5 文件和目录

通过隔离用户进程拥有的文件系统。

作用于正在运行的用户进程和它的子进程，改变它外显的根目录，设置后的用户进程不能够对这个指定根目录之外的文件进行访问动作，不能读取，也不能更改它的内容。

3.4 容器中的不确定性来源

……

第4章 基于RISC-V架构的可重现容器化设计与实现

本章提出了一种在资源受限的RISC-V架构上的容器化方法的设计与实现，并将第3章中的可重现容器化方案移植、优化到RISC-V架构平台上。RISC-V是由加州大学伯克利分校提出的开源指令集。作为近年来最火热的开源指令集架构，被广泛应用于各个特定领域的单核或多核微处理器的模块化定制。但是，现有的RISC-V应用部署流程需要将传统软件或模型在RISC-V指令集上重新编译或优化，故如何能快速地在RISC-V体系结构上部署、运行及测试应用程序是一个亟待解决的技术挑战。使用虚拟化技术可以解决跨平台的模型部署及运行问题。但传统的虚拟化技术，例如虚拟机，对原生系统性能要求高、资源占用多，运行响应慢，往往不适用于RISC-V架构的应用场景。

4.1 基于Gem5的RISC-V全系统仿真

在RISC-V硬件平台上实现体系结构实验中的思路需要增加额外的部署成本，因此本文选择使用性能优良的软件模拟器来验证容器化方案。目前有多种软件模拟器支持模拟RISC-V硬件平台，函数级（Function-Level）仿真有Spike[11]、QEMU[12]、FireSim[13]、RV8[14]；寄存器传输级（Register-Transfer-Level, RTL）仿真器包括Rocket[15]、BOOM[16]、Ariane[17]；FPGA级仿真模型Rocket Zedboard[18]等。函数级仿真速度快、易于修改，但无法捕捉目标系统的时序。RTL仿真速度慢、难以修改，但是可以精准模拟目标系统的时序周期。FPGA仿真最准确和快速，但是需要较长的综合和布局布线时间，也更难以修改调试。因此选择离散事件全驱动模拟器Gem5实现RISC-V平台仿真。Gem5模拟器能够以全系统模式模拟多核RISC-V处理器[21][22][23]，支持模拟大部分RISC-V指令和系统调用，支持线程相关系统调用和同步指令，对多核条件下的可重现方法进行模拟验证。

4.1.1 Gem5全系统模拟器

Gem5是一个模块化的离散事件驱动全系统模拟器，它结合了M5和GEMS二者的优势[24]，且高度可配置、集成多种指令集架构和多种CPU模型的体系结构模拟器。Gem5广泛用于计算机体系结构研究，平衡仿真速度、准确性和开发速度。用户可以通过Python接口选择不同型号的CPU、系统模式和内存系统，以实现具有所需要的模拟处理器配置。同时为了保证仿真速度，Gem5的关键性能模块通过C++实现。对于每种指令集架构，gem5分别提供两种模拟模式：系统调用模拟 (System-call Emulation, SE) 和全系统 (Full System, FS) 模拟。之前的工作[21][22]保证gem5能够在 SE 模式下支持模拟大多数 RISC-V 指令和系统调用。SE模式能够快速实现用户工作负载的执行和分析。

Gem5可以通过FS 模式准确模拟系统组件和硬件设备，并加载系统软件（通常是 Linux 内核）。 FS模式支持包括虚拟内存、虚拟化、分布式系统、存储堆栈性能和网络相关等相关功能。Gem5-21.0版本已经支持在模拟RISC-V硬件平台上运行GNU/Linux Busybox发行版，Linux内核版本为5.10[23]。

本章在Gem5模拟器上模拟了一个RISC-V最小系统，包括满足运行引导加载程序和Linux内核所需的最少硬件。由于RISC-V和Gem5的模块化设计，后期可以根据用户需要快速拓展支持的指令集模块和硬件设备。在4.1.2章中详细介绍了模拟的RISC-V目标系统。

在Ubuntu 16.04上，Gem5模拟器上的全系统仿真步骤如下图所示。首先，安装所有必需的依赖项；其次，建立Gem5文件夹，通过Git下载存储库；然后，配置RISC-V选项，编译Gem5源文件，构建RISC-V模拟平台；最后，对Gem5进行测试。

4.1.2 RISC-V目标系统构建

4.2 RISC-V容器化方法

4.2.1 基于命名空间的容器架构

4.2.2 基于QEMU的模拟器

4.2.3 动态二进制指令翻译

4.3 RISC-V容器化方法中的不确定性来源

4.4 RISC-V特权级切换

第5章 实验设计与分析

5.1 软硬件平台

模拟硬件：Gen5模拟器全系统模式下，模拟多核RISC-V处理器。

软件：Linux 4.12内核版本。

5.2实验环境配置

配置流程。

5.3 可重现性方法功能验证

运行包含getid()、等的程序。

5.4 可重现方法性能损耗分析

重复调用，记录执行时间。

第6章 结论与展望

……

……

致 谢

……

xxx（学生姓名落款）

年 月

参考文献

1. Hennessy J L, Patterson D A. A new golden age for computer architecture[J]. Communications of the ACM, 2019, 62(2): 48-60.
2. Celio C, Chiu PF, Asanović K, et al. Broom: an open-source out-of-order processor with resilient low-voltage operation in 28-nm CMOS [J]. IEEE Micro, 2019, 39(2):52-60.
3. O'Callahan R, Jones C, Froyd N, et al. Engineering record and replay for deployability[C]. 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017: 377-389.
4. Narayanasamy S, Pokam G, Calder B. Bugnet: Continuously recording program execution for deterministic replay debugging[C]. 32nd International Symposium on Computer Architecture (ISCA'05), IEEE, 2005: 284-295.
5. Srinivasan S M, Kandula S, Andrews C R, et al. Flashback: A lightweight extension for rollback and deterministic replay for software debugging[C]. USENIX Annual Technical Conference, General Track, 2004: 29-44.
6. Saito Y. Jockey: a user-space library for record-replay debugging[C]. Proceedings of the sixth international symposium on Automated analysis-driven debugging, 2005, 69-76.
7. Dunlap G W, King S T, Cinar S, et al. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay[C]. 5th Symposium on Operating Systems Design and Implementation (OSDI 02). 2002.
8. Sheldon M, Weissman G V B. Retrace: Collecting execution trace with virtual machine deterministic replay[C]. Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2007). 2007.
9. Navarro Leija O S, Shiptoski K, Scott R G, et al. Reproducible containers[C]. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2020: 167-182.
10. Aviram A, Weng S C, Hu S, et al. Efficient System-Enforced Deterministic Parallelism[C]. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
11. Hunt N, Bergan T, Ceze L, et al. DDOS: taming nondeterminism in distributed systems[J]. ACM SIGPLAN Notices, 2013, 48(4): 499-508.
12. Bergan T, Hunt N, Ceze L, et al. Deterministic Process Groups in dOS[C]. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
13. Devecsery D, Chow M, Dou X, et al. Eidetic systems[C]. 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 525-540.
14. A. Waterman and Y. Lee. Spike - RISC-V ISA Simulator[OL]. https://github.com/riscv/ riscv-isa-sim.
15. Bellard F. QEMU, a fast and portable dynamic translator[C]. USENIX annual technical conference, FREENIX Track. 2005, 41(46): 10.5555.
16. Karandikar S, Biancolin D, Amid A, et al. Using FireSim to Enable Agile End-to-End RISC-V Computer Architecture Research[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2019.
17. Clark M, Hoult B. rv8: a high performance RISC-V to x86 binary translator[C]. First Workshop on Computer Architecture Research with RISC-V (CARRV). 2017.
18. Asanovic K, Avizienis R, Bachrach J, et al. The rocket chip generator[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, 4.
19. Asanovic K, Patterson D A, Celio C. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor[R]. University of California at Berkeley Berkeley United States, 2015.
20. Balkind J, Lim K, Gao F, et al. OpenPiton+ Ariane: The first open-source, SMP Linux-booting RISC-V system scaling from one to many cores[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2019: 1-6.
21. Vega L, Taylor M B. RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2017.
22. Roelke A, Stan M R. Risc5: Implementing the RISC-V ISA in gem5[C]. First Workshop on Computer Architecture Research with RISC-V (CARRV). 2017, 7(17).
23. Ta T, Cheng L, Batten C. Simulating multi-core RISC-V systems in gem5[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2018.
24. Hin P Y H, Liao X, Cui J, et al. Supporting RISC-V Full System Simulation in gem5[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2021.
25. Butko A, Garibotti R, Ost L, et al. Accuracy evaluation of gem5 simulator system[C]. 7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC). IEEE, 2012: 1-7.

……

附录A ×××××

……

攻读学位期间的研究成果

**已发表论文：**

徐子晨，崔傲，王玉皞，刘韬. 基于RISC-V架构的强化学习容器化方法研究. 计算机工程与科学[J]. 2021，43(2): 70-74.

XXX，XX. 区域经济相互作用研究[J]. 国内经济与中部经济，2002 (已录用).

……