基于RISC-V架构的强化学习容器化方法研究

崔傲, 徐子晨, 王玉皞

（南昌大学信息工程学院，江西 南昌 330031）

摘 要：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。但是，现有的RISC-V应用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、运行及测试机器学习框架是一个亟待解决的技术挑战。使用虚拟化技术可以实现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。尝试解决在资源受限的RISC-V架构上的强化学习虚拟化问题。首先，通过采用容器化技术，减少上层软件构建虚拟化代价，去除冗余中间件，定制命名空间隔离特定进程，有效提升学习任务资源利用率，实现模型训练快速执行；其次，利用RISC-V指令集的特征，进一步优化上层神经网络模型，优化强化学习效率；最后，实现整体优化及容器化方法系统原型，并通过测试多种基准测试集，完成系统原型性能评估。容器化技术和传统RISC-V架构下交叉编译深度神经网络模型的方法相比，仅付出相对较小的额外性能代价，能快速实现更多、更复杂的深度学习软件框架的部署及运行；与硬件虚拟化方法相比，基于RISC-V的模型具有近似的部署时间，并减少大量的性能损失。初步实验结果证明，容器化及其上的优化方法是解决基于RISC-V架构的软件及学习模型快速部署的一种有效方法。

关键词：虚拟化，神经网络，RISC-V

1. 引言

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]，且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[3]，在边缘微处理器的应用中具有广阔的前景。

深度学习在计算机视觉、语音识别、自然语言处理等领域有着越来越密集的应用。构建一个深度神经网络的工作流通常包括以下几步：（1）收集与准备训练数据；（2） 选择并优化深度学习算法；（3）训练与调整模型；（4）在生产环境中部署模型。近年来，深度强化学习（Reinforcement Learning, RL）在自动驾驶、连续控制等领域的表现优异[4]，但是目前深度强化学习训练任务中的工作负载需要大量的计算资源，甚至可能需要数天时间才能完成。强化学习的训练过程中带有大量的循环，适合在支持乱序执行的RISC-V处理器中进行加速。因此，在基于RISC-V指令架构的平台上构建深度强化学习模型，探索潜在的计算加速具有十分重要的意义。

深度学习的快速部署及推理应用是一个面向RISC-V架构的新挑战。目前传统的基于Python的深度学习框架（TensorFlow、PyTorch、MXNet等）并不支持RISC-V指令架构。在RISC-V平台上运行一个深度神经网络模型的推理过程，研究人员往往需要构建复杂的交叉编译工具链，修改深度学习库中特定的机器源代码，自定义指令集拓展[5]，不利于模型的快速部署和优化。

快速部署的解决方案首推虚拟化技术。传统的虚拟化技术通过虚拟机监视器（Virtual Machine Monitor，VMM，或称为Hypervisor）实现，允许在宿主机设备中运行多个异构的体系结构应用，为用户提供抽象、虚拟的硬件环境。使用Hypervisor实现的虚拟化产品有VMware ® Workstation和Virtual PC等。Hypervisor提供了良好的跨平台兼容性，但是，在RISC-V架构上直接使用虚拟化技术有以下几个问题：（1）每个虚拟机都需要运行一个完整的操作系统以及其中安装好的大量应用程序；（2）资源占用多；（3）运行响应慢[6]。实际开发环境里，我们更关注的是自己部署的应用程序。其次，端设备上的硬件资源可能有限，用户需要使用在操作系统层面实现的更加轻量级的虚拟化技术，在提供高质量的虚拟环境的同时，降低对系统性能的影响。在云计算框架中，面向轻量级软件虚拟化，Pahl等人[7]提出了容器化的解决方案。容器化技术通过名字空间Namespace为每个容器提供特定的命名空间，对进程实现隔离，相对于传统的虚拟机，具有更少的系统占用，更快的启动速度和更高的资源利用率。Docker是目前最为常用的容器技术，在容器的基础上从文件系统、网络互联到进程隔离等等进行了进一步的封装，极大的简化了容器的创建和维护。使得 Docker 技术比虚拟机技术更为轻便、快捷。但Docker尚不支持RISC-V架构，使用容器技术在RISC-V上实现模型的快速部署是一个亟待解决的问题。

本文尝试对基于RISC-V架构的强化学习容器化方法进行研究，首先，通过采用容器化技术，减少上层软件构建虚拟化代价，去除冗余中间件，定制命名空间隔离特定进程，有效提升学习任务资源利用率，实现模型训练快速执行；其次，利用RISC-V指令集的特征，进一步优化上层神经网络模型，优化强化学习效率；最后，实现整体优化及容器化方法系统原型，并通过测试多种基准测试集，完成系统原型性能评估。在原型系统实现里，我们使用QEMU模拟器下仿真的RISC-V指令架构作为实验平台，叠进式设计、实现、测试了多种强化学习优化算法，讨论模型在RISC-V平台上的可移植性和性能表现，

本文的结构如下：第2节讨论了基于RISC-V上虚拟化相关的工作；第3节介绍了容器化方法的设计与实现；第4节对初步的实验结果进行评估；最后一节总结目前得到的结论以及工作的不足之处。

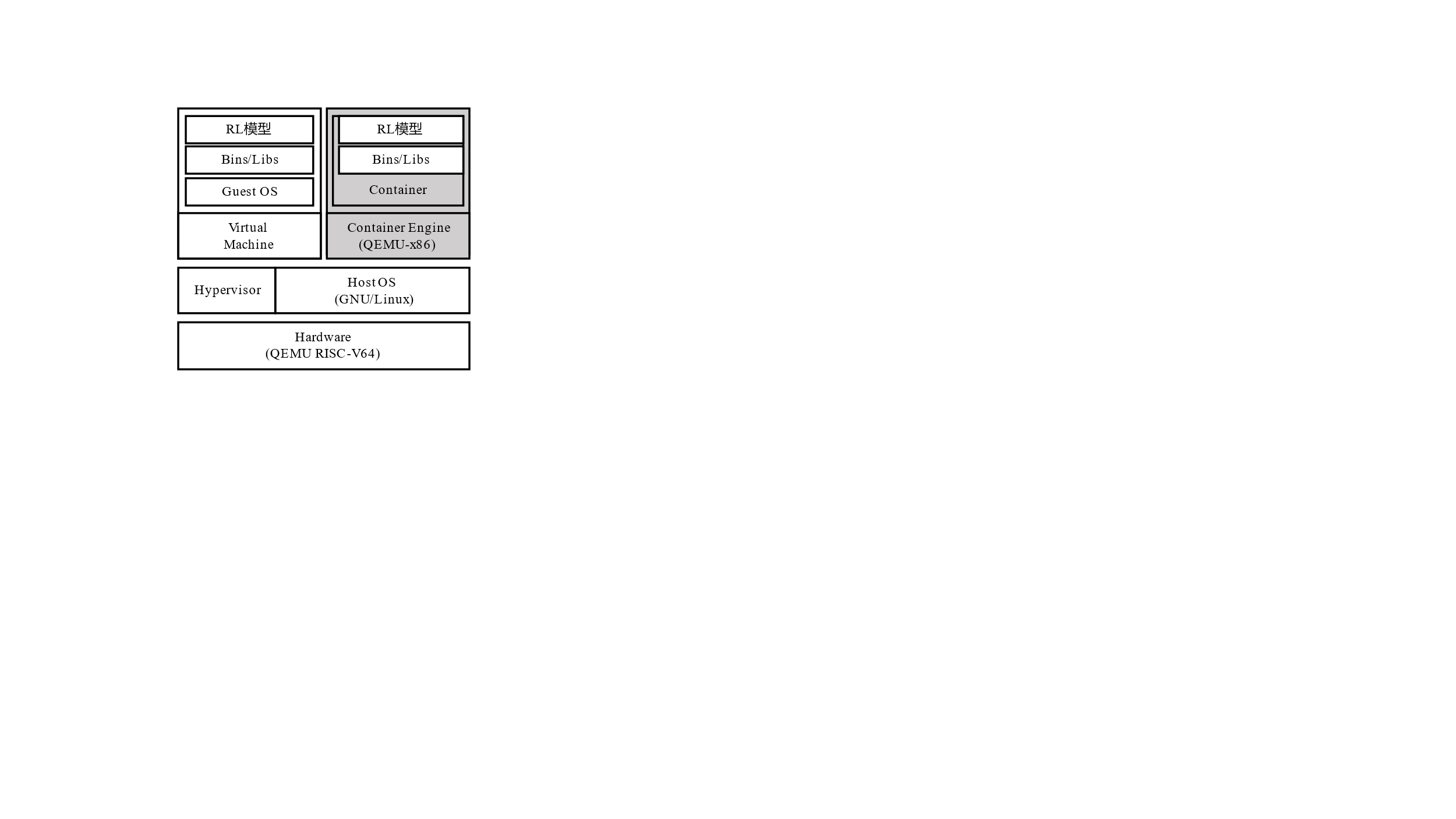
2 相关工作

本文尝试解决在资源受限RISC-V架构上的强化学习虚拟化问题，实现强化学习模型的快速部署。通过使用容器化技术，对多种深度强化学习模型进行封装，在基于RISC-V的操作系统中定制命名空间，隔离特定进程，有效提升学习任务资源利用率，实现模型训练快速执行。

2.1 虚拟化技术

虚拟化技术的目的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。

2.2 基于RISC-V架构的加速及优化

在基于RISC-V平台上有关深度学习的工作大多是将深度学习的计算负载（卷积、激活、池化等）从RISC-V处理器转移到专用的硬件加速器中[5,12,13]，采用软硬件协同设计的方法实现深度神经网络模型推理计算加速。这种方法通常需要根据特定用途设计专用的硬件加速器，同时需要相应的自定义函数库、编译器等工具链。

在基于RISC-V的平台上部署模型最简单的方法似乎是直接在RISC-V中编译深度神经网络模型，但由于硬件资源和模型性能存在限制，实际开发中通常采用交叉编译的方式来部署模型。Kong等人[14]提出了AIRV，一个可以在基于RISC-V硬件平台（FPGA、QEMU模拟器等）部署深度神经网络模型的计算框架，允许在RISC-V平台而不是硬件加速器中运行深度神经网络模型的推理过程。此外还证明相比于直接在RISC-V硬件平台上编译网络模型，在x86平台上交叉编译RISC-V目标架构的深度神经网络模型具有更高的资源利用率。Louis等人[5]使用RISC-V 指令集中的V矢量拓展模块，在此基础上增加一层软件结构，修改了TensorFlow Lite C/C++ 库函数。此外为RISC-V指令集交叉编译了TensorFlow Lite源码，并在RISC-V模拟器Spike上进行验证。这种方法支持在RISC-V上部署多种网络模型，实验结果表明，使用这种方法可以将向处理器提交的指令数减少8倍。Vega等人[15]提出了一种I/O虚拟化的硬件支持RV-IOV，克服部署Rocket内核时的资源限制问题。Rocket Chip是一个开源的System-on-Chip设计生成器，生成使用RISC-V 指令集的通用处理器核心，并提供有序核心生成器（Rocket）和无序核心生成器（BOOM）[16]。RV-IOV使用I / O虚拟化技术将Rocket内核与主机解耦，并使内核可以在ASIC或更大的FPGA中实现。

**图 1 原型系统整体架构。左边是传统的使用Hypervisor的虚拟机架构，右边是容器化架构**

3 设计与实现

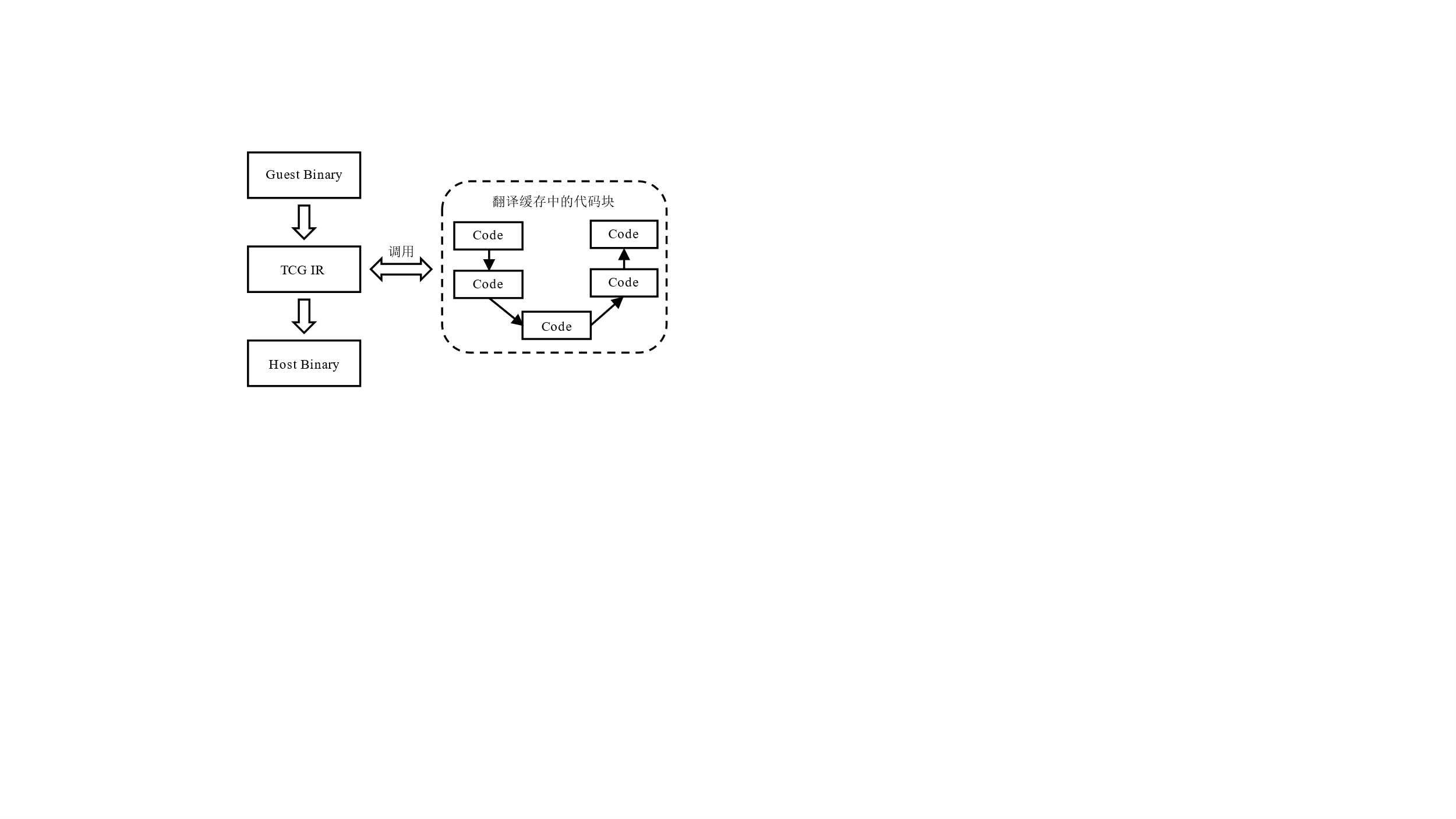
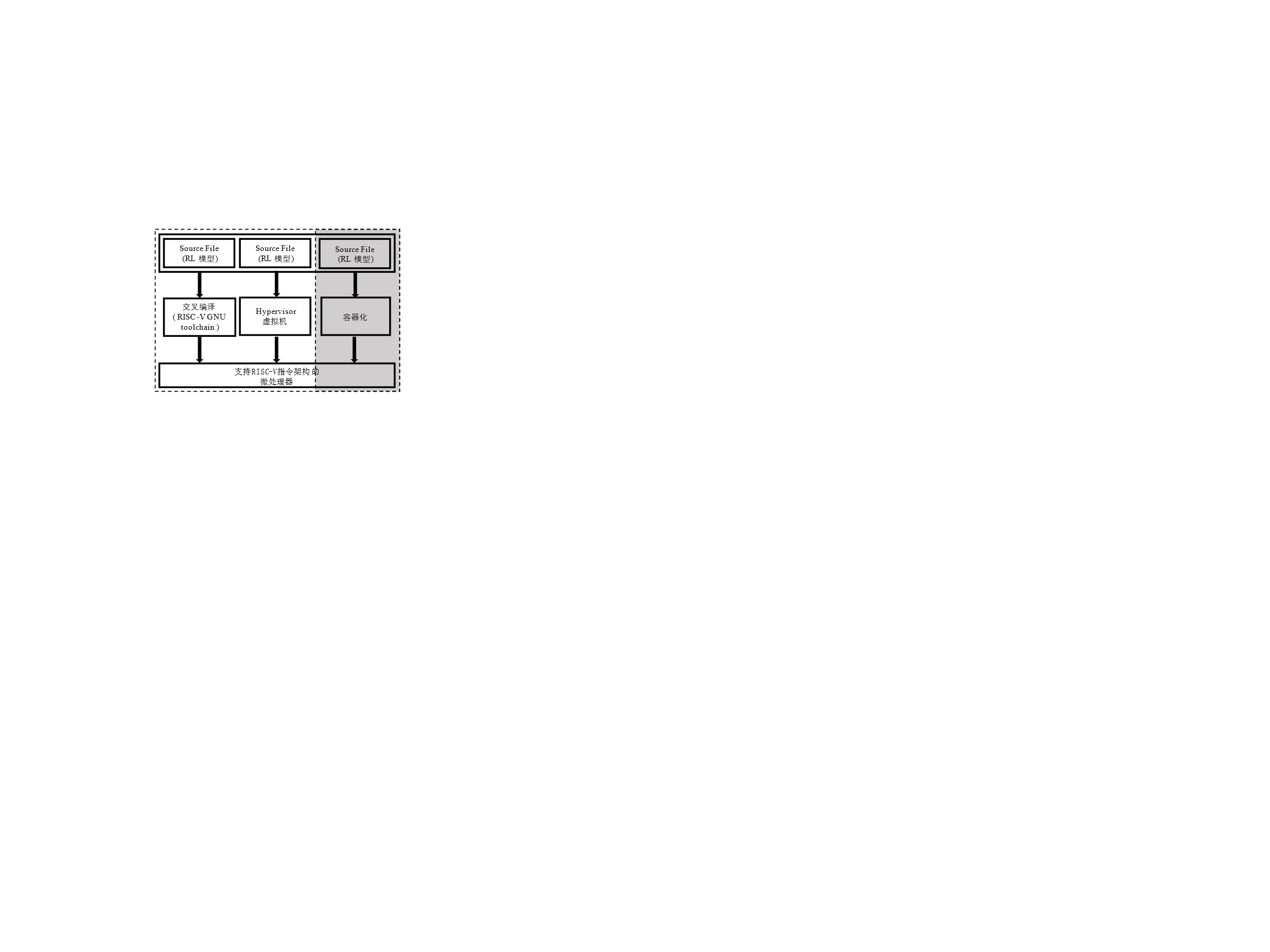
3.1 容器化方法设计

在原型系统设计中，我们使用QEMU作为容器化引擎，定制命名空间隔离特定进程，，在操作系统级别虚拟化方法。图1展示了原型系统的整体架构，左边是传统的使用Hypervisor的虚拟机架构，右边是容器化架构。原型系统使用QEMU模拟RISC-V处理器内核作为实验硬件平台。在RISC-V处理器上运行Linux系统，并安装QEMU。

QEMU是一个可执行硬件虚拟化的开源托管虚拟机，具有跨平台的特性。可通过纯软件方式实现硬件的虚拟化，模拟外部硬件，为用户提供抽象、虚拟的硬件环境。QEMU既可实现全系统硬件虚拟化，也可在User Mode下通过名字空间Namespace为每个容器提供特定的命名空间，对进程实现隔离，实现容器化设计。在 User Mode 下，QEMU不会仿真所有硬件，而是通过内核代码的TCG ( Tiny Code Generator )模块对异构应用的二进制代码进行翻译、转换。异构文件在执行时，通过binfmt\_misc识别可执行文件格式并传递至QEMU中。binfmt\_misc是Linux内核的一种功能，它允许识别任意可执行文件格式，并将其传递给特定的用户空间应用程序，如仿真器和虚拟机。QEMU将注册的异构二进制程序拦截、转换成本地指令架构代码，同时将系统调用按需从目标系统转换成当前系统，并将其转发至本地主机内核。TCG 定义了一系列IR (Intermediate Representation )，已经翻译的代码块放在转换缓存中，并通过跳转指令将源处理器的指令集（ISA）和目标处理器的指令集（ISA）链接在一起。当Hypervisor在执行代码时，存放于转换缓存中的链接指令可以跳转到指定的代码块，并且执行可以在不同的已翻译代码块上运行，直到需要翻译新块为止。在执行的过程中，如果遇到了需要翻译的代码块，执行动作就会暂停并回会跳回到Hypervisor，Hypervisor就会使用和协调TCG对需要进行二进制翻译的源处理器指令集（ISA）进行转换和翻译并存储到转换缓存中。图2展示了TCG的工作示意图。

4 性能评估及分析

4.1 原型设计

设计原型使用 QEMU 模拟的RISC-V64位指令架构4核处理器，内存为2G，主频为1.7 GHz，安装Linux系统及QEMU。图3展示了在RISC-V处理器上部署深度神经网络模型的工作流。我们设置了3组对照实验：（1）交叉编译；（2）Hypervisor虚拟机；（3）容器化方法。我们首先在x86平台下构建强化学习（Reinforcement Learning, RL）模型，分别采用（1）交叉编译的方式[5, 14]部署深度学习网络模型，需要对每个模型单独配置环境，修改强化学习库函数，构建交叉编译工具链，不利于模型快速部署；（2）Hypervisor虚拟机的方式使用QEMU在RISC-V平台上进行全系统模拟，并在虚拟机中安装基于x86架构的操作系统，配置RL模型开发运行环境；（3）容器化方式使用QEMU User Mode在容器化的进程中执行基于x86架构下的模型，模型文件需将运行所需的依赖库封装至二进制可执行文件，实现模型训练快速执行。

**图3 在RISC-V平台上部署深度神经网络模型的三种工作流。从左到右分别为（1）交叉编译（2）Hypervisor虚拟机（3）容器化**

4.2 强化学习算法

强化学习在自动驾驶、连续控制等领域的表现甚至可以和人类相媲美[4]。图4展示了在实验评估过程中模拟连续控制领域的经典问题——Cart-Pole模型：木棍在一个可移动的小车上竖立，通过学习使木棍竖立的时间尽量长。在x86平台使用PyTorch[17]构建深度神经网络模型，在Gym[18]环境下模拟Cart-Pole模型的训练，并部署在RISC-V实验平台上。基准测试集中的强化学习算法包括：（1）随机代理 (Random Policy )；（2）交叉熵 (Cross-entropy)；（3）策略梯度 (Policy Gradient )[19]，叠进式设计、实现、测试了多种强化学习优化算法，完成系统原型性能的评估。

**图2 Tiny Code Generator翻译过程**

**图4 Cart-Pole模型**

**图5 （1）交叉编译（2）Hypervisor虚拟机（3）容器化方法下强化学习模型训练时间**

使用随机代理训练的模型不会收敛，只是记录执行一定步数后的奖励（Reward）总和，木棍无法在小车上竖立后循环停止。基于交叉熵算法可以实现一个稳定收敛的模型，基本思想是：我们使用当前策略（从一些随机的初始策略开始）对事件进行采样，并使用我们的策略将最成功的样本的负面对数可能性最小化，恰好等于最小化交叉熵。具体做法是，首先，在环境和模型上运行N个轮次（Episode），每个轮次都是从开始到结束执行一次算法，计算每个轮次的奖励总和并确定一个奖励边界（如70%）；其次，舍弃所有低于边界值的轮次；然后，在剩余的轮次中进行训练；重复上述步骤直到对结果满意为止。

基于策略梯度的训练和基于交叉熵的训练相比，在轮次的分割中具有更细的粒度。策略梯度的基本原理是通过反馈调整策略：在得到正向奖励时，增加相应的动作的概率；得到负向的奖励时，降低相应动作的概率。策略梯度具有下列优点：（1）更好的收敛性；（2）适合高维度或连续状态空间；（3）不必计算复杂的价值函数。

**图6 部署模型需要修改的代码行数**

4.3 实验结果

我们首先进行了交叉编译、Hypervisor虚拟机和容器化方法的性能比较实验。图5展示了随机代理、交叉熵与策略梯度算法在三种工作流下的不同性能表现，我们比较了强化学习模型在三种工作流下的模型训练时间，其中，交叉编译方式运行的模型训练时间最短；Hypervisor虚拟机的方式运行模型的时间最长，和交叉编译的方式相比大约增加了102倍以上的模型训练时间；容器化方法的模型训练时间居于二者之间，比交叉编译方式约增加了101倍以上的模型训练时间。

图6展示了随机代理、交叉熵与策略梯度算法在三种工作流下部署模型需要修改的代码行数。其中，在所有的算法中，交叉编译方式除了对模型本身文件进行修改，还需要修改深度学习模型依赖库（如Numpy，Scipy，Gym等）中的函数，代码数量最多；使用Hypervisor虚拟机方式只需对模型本身文件进行修改，所需的代码量最少；容器化技术需要将深度学习模型文件和所需依赖库进行封装，无需对库函数进行修改，和Hypervisor虚拟机相比增加了约40%的代码。

我们进一步将端到端的模型训练及运行时延进行了分解分析，如图7所示。三种工作流下建立模型的时间相同，部署到RISC-V指令架构的实验平台的过程中，交叉编译的方式耗费时间最多，Hypervisor虚拟机耗费的时间最少。和虚拟机技术相比，容器化技术在模型部署过程中多耗费了约30%的时间开销；和交叉编译相比，减少了约85%的时间开销。

**图7 模型部署时间为。比较三种工作流下建立模型、在RISC-V部署模型和训练模型的总时间**

4.4 评估分析

图8综合了图5至图7的实验结果，X轴表示模型部署时间，Y轴表示模型训练时间，气泡大小表示模型部署包括的代码数量。其中，容器化方法在三个指标上均处于中间位置。和交叉编译的方式相比，容器化付出了约10倍的性能代价，减少了85%的模型部署时间和95%以上的代码数量；和虚拟机的方式相比，仅增加了约30%的模型部署时间和约40%的代码数量，减少了102倍的模型训练时间，大幅提升了模型性能。

**图8 综合实验结果。三种工作流下模型部署时间、代码量和部署后的模型训练时间**

初步的实验结果表明，使用交叉编译方式部署模型虽然可以得到最高的模型性能，但部署周期最长，这是因为在移植模型的过程中需要对库中所有特定的机器源代码进行修改；在设置完毕虚拟机环境后，硬件虚拟化的方式可以大幅减少模型部署时间的消耗，但每次模型训练都需要运行完整的虚拟机和相关环境，和交叉编译方式相比会带来额外的资源占用；容器化技术和传统RISC-V架构下交叉编译深度神经网络模型的方法和使用Hypervisor虚拟机的方式相比，仅付出相对较小的额外性能代价，在近似的部署时间内，实现更多、更复杂的深度学习软件框架的快速部署及运行。初步实验结果证明，容器化及其上的优化方法是解决基于RISC-V架构的软件及学习模型快速部署的一种有效方法。

5 结束语

使用虚拟化技术可以实现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。尝试解决在资源受限的RISC-V架构上的强化学习虚拟化问题。首先，通过采用容器化技术，减少上层软件构建虚拟化代价，去除冗余中间件，定制命名空间隔离特定进程，有效提升学习任务资源利用率，实现模型训练快速执行；其次，利用RISC-V指令集的特征，进一步优化上层神经网络模型，优化强化学习效率；最后，实现整体优化及容器化方法系统原型，并通过测试多种基准测试集，完成系统原型性能评估。容器化技术和传统RISC-V架构下交叉编译深度神经网络模型的方法相比，仅付出较小的额外80%性能代价，能快速实现更多、更复杂的深度学习软件框架的部署及运行；与硬件虚拟化方法相比，基于RISC-V的模型具有近似的部署时间，并减少大量的性能损失。初步实验结果证明，容器化及其上的优化方法是解决基于RISC-V架构的软件及学习模型快速部署的一种有效方法。

目前在RISC-V平台上对各种虚拟化方案性能方面的探索仍有待于进一步研究，未来我们会对深度神经网络模型进行量化、减枝等操作，针对特定领域对模型进行专门优化，构建模型制作、镜像打包、容器化技术部署在RISC-V硬件平台上的完整工作流。

参考文献：

1. Hennessy JL, Patterson DA. 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. Greengard S. Will RISC-V revolutionize computing? [N]. Commun. ACM 63, 5 (April 2020):30–32.
4. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning [C]//Proc of the ICLR (Poster), 2016.
5. Louis MS, Azad Z, Delshadtehrani L, et al. Towards Deep Learning using TensorFlow Lite on RISC-V [C]// Proc of the ACM CARRV, 2019.
6. Ramakrishnan, J, Shabbir, M. S., Kassim, et al. A comprehensive and systematic review of the network virtualization techniques in the IoT [J]. International Journal of Communication Systems, 2020, 33(7):e4331.
7. Pahl C. Containerization and the PaaS cloud [J]. IEEE Cloud Computing. 2015, 2(3):24-31.
8. Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation architectures [J]. Communications of the ACM. 1974, 17(7):412-21.
9. Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization [C]//Proc of the ASPLOS, 2006, 41(11):2-13.
10. Shuja J, Gani A, Bilal K, et al. A survey of mobile device virtualization: Taxonomy and state of the art [J]. ACM Computing Surveys (CSUR). 2016, 49(1):1-36.
11. Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing. 2014, 1(3):81-4.
12. Davidson S, Xie S, Torng C, et al. The Celerity open-source 511-core RISC-V tiered accelerator fabric: Fast architectures and design methodologies for fast chips [J]. IEEE Micro. 2018, 38(2):30-41.
13. Flamand E, Rossi D, Conti F, et al. GAP-8: A RISC-V SoC for AI at the edge of the IoT [C]//Proc of the IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP) 2018:1-4..
14. Kong Y. AIRV: Enabling Deep Learning Inference on RISC-V [R]. 2019 Bench Council International Artificial Intelligence System Challenges.
15. Vega L, Taylor M. RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization [C]// Proc of the ACM CARRV, 2017.
16. Asanovic K, Patterson DA, Celio C. The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor [R]. University of California at Berkeley United States; 2015 Jun 13.
17. Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library [C]//Proc of the NeurIPS 2019:8024-8035.
18. Brockman G, Cheung V, Pettersson L, et al. OpenAI Gym [J]. CoRR 2016.
19. Mirhoseini A, Pham H, Le QV, et al. Device placement optimization with reinforcement learning [C]//Proc of the ICML 2017: 2430-2439.