分 类 号_____ 学 号 M201777257

学校代码 10487 密 级 _____

華中科技大學

硕士学位论文

ARM 平台下的虚拟化实现及应用

学位申请人: 黄宇飞

学 科 专 业 : 计算机技术

指 导 教 师 : 付才 教授

答 辩 日 期 : 2019 年 5 月 28 日

A Thesis Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Engineering

Virtuailzation on ARM machine platform

Candidate : Huang Yufei

Major : Computer Technology

Supervisor : Prof. Fu Cai

Huazhong University of Science and Technology

Wuhan, Hubei 430074, P. R. China

May, 2019

独创性声明

本人声明：所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本课题的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：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。本人授权华中科技大学可以将本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和汇编本学位论文。

保密□，在_____年解密后适用本授权书。

本论文属于

不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

2007 年，苹果公司发布了革命性的产品——iPhone 智能手机，从此开启了智能手机的时代，十多年来，随着智能手机的日益普及，基于 ARM 指令集架构的处理器的嵌入式计算机数量也越来越多，性能也越来越强大，在 ARM 平台性能强大的同时，作为重要应用的虚拟化技术也开始渐渐被人关注，哥伦比亚大学在 2014 年的时候提出了一个开源计划：KVM/ARM，旨在在 ARM 上实现 KVM 虚拟机加速。

论文将在现有的 ARM 虚拟化技术支持下，将该技术应用于实际场景中，在目标机器上，修改 KVM 源码，开启 KVM 虚拟化加速功能，将服务器操作系统安装在虚拟机上，同时将 UEFI 交叉编译成 ARM 版本并安装操作系统，通过目标机器的处理器提供的硬件双屏异显功能，并分析 QEMU 显示设备工作原理，修改 QEMU 显示设备的代码，将双屏异显功能实现于虚拟机上，最后，将物理设备直通功能应用于 ARM 平台下，通过 vfio 技术实现将指定的 PCI-e 设备直通进入虚拟机，从而实现硬件直通功能，同时，对以上实现的几个功能进行实际验证，通过软件跑分、实机运行等多种方式来验证其是否达到目标要求。

通过以上的各功能的实现，来证明在 ARM 下同样可以实现类似 X86 的虚拟化效果，并在此基础上进行扩展，并且对于 ARM 进军个人 PC、服务器等领域提供了一种可能性，也为工业控制领域、汽车工业领域、区块链以及数据中心等领域提供了一些 ARM 下的行业解决方案，为 ARM 的应用生态做了一些贡献，使得 ARM 平台的应用领域更为宽广。

关键词：嵌入式虚拟化，双屏异显，统一的可扩展固件接口，硬件直通

Abstract

In 2007, Apple released a revolutionary product, the iPhone smartphone, which opened the era of smartphones. Over the past decade, with the increasing popularity of smartphones, embedded computers based on ARM instruction set architecture processors. The number is also increasing, and the performance is getting stronger and stronger. While the performance of the ARM platform is strong, the virtualization technology as an important application is gradually getting attention. Columbia University proposed an open source plan in 2014: KVM. /ARM, designed to achieve KVM virtual machine acceleration on ARM.

The paper will be applied to the actual scene with the support of the existing ARM virtualization technology. On the target machine, modify the KVM source code, enable the KVM virtualization acceleration function, and install the server operating system on the virtual machine. UEFI cross-compiles into ARM version and installs the operating system, through the hardware dual-screen display function provided by the processor of the target machine, and analyzes the working principle of QEMU display device, modifies the code of QEMU display device, realizes dual-screen display function in virtual On the machine, finally, the physical device pass-through function is applied to the ARM platform, and the specified PCI-e device is directly connected to the virtual machine through the vfio technology, thereby realizing the hardware straight-through function, and at the same time, actually verifying several functions implemented above. , through software running points, real machine operation and other ways to verify whether it meets the target requirements.

Through the implementation of the above functions, it is proved that the virtualization effect similar to X86 can be realized under ARM and expanded on the basis of this, and it provides a possibility for ARM to enter the field of personal PC, server, etc. Some areas of ARM's industry solutions are available in the areas of control, automotive, blockchain and data centers.

Keywords:ARM virtualization, dual-screen different display, UEFI, hardware pass-through

目 录

1 绪 论

1.1 课题背景

在苹果公司发布了第一代 iPhone 以及 iPad 后，基于 ARM 指令集的 cpu 就越来越受到各大嵌入式设备厂商的青睐，ARM 平台的设备也越来越多，从最开始的掌上游戏机、工业控制、家电控制到现在的智能手机、平板电脑等。

在设备呈现爆炸式增长的同时，ARM 公司也在对 ARM 架构做出一系列性能增强以及架构更新，2012 年推出 Cortex 系列，将原来的 ARM 架构处理器分为了三种不同定位的产品，其中 A 系列用于消费电子等高性能计算领域，而 R、M 系列则坚持原来的工业控制嵌入式定位，2013 年在 ARM 上引入了“多核概念”使智能手机进入多核时代，2014 年推出了 v8 指令集，ARM 正式进入 64 位时代，最近，ARM 推出了全新的架构 Cortex-A76，其搭载的超线程技术，将 PC 的超线程技术运用到了嵌入式设备中，其宣称搭载该技术后，相较前代产品而言，性能有 1.8 倍的提升。甚至可以达到 PC 机的性能效果。

作为 ARM 处理器的的生产厂商们，也在 ARM 公司的架构基础上进行一系列的商用产品开发，配合 ARM 公司提高了 ARM 架构处理器的性能，打算和 intel 公司的 x86 架构处理器产生竞争效果，高通公司与微软合作推出了适用于 ARM 架构的 windows 10 和 windows server 操作系统，为此，高通公司推出了骁龙 1000 处理器，声称可以媲美 intel 公司的酷睿 i3 处理器。同时，华为公司推出了面向服务器的泰山处理器，旨在挑战 intel 在服务器市场的霸主地位。

在 ARM 处理器性能越来越高的背景下，很多针对 PC 软件开发的开发者们开始将目光转向 ARM 平台，例如手机软件的数量现在已经超过了 PC 软件，linaro 公司甚至直接将 gcc 编译器移植到了 ARM 平台，足以证明 ARM 的性能已经可以与 PC 机媲美。同时，作为重要应用之一的虚拟化也开始受到关注，2014 年，哥伦比亚大学提出了一个新的项目：KVM/ARM，旨在在 ARM 平台的 linux 系统下使用 KVM 加速，Xen 也针对部分 ARM 开发板平台推出了自家的 Xen 加速方案。2013 年，一个叫 virtual open system 的开源团队宣称他们率先在 vexpress-a15 平台（32 位 ARM 处理器）上成功运行了 KVM 加速。

1.2 国内外现状研究

目前对于 ARM 下的虚拟化研究尚处于初始阶段，从硬件方面看，2015 年 ARM 推出了 v8 指令集的强化版本——v8.1 指令集，提供了硬件级别的虚拟化支持。

在 2018 年底，2019 年初，ARM 公司推出了最新版本的 Cortex-A 架构——Cortex-A65AE，该架构定位于车载系统，其中最有亮点的地方就是将超线程技术首次应用在 ARM 架构的 CPU 上，而在此之前，只有 intel 的 X86 架构处理器使用了超线程技术。通过超线程技术，可以大大提高 ARM 处理器的性能。

从软件方面看，哥伦比亚大学的 KVM/ARM 项目虽然在 2014 年启动，但目前研究在 2016 年后就没有任何更新，virtual open system 团队仅仅只做了 32 位下的虚拟化实现，但在 ARM 公司推出 64 位指令集后，对其虚拟化则并未做任何进一步的实现。

Linux 内核在 3.0 版本后加入了对于 64 位 ARM 架构下的 KVM 支持（32 位仅仅只有一款可以支持，即 Cortex-A15），同时作为开源虚拟机软件的 QEMU 则在软件层面推出了 ARM 架构下的虚拟机，并且可以使用 KVM 以及 Xen 的加速方案。

Google 公司为了推广自己的安卓操作系统，在 QEMU 的基础上进行了一系列修改，推出了 android 模拟器，改模拟器可以在 x86 机器上模拟运行 ARM 架构的虚拟手机，从而帮助 android 开发者在没有安卓设备或者涉及硬件安全的情况下方便调试其所开发的安卓应用程序。但由于该模拟器只能在 x86 的计算机上运行，所以无法使用硬件加速，而是纯软件模拟，所以运行速度极慢。

作为 ARM 和其他厂商，例如 IBM、高通、华为、MTK 等公司一起投资组建的 ARM 平台相关技术开源公司 linaro，其打算在 qemu 基础上进行一系列的修改，意图将目前的虚拟化技术运用到 ARM 服务器中，实现类似虚拟主机一样的效果。这样就可以在 ARM 服务器上实现数据中心所需要的 VPS 功能，并且可以将这些虚拟机租售给客户使用。

1.3 主要内容

本文将在现有的 ARM 虚拟化技术的研究成果上进行进一步的扩展和优化，首先就是将 KVM 在指定的开发板上面可以运行，并且可以使用 QEMU 在开发板平台系统上启动虚拟机并使用 KVM 加速，达到性能提升的效果，其次就是将 UEFI 固件移

植到 ARM 平台上，使得 QEMU 模拟的 ARM 虚拟机可以启动 UEFI 引导，从而引导 linux 系统启动。

其次，本文将分析并实现在虚拟机上，基于双屏异显技术，将虚拟机的显示内容，单独投射到指定的屏幕之上，实现物理机与虚拟机双屏异显的效果

最后，研究在 ARM 下的硬件直通技术，尝试 ARM 架构下的虚拟机是否同样可以支持硬件直通，为未来 ARM 服务器下的实际应用打下基础。

1.4 实现难点

由于 x86 与 ARM 的架构完全不同，而且相比 ARM 虚拟化来说，x86 虚拟化虽然成熟但很多技术无法用于 ARM 下，因此会有以下几个难点：

(1) ARM 与 x86 的机器结构有很大不同，x86 的机器经过了标准化，各方面的接口特性都是标准化的，所以 x86 所有机器都是“兼容机”，即使不同厂商也可以相互兼容操作系统和驱动，而 ARM 的机器并未经过标准化，所以各大厂商的除了 CPU 部分均为 ARM 处理器之外，其他的器件和接口都是厂商自行定义的，因此与 x86 不同，操作系统的移植需要器件厂商根据自定义接口移植，同样的，ARM 的虚拟机也必须单独设计机器结构，这一点在 QEMU 上面就有体现，QEMU 对 ARM 的模拟是通过将市面上多个主流 ARM 机器进行直接模拟，需要模拟时必须指定其实现的机器型号，不同的机器对于设备的支持也不同，而不是像 x86 那样不需要指定机器型号直接挂载设备即可。

(2) 双屏异显的功能在某些应用中实现过，但是大多是播放视频或者 PPT 幻灯片之类的，尚未有人将虚拟机的屏幕投射到第二块屏幕中，而且 QEMU 并未提供这种虚拟机屏幕读取的接口供其他软件使用，如何将虚拟机的内容实时读取出来，并且投射到第二块屏幕上，是一个不小的挑战。

(3) 目前的虚拟硬件直通技术大多基于 SR-IOV，这是一个在 intel 下可以实现的技术，然而该技术仅仅只能在 X86 下实现，而且是 intel 提供的硬件实现，目前看来在 ARM 下需要实现硬件直通技术，需要其他的方案。

1.5 文章结构

本文共分为八个章节，各章节的主要内容如下：

第一章 绪论：本章主要介绍课题背景研究意义以及本论文的实现难点以及对实际应用上作出的贡献。

第二章 相关技术分析：本章是对于虚拟化技术、QEMU 设备模拟以及 ARM 下的一些技术和虚拟机硬件直通技术进行分析介绍。

第三章 虚拟化总体结构设计，本章确定虚拟化所使用的硬件平台和虚拟化技术以及操作系统之上所选用的虚拟机软件，确定结构后对需要实现的目标进行确定。

第四章 UEFI 的移植，本章首先介绍了 UEFI 在 ARM 下移植的必要性，然后描述了 UEFI 的移植实现过程，以及其中的一些难点，最后在 QEMU 上运行 UEFI 来测试是否成功移植，为后面章节的性能测试提供了基础。

第五章 ARM 下的 KVM 加速优化及测试对比，本章将讨论 ARM 下 KVM 的部分实现原理以及解决指定 CPU 型号下的 KVM 性能优化，再将 KVM 模块添加进开发板中，通过在操作系统层之上的跑分软件结果对比证明修改后的 KVM 对于虚拟机性能有明显的优化。

第六章 虚拟机双屏异显的实现：本章首先介绍如果要做到双屏异显所需要的硬件要求和 x86 的不同之处，以及虚拟机实现双屏异显的具体方案和实现实现过程以及最终效果。

第七章 虚拟机硬件直通的实现：本章将介绍在 ARM 下实现硬件直通技术的可能性，然后在内核中调通 ARM 下的 VFIO 硬件直通以及 IOMMU 技术，同时在 ARM 上尝试接入 PCI-E 设备使其正常运行，最后在 QEMU 上直通 PCI-E 设备，测试性能并得出结论。

第八章 总结与展望：本章对本文所做的工作进行总结，归纳文章的工作与创新点，并分析文章的不足之处，提出下一步的研究目标。

2 相关技术分析

本章将介绍 ARM 下虚拟化所涉及到的一些技术资料信息。通过本章的介绍将其中一些经常使用到的技术进行分析。

2.1 虚拟化技术

虚拟化技术，就是在一台物理机上虚拟出多台逻辑计算机，在一台计算机上可以同时运行这些逻辑计算机，这些虚拟机之间的运行不会相互影响[1]，而且即使虚拟机出现宕机或者其他故障也不会影响到物理机的运行。

2.1.1 虚拟化技术的应用

虚拟化技术应用在如下方面[2]：

（1）数据中心领域，很多网络软件需要通过服务器来运行软件的服务端，但是大多数情况下，会出现两种情况：第一种是对于资源要求不高的服务端，单个服务器硬件过剩，不能得到有效利用，但为了防止资源争抢和由于一个服务端将整个操作系统崩溃从而影响其他服务端的正常运行，因此也不能在同一台服务器运行多个不同的服务端程序来解决问题。第二种情况是单个服务器资源不够，需要将多个服务器的资源“虚拟”成一台服务器的资源，这两种情况都需要虚拟化来解决问题，而且对于 IDC（数据中心）来说，将一台服务器分割为多个虚拟机，不仅可以优化资源配置，而且相对于直接整租一台服务器的利润而言，一台服务器运行多个 VPS 所带来的收益是更高的。

（2）网络空间安全领域，在当前的应用程序越来越多的情况下，对于用户而言，并不知道哪个软件是有害的，或者该软件可能带有一定的操作风险，例如宕机之类的，为了防止这种情况，虚拟化技术可以提供一个与物理机完全隔离的环境，让未知应用可以在虚拟机的环境里运行，从而不干涉到物理机的运行安全；同时，也可以为网络安全工程师研究恶意软件提供一个安全可靠地环境，不会影响到生产环境的安全。

（3）汽车工业领域，由于 android 操作系统的普及，大量娱乐应用在 android 上面出现，甚至使得汽车制造公司开始考虑将 android 操作系统移植到他们的车载电脑中，但由于 android 操作系统主要用于手机、平板等家用计算机领域，在安全性、可靠性、稳定性上远远达不到车载系统的要求，各大汽车制造公司对此持有观望态度，而虚拟化是解决这一矛盾的方法之一，其中，黑莓公司为这些车载企业推出了一套虚拟化系统 QNX 来解决问题，然而这套系统相对较贵，而且也不能和当前的 linux 系统兼容。因此汽车厂商们也在寻找一套可以通用的 linux 虚拟化技术。

（4）企业商务工作领域，不少的企业开始担心他们的计算机不可靠，从而造成数据丢失、运算中断等，影响企业工作效率，因此，一些企业将虚拟化引入到他们的工作中，也就是虚拟云桌面技术，将所有工作全部放在虚拟机中，并且随时保存数据快照备份，一旦遇到灾难事件可以随时 100%恢复当前运行状态，而且，通过虚拟机的制备也减少了企业对于计算机的投入成本，以前，每位员工都必须配备一台主机，现在，只需要几台服务器即可。

（5）个人应用领域，虚拟化在个人的领域中也应用十分丰富，例如对于只有一台计算机的用户来说，需要同时使用 windows 操作系统和 linux 操作系统，那么虚拟化将为其解决这一问题，不仅仅可以虚拟同架构计算机，而且还可以虚拟异构计算机，例如一些游戏机的模拟器，游戏机的体系结构、软件都和一般 PC 不同，将整个硬件虚拟在 PC 上，从而使个人玩家可以在 PC 上运行这些游戏机的程序来玩游戏机上的程序（事实上，这种行为可能会触犯游戏机公司的版权，引发法律问题，因此这里只讨论技术可能性）。

2.1.2 主流虚拟化技术介绍

由于虚拟化的重要作用，因此成为了很多软件公司的研究课题，也因为这些软件公司的研究，也出现了很多应用于实际的虚拟化技术：

（1）KVM：kernel-based virtual machine，这是一个开源的虚拟化技术[1]，最早是由一家以色列公司所开发，当时为了降低开发的难度，而选择了基于 linux 内核下作为 linux 内核中的一个模块进行开发，当此产品推广出来的时候，KVM 也引起了不小的轰动，最后 KVM 模块的源代码被 linux 内核所接受，成为 linux 内核源代码的重要组成部分，同时作为开源软件界的龙头企业，红帽公司收购了这家以色列公

司，并且商业发行版的 linux——RHEL6.0 版本中，将原来使用的 xen 替换为 kvm，并且在此基础上，推出了商业虚拟化管理软件 RHEV，其开源免费版就是现在很著名的 Ovirt 项目。同样，oracle 公司的 virtualbox 也是基于 KVM 所制作的

（2）Xen：Xen 是英国剑桥大学所开发的虚拟化技术[3]，采用 ICA 协议，其特点在于需要将操作系统进行显示的修改才能在 xen 上运行，虽然这样做，可能移植操作系统有点麻烦，但是经过修改后的操作系统，其运行效率甚至在没有虚拟化硬件支持（例如 intel VT 虚拟化技术）下都是相当高的。Xen 引入了一个“半虚拟化”的概念[4]，相比较 KVM 的全虚拟化将虚拟机指令集完全翻译成物理机器指令集，Xen 是将操作系统所发出的指令最优化，便于在虚拟机中执行[5]。然而由于 Xen 的这种方案，对于 linux 而言相当于重新替代了内核中的资源管理架构而遭到了 linux 内核开发工作组的不满和抵制，因此至今未纳入到 linux 内核中。

（3）Vmware：vmware 是一种商业化的闭源虚拟化技术[6]，是由几位来自不同高校计算机专业的工程师所创建的，其公司总部位于美国加州，后被 EMC 公司收购，是目前当前市场份额极大的虚拟化技术之一，其主要定位于 X86 架构下的虚拟化，所虚拟的目标机器也是 x86 架构，最初的 vmware 只是在 windows 或者 linux 下的一个应用软件，后来退出了自己的虚拟机专用操作系统 esxi[7]，该系统能够做到比在通用操作系统上使虚拟机获得更高的效率（特别是存储方面），同时该公司基于自己的虚拟化技术，还推出了很多商用产品，例如 vmware horizon 虚拟化桌面，能够帮助大型企业部属自己的虚拟化桌面，是一种成熟、完善的商业化方案。

（4）Hyper-V：Hyper-V 是微软公司随 windows server 2008 操作系统捆绑推出的一个基于 X86 的虚拟化技术[8]，前身是 Microsoft virtual PC，其创建之初的定位是与其他商业化的虚拟化技术（例如 xen 和 vmware）进行竞争，同时也是为了与 linux 系统进行抗衡（在此之前，linux 早已将 kvm 虚拟化纳入内核之中），设计目的是为了使用户能够更为熟悉的接触虚拟化技术，而不需要去购买其他公司的虚拟化产品，达到降低企业成本支出的目的[9]，具有戏剧性意义的是，hyper-v 技术是可以创建运行 linux 操作系统虚拟机的，微软相信他们的 hyper-v 在 windows 操作系统下的性能是其他虚拟化技术所不能比的，其虚拟化架构如图 2.1 所示。

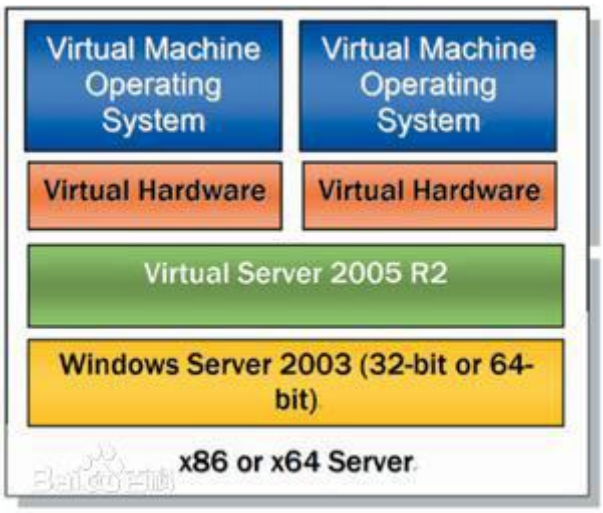

图 2.1 微软 Hyper-V 技术的系统结构

2.2 ARM 处理器及相关技术

ARM 最初是由英国 acorn 公司（现已更名为 ARM 公司）推出的一种低功耗 RISC 微处理器架构，其特点具有能耗低，功能强[10]，最开始出现在一些家用计算机上，可以运行 unix 系统，但由于基于 intel x86 架构的计算机快速占据市场份额，导致 ARM 处理器不得不退出这块市场，转而在工业控制、单片机和个人手持 PDA 设备领域发展[6]。

2008 年，随着苹果公司推出划时代的产品 iPhone 以及 iPad 后，作为幕后英雄的 ARM 公司慢慢浮出水面，在此之后其性能水平开始出现明显的飞跃，2015 年左右推出了 ARM V8 指令集，开始向 x86 传统应用领域发展。其中就包括虚拟化技术。

2.2.1 ARM v8 指令集

在 V8 指令集以前，ARM 仅仅支持 32 位处理技术，但是 v8 指令集推出后，ARM 处理器开始支持 64 位处理技术，在该架构下，处理器分为 2 个状态，32 位状态和 64 位状态[11]，并且引入了多种新的技术，如：TrustZone 技术、NEON advanced SIMD 技术等。

2.2.2 ARM v8 下的虚拟化技术

在 v8 指令集下，虚拟化技术已经成为了整个系统架构的一部分，进入了 EL2 特权级别中，该模式仅解决了 CPU 访问内存等外围设备的问题，其系统结构如图 2.2 所示。

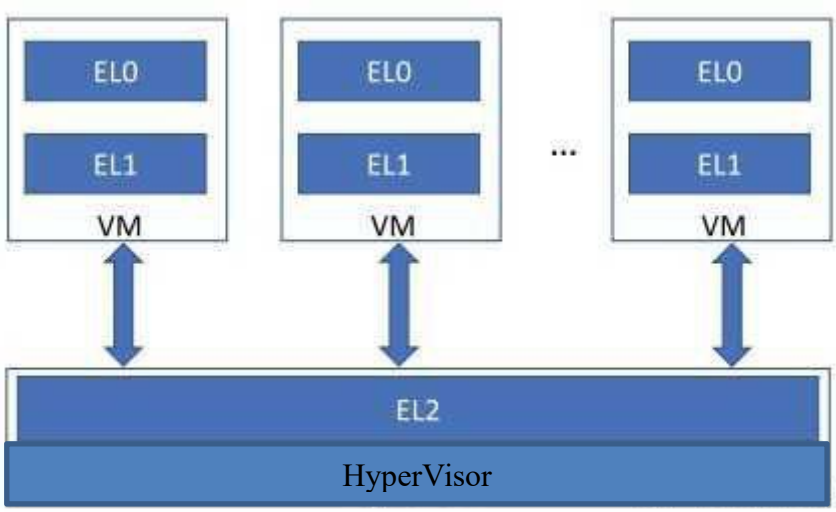

图 2.2 ARM v8 指令集的虚拟化系统结构

上图说明了整个 ARM 虚拟化的体系结构，EL2 层运行虚拟机的管理程序，控制虚拟机的执行和资源共享[12]，至于 EL1 层（操作系统层）和 EL0 层（用户空间层）的具体实现则交给虚拟机实例去实现。地址转换分位两级转换，第一级转换，使用第一级转换表从虚拟地址计算出中间的物理地址；而第二级转换，使用管理程序准备的第二级表计算实际物理地址。这样的架构可以保证虚拟机和物理机的运行在逻辑上是完全隔离的。

对于内存和外设的管理，ARM V8 架构也提供了类似 intel VT-D 的通用中断控制器（GIC）和系统内存管理单元（SMMU）两个管理单元：

（1）SMMU 执行 I / O 地址转换的方式与 CPU 启动的存储访问相同。该单元支持 I / O 地址的一级和两级转换。因此，可以在 VM 和管理程序中使用转换和保护内存区域的优势。因此，允许设备仅对特定存储器地址范围进行读/写。

（2）GIC 实现了中断的虚拟化，在 V8 架构中，虚拟中断可以分为两个虚拟组的一个：0 和 1，前者保存快速中断请求，而后者保存其他的中断请求，处理虚拟中

断的方式与物理中断方式几乎一致。来自设备的中断被发送到 EL2 级别的虚拟机管理程序，当该中断用于虚拟机管理程序时。则管理程序激活虚拟处理器所对应的虚拟中断。

2.3 应用软件层的虚拟化技术

除了在操作系统层之下的虚拟化支持外，在操作系统层之上也需要对应软件来支持虚拟化。本节将介绍所需要使用的虚拟化应用软件技术。

2.3.1 QEMU

QEMU 是由 Fabrice Bellard 所开发的开源模拟器，在 linux 平台上使用非常广泛，其工作在操作系统内核之上，所以实际上所实现的是软件层面的虚拟化实现，必须调用操作系统接口来完成虚拟化工作，将虚拟 CPU 指令转换为对应的操作系统 API，因此实际上性能是不高的。

QEMU 可虚拟多种平台，例如 X86、X86_64、ARM、MIPS 等多种指令集的虚拟机，其分为两种运作模式：

（1）User mode（用户模式）：在该模式下，qemu 将那些已经编译成某种机器指令集的本地二进制代码的 linux 程序，在另一台机器上的 linux 系统之上运行，这种模式仅仅只是做了指令集的翻译工作，因此，该模式下的程序不一定能保证正常运行，但资源消耗较低。

（2）System mode（系统模式）：在该模式下，qemu 模拟的是某一个机器架构的整个机器运行环境（CPU、内存、I/O 等），然后在这个环境中安装操作系统，再在该操作系统上运行用户程序，所以在这种模式下运行的用户程序几乎是和真机一样的（指功能而非性能上），但资源消耗较高。

本文所有的开发和测试都是在 System Mode 下进行的，因此将对 System mode 进行详细介绍。

2.3.2 QEMU-KVM

QEMU 自身的 System Mode 实际上是一个软件虚拟模式，不管是 CPU、内存、

I/O 的模拟都是在操作系统层之上的，其效率较低，然而，前一章所介绍的 KVM 虚拟化仅仅实现了 CPU 和内存的虚拟化实现，因此还需要一个运行在用户空间，可以虚拟 I/O 设备的开源软件，所以 KVM 的开发者，选择了 QEMU 这么一个成熟的虚拟软件来虚拟 I/O 设备[1]。并且对其进行了一定的修改，最后形成了 QEMU-KVM 这种体系，整个 QEMU-KVM 的体系结构图 2.3 所示。

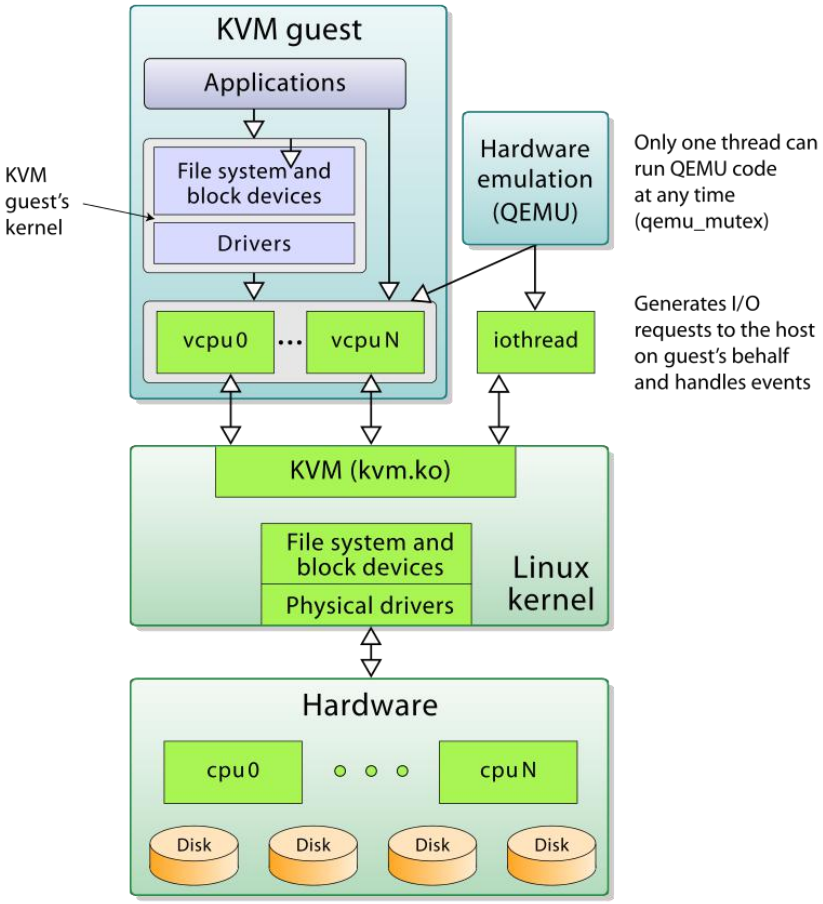

图 2.3 QEMU-KVM 体系结构

在该体系结构中，QEMU 运行在用户空间，而 KVM 则运行在内核空间，QEMU 在用户空间创建和管理各种虚拟硬件（例如虚拟硬盘等），而 CPU 指令和内存的虚拟化则通过/ioctl 调用/dev/kvm，从而实现了将这一部分交给了 KVM 来做。

2.4 硬件 Passthrough 技术

在虚拟化技术中，经常会遇到虚拟机需要使用物理硬件的情况，例如硬盘、网卡、显卡等外部设备，因此硬件 Passthrough 技术十分重要。下面将介绍 2 种主流的

硬件 Passthrough 技术。

2.4.1 SR-IOV 技术

SR-IOV 是 intel 提供的一种基于硬件 PCI 设备直通技术，其主要适用于网卡，该标准允许虚拟机之间高效共享 PCI-E 设备，由于是通过硬件实现的，因此效率很高，性能几乎可以媲美本机直接连接 PCI-E 设备[14]。

单个 I/O 设备可以被多个虚拟机共享。被共享的设备将会提供专用的资源给虚拟机。这样，每一个虚拟机都可以访问唯一的资源。因此，启用了 SR-IOV 并且具有适当的硬件和操作系统支持的 PCI-E 设备（例如以太网端口）可以显示为多个单独的物理设备，每个都具有自己的 PCI-E 配置空间。SR-IOV 技术的结构如下图 2.4 所示。

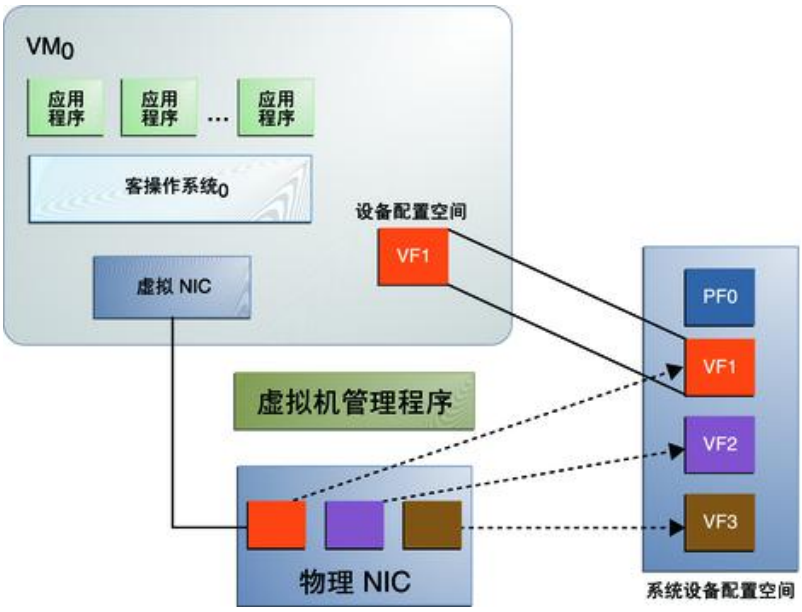

图 2.4 SR-IOV 技术的体系结构

使用 SR-IOV 技术有如下优点：

（1）由于是通过硬件实现，而且虚拟机环境可直接访问 PCI-E 硬件，其性能比虚拟设备来说，提高了不少

（2）由于该技术支持多个虚拟机共享一块设备，且可以有较高的性能，因此物理机不需要为了多个虚拟机提供等数量的 PCI-E 设备或者接口（比如网卡，在某些高性能网络需求下面，需要根据虚拟机数量配备足量的网口的网卡以保证网络性能，

而现在则不需要那么多，只需要一半甚至更少即可达到之前的网络性能），直接降低了企业的硬件成本和电能消耗，也间接减少了布线复杂程度。

2.4.2 VFIO+IOMMU 技术

VFIO 是 KVM 提供的一套用户态驱动框架，其提供两种服务：向用户态提供设备访问接口，向用户态提供配置 IOMMU 接口[15]。

VFIO 可以实现高效的用户态驱动。在虚拟化场景可用于设备的直通，用户可以通过用户态配置 IOMMU 接口，将 DMA 地址空间映射在进程限制的虚拟空间中。对于一些需要直接操作硬件的虚拟机来说非常重要，与 SR-IOV 相比，该技术则仅支持虚拟机排他式的独占，即当一个虚拟机占用设备时，其他虚拟机无法使用。

IOMMU 是针对 I/O 设备的内存管理单元，其作用是连接 DMA 的 I/O 总线和内存，完成从设备虚拟地址到物理地址的映射。以及提供对故障设备的内存保护的功能。如图 2.5 所示。

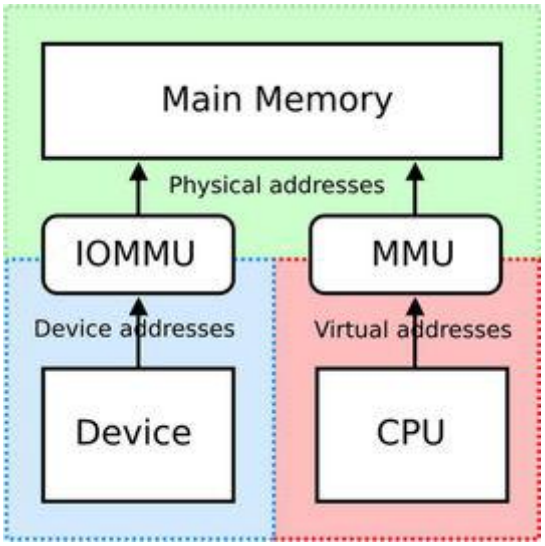

图 2.5 IOMMU 的逻辑结构，以及和 MMU 的异同点

IOMMU 技术有如下优点：

（1）由于 IOMMU 的映射，可以将多个不连续的物理地址映射为大块连续的地址提供给设备使用，便于简化驱动的设计

（2）使得老旧设备（主要是 32 位下的设备）可以使用高位地址，这样可以改善内存的使用效率，间接提高性能

（3）MMU 的提出就是为了保护物理内存地址，避免出现地址越界的问题，而 IOMMU 也同样有这个效果，将设备使用的内存地址保护起来，避免设备使用了不属于他的内存地址。

（4）提供硬件中断 remapping 功能

同时，也有一些缺点：

（1）和 MMU 一样，IOMMU 也是需要地址转换，相比较直接读取物理内存的方式，性能会有所降低

（2）消耗的物理内存容量比直接读取物理内存来说较高。

2.5 本章小结

本章将论文所需的技术进行了一些介绍。首先介绍了虚拟化技术的含义及业内使用情况，其次对 ARM 下提供对虚拟化技术的一些硬件支持进行了介绍，并分析其原理，最后将需要使用到的操作系统层之上的虚拟化软件以及硬件直通技术进行了介绍。为后文的开发和实验提供了技术基础。

3 虚拟化总体结构设计

3.1 实验环境的选择

首先需要针对后续章节的实现及测试，选择所需的开发平台。

由于虚拟化对于硬件的要求较高，因此我们需要选择一款高性能的 ARM 实验平台来完成后续的各种开发。

瑞芯微公司推出了一款高性能的 SOC——RK3399，该 SOC 方案的部分详细参数如表 3.1 所示。

表 3.1 RK3399 的部分技术参数

CPU	双 Cortex-A72+四 Cortex-A53 大小核 CPU 结构，频率最高 1.8Ghz
GPU	Mali-T864，支持 OpenGL ES1.1/2.0/3.0/3.1, OpenVG1.1, OpenCL, DX11
内存控制器	双通道 DDR3-1866/DDR3L-1866/LPDDR3-1866/LPDDR4
存储支持	支持 eMMC 5.1，SDIO3.0
显示支持	HDMI 2.0a 支持 4K 60Hz 显示，支持 HDCP 1.4/2.2
总线支持	USB 3.0、PCI-e 2.1 等

可以看出这款 SOC 方案的性能和其他设备提供的环境，对于后续的实验、开发而言是足够的，而且从价格来看也是相对较低，具有性价比高的优势。

搭载该 SOC 的开发板设备很多，为了方便实验测试，因此选择了天启科技提供的 Firefly-RK3399 开发板，该开发板搭载 RK3399 SOC，RAM 4GB，ROM 16GB，并且提供了丰富的硬件接口，同时也支持 linux 操作系统，因此可以作为开发实验用测试平台。

3.2 虚拟化技术的选择和结构设计

确定了实验平台后，就需要对整个虚拟化的架构进行设计。

在第二章介绍了很多虚拟化技术，而在后续的测试开发中，需要具体选择某一种虚拟化方案来完成，在 linux 操作系统下可以选用的虚拟化有 VMware、Xen 和

KVM，但由于 VMware 是基于 X86 的，因此不予考虑，而 Xen 虽然支持 ARM 下的虚拟化，但是需要对内核驱动进行修改，且面对不同的机器有不同的实现方案和固件，无法做到通用化，最重要的是，Xen 并未对目前的实验平台做任何适配，因此 Xen 的虚拟化方案也是不能考虑的，而 KVM 是纳入到 linux 内核下的，可以说只要硬件平台支持 linux 内核，就可以支持 KVM，因此，后续的开发测试全部基于 KVM 来实现。

确定了虚拟化技术后，需要对操作系统层之上的虚拟机软件进行选择，再上一张介绍过，KVM 仅仅实现了 CPU 和内存部分的虚拟化，而外设部分是通过 QEMU 这个开源软件实现的，因此这里选择 QEMU 作为操作系统层之上的虚拟机软件来帮助实现后续的优化和测试。

综上所述，对于整个虚拟化的架构设计，如图 3.1 所示。

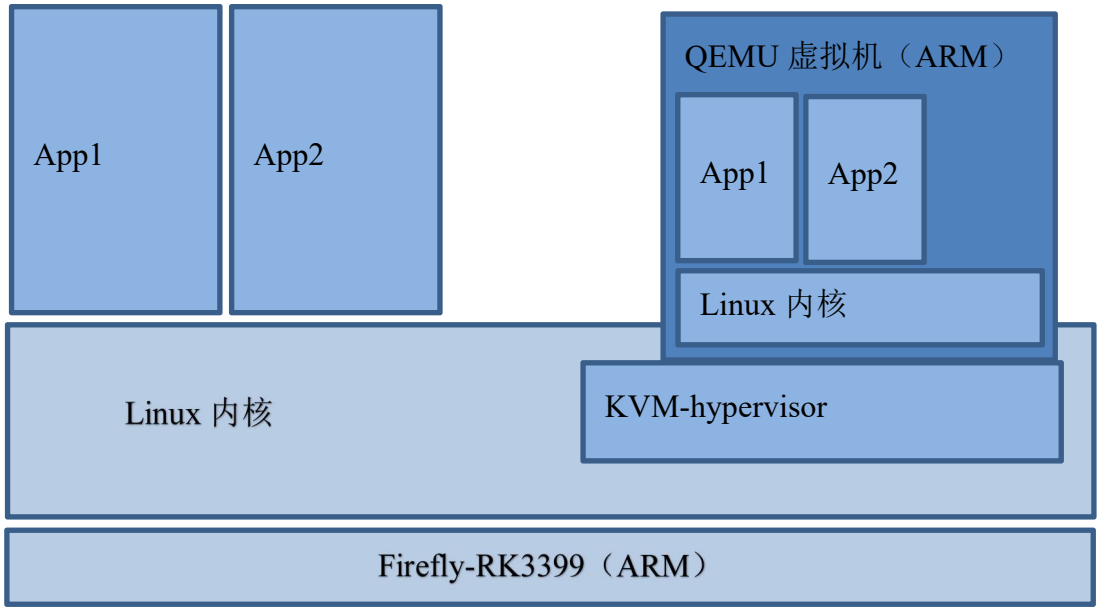

图 3.1 RK3399 上的虚拟化技术的整体架构设计

在 RK3399 平台上，运行 linux 内核的操作系统，其中内核支持 KVM 虚拟化技术，在操作系统层之上运行 QEMU-ARM 虚拟机，并且启用 KVM 作为 CPU 和内存的虚拟化从而达到加速 QEMU-ARM 虚拟机的效果。

3.3 需要实现的目标

上一节是针对整个虚拟化技术的选择和总体结构的架设，在该结构中，将实现以下目标，这些将在后面的章节进行具体实现：

（1）确定所要虚拟的目标机器型号，并且将 UEFI 修改移植到该机器中并正常运行，作为启动虚拟机操作系统的引导程序，为后面章节的优化和测试提供基础。

（2）优化现有的 ARM-KVM 代码，使其能够更好的支持 RK3399SOC 方案中的 CPU，最终达到优化虚拟机性能的效果，并且通过在虚拟机操作系统之上一系列测试，来证明这种优化是明显有效的。

（3）在 RK3399 提供的硬件支持以及 QEMU 提供的虚拟设备上进行修改，实现 RK3399 开发平台双屏异显，即一个屏幕显示虚拟机界面，而另一个屏幕显示物理机界面的效果。

（4）在现有的 vfio 技术上，实现在 RK3399 开发平台上的虚拟机硬件直通功能，使其可以直接访问物理外设。

3.4 本章小结

本章是对整个虚拟化架构技术选择以及总体设计，以及确定需要在该结构之上所实现的几个目标，例如 UEFI 的移植，KVM 的优化、双屏异显的实现以及虚拟机硬件的直通访问，这些功能将在后面的几章详细描述实现过程。

4 UEFI 的移植

4.1 移植 UEFI 的必要性

最早提出 UEFI 的是 INTEL 公司，起初是为了替代古老的 BIOS 启动方式[16]，而 ARM 的启动主要通过 BootLoader 或者 uboot 等方式进行启动，因为这种启动方式相对于 intel 的 bios 来说，启动速度快且功耗低，适合于工业控制领域那种实时性要求很高的场景，然而自从 ARM 开始在 PC 和服务器领域发力后，一些机构和个人也在研究如何在 ARM 的硬件系统上运行 UEFI，以方便未来 ARM 服务器的标准化实现，并且运行现有的服务器操作系统，例如 fedora server、windows server 等。

而实验中我们选择的测试操作系统为 fedora server，该操作系统需要通过 UEFI 启动，因此必须将 UEFI 进行移植和交叉编译为虚拟机可启动的 UEFI 固件，来引导 fedora server 的启动。

4.2 UEFI 移植的难点

移植 UEFI 固件的难点如下：

（1）UEFI 是 intel 公司提出的，虽然 intel 公司在 github 上发布了 UEFI 的源码，但是 intel 公司主要是给自家的 x86 平台提供的启动解决方案，因此 UEFI 想移植到 ARM 之上需要通过移植实现。

（2）x86 平台基本上在上世纪就已经统一标准，所以基本上 x86 平台的机器的驱动几乎是通用的，只需要 intel 公司对于新出厂的 CPU 和芯片组进行一些接口定义，而 ARM 平台的机器，基本上一套方案就是一个结构，生产厂商山头林立，相互之间并不兼容，仅仅是指令集使用同一方案，所以，首先需要确认移植的目标机器，再根据该机器的移植部分驱动使得其能够正常运行[17]，例如显示驱动等。

4.3 UEFI 移植的实现

首先，要确定需要移植的目标机器，ARM 机器与 x86 的机器不同，往往是一个

机器一个结构（哪怕是使用同样的 CPU 甚至同一种 SOC 都有可能不同），因此在移植前必须选择要移植哪款机器。

最新版本的 QEMU 针对 ARM 虚拟机提供了 20 多种比较通用的硬件平台，而且随着版本更新，其可支持的硬件数量在不断增加当中，其部分硬件平台列表如表 4.1 所示。

表 4.1 QEMU-ARM 下支持的设备列表及信息

硬件名	硬件介绍	处理器
akita	夏普公司开发的 PDA 设备	Intel PXA
ast2500-evb	Aspeed AST2500 EVB	ARM11
borzoi	夏普公司开发的 PDA 设备	OMAP310
canon-a1100	Canon PowerShot A1100 IS	（未知）
cheetah	Palm 公司开发的 PDA 设备	OMAP310
collie	夏普公司开发的 PDA 设备	SA-1110
connex	英国 Essex 大学设计的一款单板计算机	PXA255
cubieboard	珠海的 Cubietech 团队设计的 IOT 开发板	全志 A10
highbank	Calxeda 公司推出的一款 ARM 服务器	ECX-1000
imx25-pdk	飞思卡尔半导体推出的一款开发板	ARM9
integratorcp	飞思卡尔半导体推出的一款开发板	ARM9
kzm	一款基于飞思卡尔 SOC 的开发板	ARM11
lm3s6965evb	德州仪器公司推出的一款单片机开发板	Cortex-M
lm3s811evb	德州仪器公司推出的一款单片机开发板	Cortex-M
mainstone	（详细信息未知）	PXA27x
midway	Calxeda 公司推出的一款 ARM 服务器	ECX-2000
musicpal	一款 FREEDOM 公司推出的网络收音机	ARM9
n800	诺基亚 n800 手机	OMAP2420
netduino2	Netduino 开发板，第二代产品	（未知）
none	无总线下的 QEMU 虚拟平台，类似 virt	（可选择）
nuri	三星公司推出的 nuri 开发板	Exynos4210
palmetto-bmc	OpenPOWER Palmetto BMC	ARM9

raspi2	树莓派基金会推出的的树莓派 2 开发板	BCM2735
realview-eb	ARM 公司推出的虚拟开发板平台	ARM9
realview-eb-mpcore	ARM 公司推出的虚拟开发板平台	ARM11（多核）
realview-pb-a8	ARM 公司推出的 cortex-a8 虚拟开发板	Cortex-a8
vexpress-a15	ARM 公司推出的 cortex-a15 评估开发板	Cortex-a15
vexpress-a9	ARM 公司推出的 cortex-a9 评估开发板	Cortex-a9
virt	QEMU 自创的虚拟机平台	（可选择）
xilinx-zynq-a9	Xilinx 公司生产的 cortex-a9 开发板	Cortex-a9
z2	Zipit 公司推出的一款 IOT 设备	PXA27x

起初使用的是一个名为 vexpress-a9 的虚拟机进行测试，但是发现该虚拟机的硬件结构不适合使用 UEFI，而更适合使用 BootLoader 启动，且无法支持 KVM 加速，因此必须更换虚拟的平台，最后参考 QEMU 官方文档后，采用了 virt 虚拟机来实现

Virt 虚拟机在官方的资料非常少，经过搜索国内外相关文献资料后发现，这个虚拟机的架构大致情况如下：

（1）支持 PCI 设备的接入，例如磁盘、显卡等

（2）支持 USB 总线接入

（3）最大支持内存 65536MB

（4）可以指定部分 CPU 型号，并且支持 KVM 加速

根据这些资料，就可以将 UEFI 移植到该虚拟机上，并且尝试在此机器上启动 fedora server 并测试其性能。

选择目标机器后，将 UEFI 进行编译，事实上，ARM 公司和其他嵌入式设备公司合资成立的开源公司 linaro 已经将 UEFI 移植到了 ARM 平台下，所以可以从 github 上下载 linaro 提供的 tianocore 代码下载到 PC 机上，同时，由于在 linaro 官方提供的 efi 中未定义 QEMU 的显示设备 virtio-gpu-pci 设备的驱动信息，但定义了 virtio 以及 PCI-e 的驱动，因此需要对 UEFI 源码进行一定的修改，若不修改，则会在虚拟机启动时显示黑屏（通过 stdio 输出显示为设备同步失败），修改的文件为 /edk2/OvmfPkg/QemuVideoDxe/QemuVideoDxe/qemu.h，在里面添加 virtio-gpu 的设备定义，使得 UEFI 可以识别 virtio-gpu，如图 4.1、4.2 所示。

```
/*Add virtio-GPU type*/
EFI_STATUS
QemuVideoVirtioGPUModeSetup (
  QEMU_VIDEO_PRIVATE_DATA  *Private
  BOOLEAN                  IsQxl
  );
```

图 4.1 添加 Virtio-GPU 模式的配置

```
extern UINT8                                Crtc_1024_768_256_60[];
extern UINT16                               Seq_1024_768_256_60[];
extern QEMU_VIDEO_CIRRUS_MODES              QemuVideoCirrusModes[];
extern QEMU_VIDEO_BOCHS_MODES               QemuVideoBochsModes[];
extern QEMU_VIDEO_VIRTIOGPU_MODES           QemuVideoVirtioGPUModes[]
extern EFI_DRIVER_BINDING_PROTOCOL          gQemuVideoDriverBinding;
extern EFI_COMPONENT_NAME_PROTOCOL          gQemuVideoComponentName;
extern EFI_COMPONENT_NAME2_PROTOCOL         gQemuVideoComponentName2;
extern EFI_DRIVER_SUPPORTED_EFI_VERSION_PROTOCOL  gQemuVideoDriverSupportedEfiVersion;
```

图 4.2 添加 Virtio-GPU 模式的配置

然后，在 PC 平台上经过交叉编译，生成 UEFI 启动文件“qemu_efi.fd”[18]，并且在 PC 机启动 qemu-system-aarch64 测试是否能成功运行，效果如图 4.3 所示：

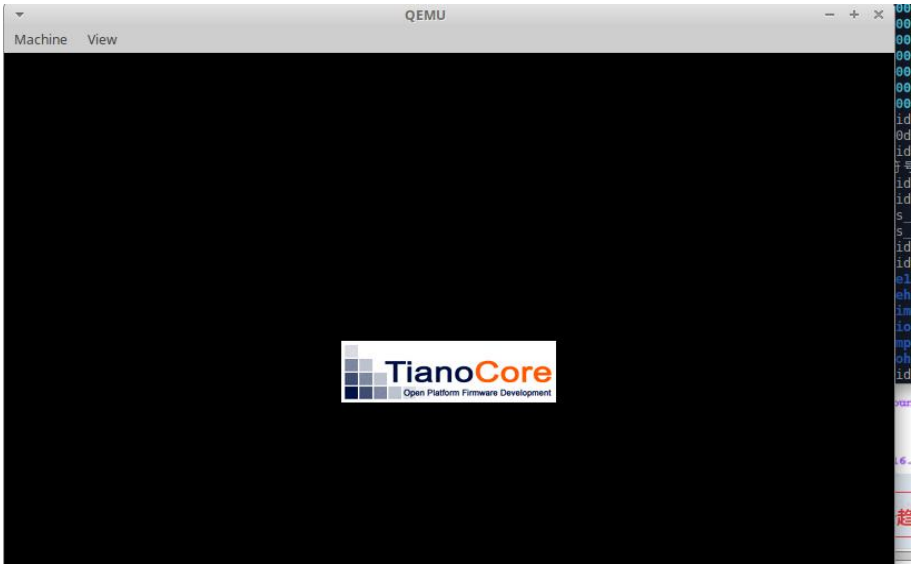

图 4.3 在 X86 上通过 QEMU-ARM 启动 UEFI 的效果

在 QEMU 中出现 UEFI 启动界面，说明 EFI 文件的编译是成功的，由于在 X86 的 PC 机器上无法进行 fedora server 的安装（启动速度太慢，下章有测试数据证明），因此该效果仅仅测试了 qemu 运行 UEFI 的效果，并不能作为实际测试，若需实际测

试性能，需要在测试硬件平台进行各指标性能对比（启动 KVM 和不启动 KVM 的情况下），与之相关的性能对比将在下一章进行。

4.4 本章小结

本章首先分析了针对 ARM 虚拟机移植 UEFI 的必要性，然后选择需要移植的目标机器，并分析该机器的特点，最后将 UEFI 部分代码进行少量修改并交叉编译成固件，在 QEMU 上的模拟目标机器并尝试启动 UEFI，来证明移植成功，为后续章节的开发和测试奠定了基础。

5 ARM 下的 KVM 加速优化及测试对比

5.1 ARM 下 KVM 的运行原理

Linux 内核在 3.x 版本后，加入了对于 ARM 下 KVM 的支持，起初是在 32 位下的软实现，后来 ARM 公司推出 V8 指令集之后，KVM 项目组利用 V8 指令集对原来的软实现进行了优化。

ARM 提供了很多新技术提供 KVM 等虚拟化方案使用，部分技术如下：

（1）hyp mode:hyp mode 是运行在 non-secure world 的最高特权级模式，它负责管理虚拟机操作系统，hypervisor 运行在这个新的模式里。这个模式将 hypervisor 和运行中的虚拟机操作系统分开，虚拟机操作系统运行在 non-secure 的 kernel mode。

（2）两级页表转换：由 hypervisor 负责把所有的虚拟机内存地址转换成实际物理地址，其实就是从物理上支持两级页表转换，而不需要使用影子页表。

（3）中断控制：ARM 创建了一个新的硬件模块，虚拟 CPU 接口，当中断到来时，所有的中断都首先被送到 hypervisor 里进行处理，由 hypervisor 通过虚拟 CPU 接口，发送给当前正在执行的虚拟机操作系统。虚拟中断也可以映射到物理中断，这样虚拟机操作系统就可以直接操作物理中断而不需要通过 hypervisor。

（4）仿真支持：当有中断进入 hypervisor 时，硬件向 hypervisor 提供一些额外的信息，这样 hypervisor 就免于取指令然后解码指令的硬件开销。因为对外设的模拟需要采用其他技术，削减这项技术的开销可以有效的提升性能。

（5）中断配置：并非虚拟机内所有在物理机上进行的中断请求，都需要中断进 hypervisor 进行处理，可以配置某些指令或操作是否需要中断，这样可以减少不必要的中断，从而减少开销，提升性能。

当 ARM 下的 KVM 运行时，QEMU 虚拟机会将物理机的 CPU 直接对接到虚拟机上，在虚拟机中查询 CPUINFO 会发现虚拟机的 CPU 不再是指定虚拟某个型号的 CPU，而是随物理机的型号而定。

5.2 针对 Cortex-A72 的 KVM 优化

Linux 内核 ARM 下的 KVM 优化中，对于某些处理器型号做了特殊的优化。

但由于开发平台的 SOC 中，存在两个 Cortex-A72 核心，而官方提供的 linux 内核版本为 4.4，其 KVM 虚拟化源码中，并未提供对于 Cortex-A72 的专属支持（注：截止本文完成时，KVM 项目组仍然未添加对于 Cortex-A72 的支持），而如果不在 KVM 源码中加入对于 Cortex-A72 的专属支持，那么 KVM 将会把其最为通用 V8 指令集 CPU 看待，不会有特别的性能优化。

首先我们需要在 linux 内核中增加对于 Cortex-A72 的定义，打开内核源码目录下的 arch/arm64/include/asm/cputype.h 文件，该文件对于 ARM 下每个型号的 CPU 的 partID 都进行了预定义，而 partID 相当于 CPU 的“识别码”，来区分不同的 CPU 型号，打开后找到每种 CPU 型号的 partID 定义代码段，其中仅定义了 A53 和 A57 的 partID，并未定于 A72 的 partID，根据 ARM 官方对于 A72 的定义，其 partID 为 0XD08，将该段预定义添加入代码中。

将该 CPU 在内核中定义后，需要修改 KVM 源码增加对虚拟机使用 Cortex-A72 的支持，arch/arm64/kvm/guest.c 文件中的 kvm_target_cpu()函数对虚拟机的使用 CPU 型号做了定义，代码如图 5.1 所示。

```
switch (implementor) {
case ARM_CPU_IMP_ARM:
        switch (part_number) {
        case ARM_CPU_PART_AEM_V8:
                return KVM_ARM_TARGET_AEM_V8;
        case ARM_CPU_PART_FOUNDATION:
                return KVM_ARM_TARGET_FOUNDATION_V8;
        case ARM_CPU_PART_CORTEX_A7:
                return KVM_ARM_TARGET_CORTEX_A7;
        case ARM_CPU_PART_CORTEX_A15:
                return KVM_ARM_TARGET_CORTEX_A15;
        case ARM_CPU_PART_CORTEX_A53:
                return KVM_ARM_TARGET_CORTEX_A53;
        case ARM_CPU_PART_CORTEX_A57:
                return KVM_ARM_TARGET_CORTEX_A57;
        };
        break;
```

图 5.1 kvm_target_cpu 函数对于不同 CPU 型号的处理方式

该段代码中对于不同的 partID 返回的 KVM_ARM_TARGET_CORTEX_XXX 的参数不同，需要对该预定义参数的含义进行分析，查找到该预定义参数在

arch/arm64/include/uapi/asm/kvm.h 中，该定义是对 KVM 所支持的 CPU 型号进行的预定义，当 CPU 属于 V8 而型号未知时，会被 KVM 认为是通用 V8 处理器而采用其他的方式运行 KVM 加速，然而在性能上是不如指定型号的，根据 ARM 公司对于 Cortex-A72 公开的技术信息表示，A72 是 A57 的性能增强版本，那么我们可以在该文件内将 A72 定义，通过“欺骗”KVM 的方式使其认为自己是运行在 A57 之上，从而使用 A57 的优化方式启用 KVM，代码修改如图 5.2 所示。

```
case ARM_CPU_IMP_ARM:
        switch (part_number) {
        case ARM_CPU_PART_AEM_V8:
                return KVM_ARM_TARGET_AEM_V8;
        case ARM_CPU_PART_FOUNDATION:
                return KVM_ARM_TARGET_FOUNDATION_V8;
        case ARM_CPU_PART_CORTEX_A7:
                return KVM_ARM_TARGET_CORTEX_A7;
        case ARM_CPU_PART_CORTEX_A15:
                return KVM_ARM_TARGET_CORTEX_A15;
        case ARM_CPU_PART_CORTEX_A53:
                return KVM_ARM_TARGET_CORTEX_A53;
        case ARM_CPU_PART_CORTEX_A57:
                return KVM_ARM_TARGET_CORTEX_A57;
        case ARM_CPU_PART_CORTEX_A72:
                return KVM_ARM_TARGET_CORTEX_A57;
        };
        break;
```

图 5.2 当客户机 CPU 为 A72 时，“欺骗”KVM 按照 A57 的方式运行

对代码进行修改后，就可以将该内核进行编译，从而在开发板中开启 KVM 加速功能。

5.3 KVM 加速功能的开启

由于瑞芯微官方提供的操作系统内核默认未开启 KVM，因此需要对开发平台的 KVM 加速进行调通，在官方网站下载内核源码到本地计算机，输入 make menuconfig 打开内核配置选项，选择 Virtualization，勾选 Kernel-based Virtual Machine(KVM) support，以及 Host Kernel accelerator 两个选项，如图 5.3 所示。

```
-*- Virtualization
[*]   Kernel-based Virtual Machine (KVM) support
<*>   Host kernel accelerator for virtio net
[*]   Cross-endian support for vhost
```

图 5.3 内核配置 KVM

选择完毕后，选择 save 保存，开始交叉编译，编译完成后会生成一个 kernel.img

的内核文件，将开发板连接到 PC 机，启动 RK 官方的烧写 flash 工具 AndroidRKtools，将该内核文件烧写进开发板中，如图 5.4 所示。

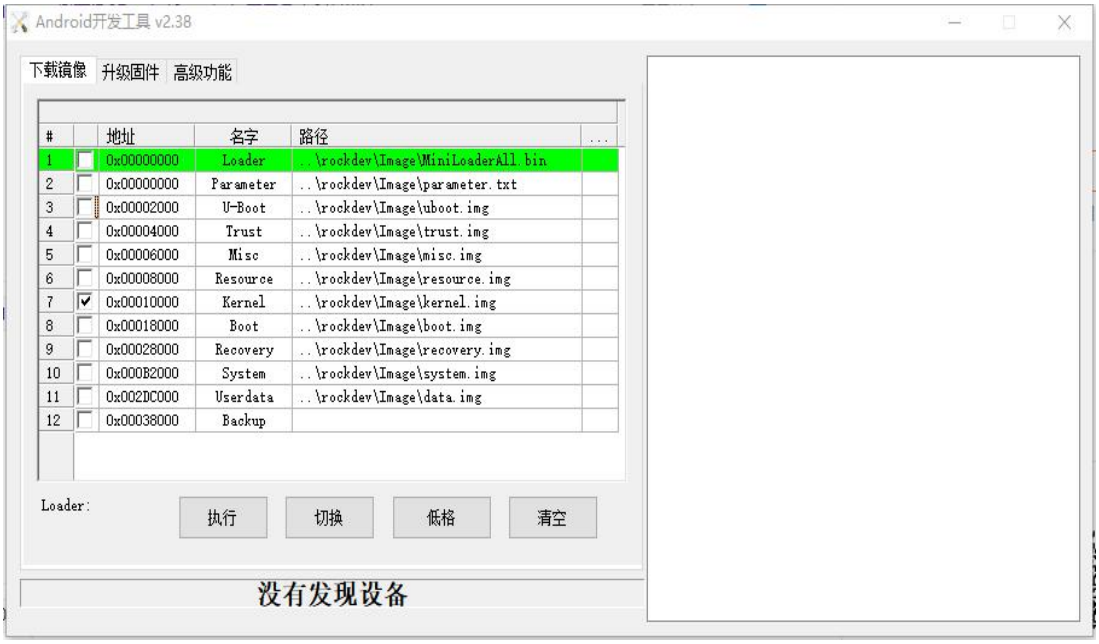

图 5.4 AndroidRKtools 烧写内核固件

烧录成功后，启动开发板系统，打开终端命令行，输入 kvm ok 显示 kvm 设备已装载，说明 KVM 加载成功。

5.4 性能测试对比

为了证明经过修改后的 KVM 源码对性能加速有明显提升，需要准备 2 个相同硬件参数的 ARM 开发板，两者的区别在于：一个开发板装载 KVM 模块经过代码修改的内核，一个装载未经过代码修改的内核，同时和未启用 KVM 以及在 X86 环境下模拟 ARM 进行对比。

可以通过安排 2 项测试来验证性能加速：

（1）安装 fedora server 的速度

（2）在 fedora 启动 unixbench 得到跑分结果

测试用 Qemu 所模拟的机器硬件配置如表 5.1 所示。

表 5.1 测试所用 QEMU 环境配置情况

	开启 KVM 情况下（经过性能优化）	开启KVM情况下（未经过性能优化）	未开启 KVM 情况下（ARM 开发板）	未开启 KVM 情况下（X86 PC 机）
CPU	Cortex-a72*2 1.8Ghz+Cortex-a53*4 1.4Ghz（随物理机 KVM 加速）		Cortex-a57*4 2.0Ghz（QEMU 软件模拟）	
内存	1GB			
硬盘空间	20GB			

首先，需要从官网下载 fedora server for aarch64 的版本到测试硬件平台上，输入 qemu 启动参数，开始安装 fedora server 到虚拟机中。

从启动安装程序的所需时间来看就可以很明显的发现性能区别，通过一张图表来表示，如图 5.5 所示。

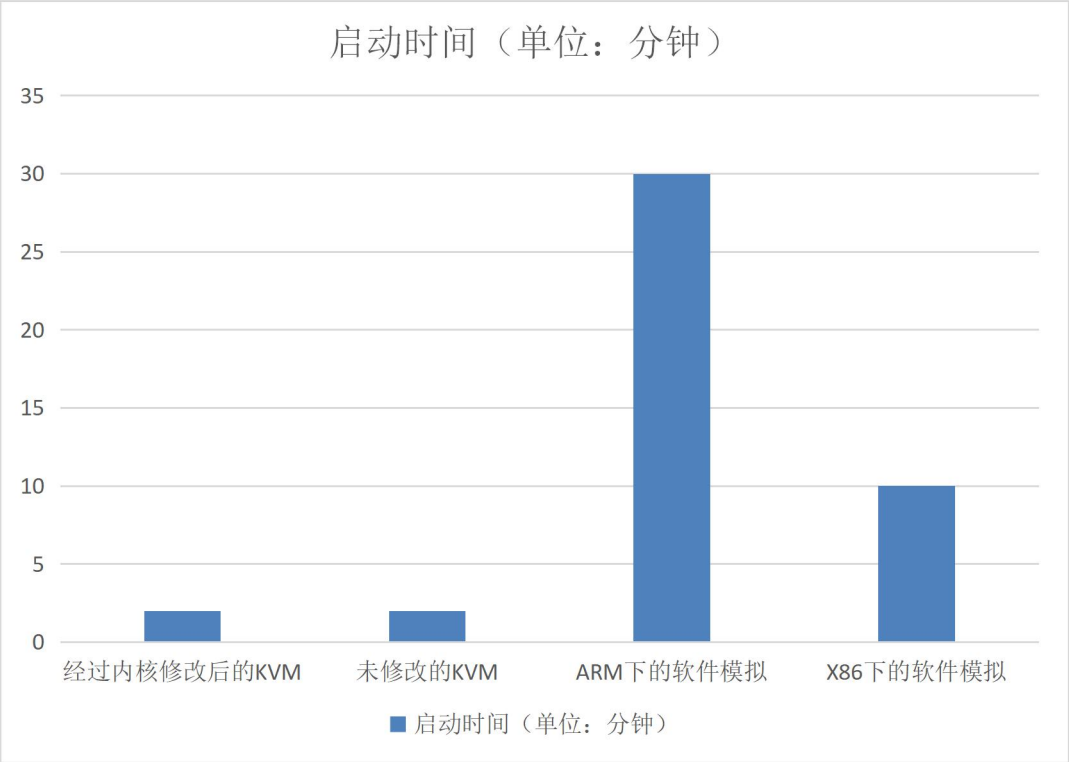

图 5.5 fedora server 安装程序在三种环境下的启动时间对比（越短越好）

从 UEFI 启动操作系统安装程序看，经过代码修改和未经过代码修改的 KVM 虚拟机启动操作系统安装程序差距并不大，然而和通过 QEMU 软件模拟以及 X86 下的

模拟相比，KVM 加速后的虚拟机，启动速度明显比软件模拟快了很多。而对于 KVM 代码修改前后的性能差距对比，需要通过其他方式来测试。

为了节省测试时间，因此直接在 KVM 加速的情况下完成操作系统的安装，安装过程与其他的操作系统几乎无任何区别，在安装后进入系统，下载 unixbench 软件进行测试。

Unixbench 是在 unix/linux 下的一个开源跑分测试软件[19]，一般用于测试服务器的性能指标，属于比较权威的跑分程序，将 unixbench 下载到虚拟机，编译后在不同的配置参数下运行，跑分结果如下图 5.6 所示。

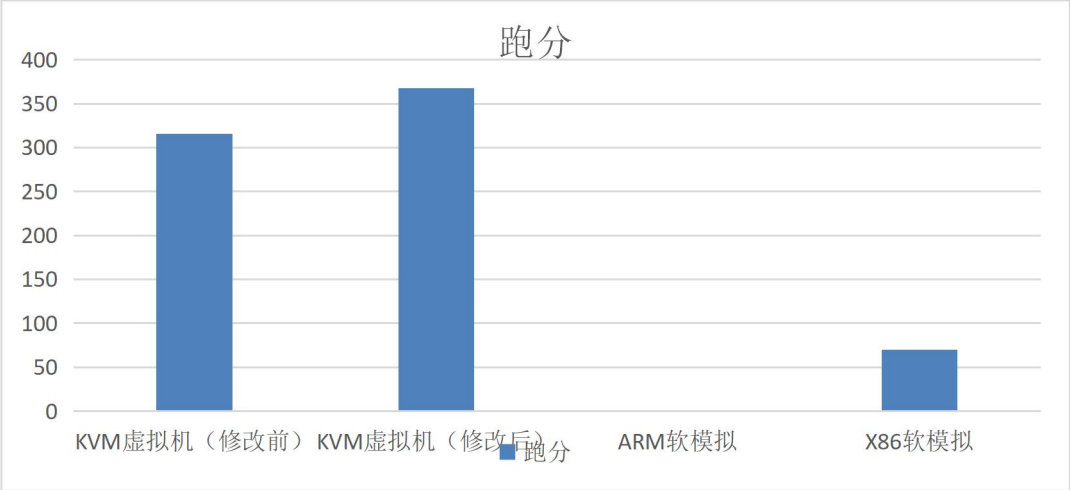

图 5.6 三种环境下运行 unixbench 的跑分结果

通过跑分结果发现，经过代码修改后的 KVM 虚拟机，其跑分结果为 367.8，而未经过代码修改的 KVM，跑分结果仅为 315.4，通过这一跑分数据结果也证明了，通过修改 KVM 代码的方式，可以使得虚拟机的性能有小幅提升。而未开启 KVM 虚拟化加速的虚拟机，跑分结果可以用“惨不忍睹”来形容，特别是在 ARM 下通过 QEMU 软模拟的虚拟机，由于启动操作系统的过程太慢，已经无法启动跑分程序，所以标记为 0，而在 X86 下的 QEMU 软模拟，也归功于 PC 平台的强大性能，勉强可以启动操作系统并跑分，但结果也是相当低，仅有 70.25。

为了验证 ARM 下 KVM 加速的效率，我们将 unixbench 跑分软件在 ARM 实验平台上进行跑分，得出的结果如图 5.7 所示。

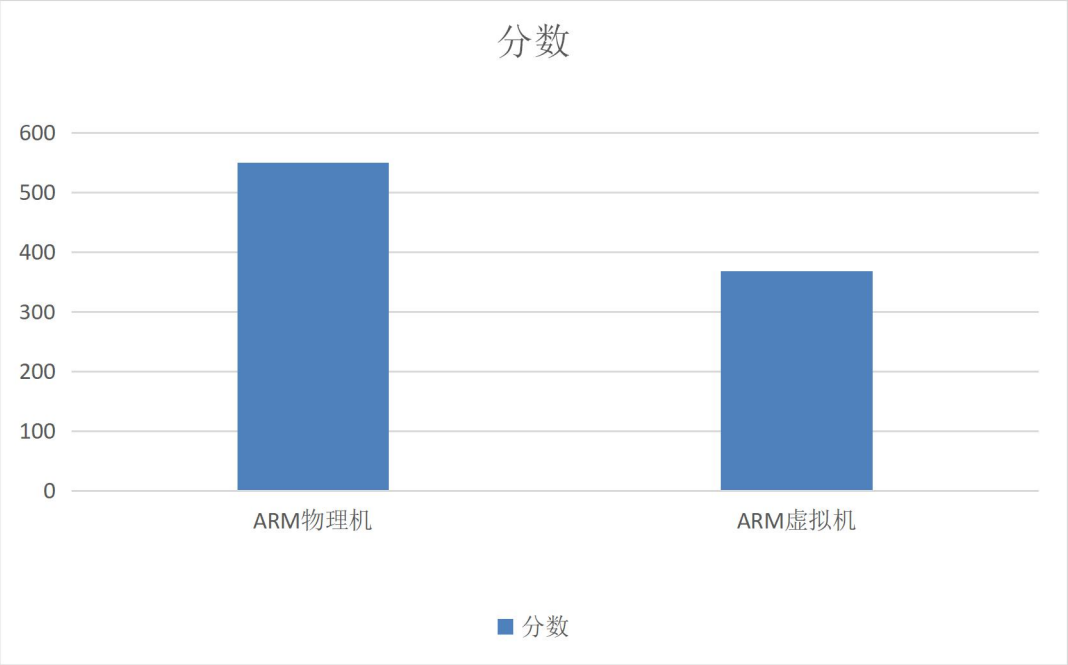

图 5.7 ARM 虚拟机和物理机跑分结果对比

在物理机下的跑分结果为 550.2，可以根据这个跑分结果与修改后的 KVM 虚拟机相比不难发现，在 ARM 下的 KVM 加速（即使是修改后的）会有一定的性能损耗，若将两者的跑分进行对比，ARM 下的 KVM 加速效率大约为 60%-80%左右。

当然，测试时也需要对比一下在 X86 下的 KVM 启动后和本机模拟的区别效果，来对比下 ARM 下 KVM 加速与 X86 下的 KVM 加速之间的效率差距，为了使得实验效果更接近真实结果，测试时对 X86 下的实验机器配置设置如表 5.2 所示。

表 5.2 X86 下物理机和虚拟机的配置参数对比

	物理机	虚拟机
CPU	Intel Xeon E5-2640 v2 2.0Ghz 八核心	
内存	8GB	2GB
硬盘空间	250GB	20GB

注意这里的“物理机”实际上还是一个虚拟机，只不过表右侧的“虚拟机”是运行在这个“物理机”之上的，虚拟化支持这种嵌套的方式，事实上，“虚拟机”不会受到“物理机”的宿主机的任何影响，所以，两台机器的跑分结果对比是有参考价值的。

最终两台机器的跑分结果图 5.8 所示。

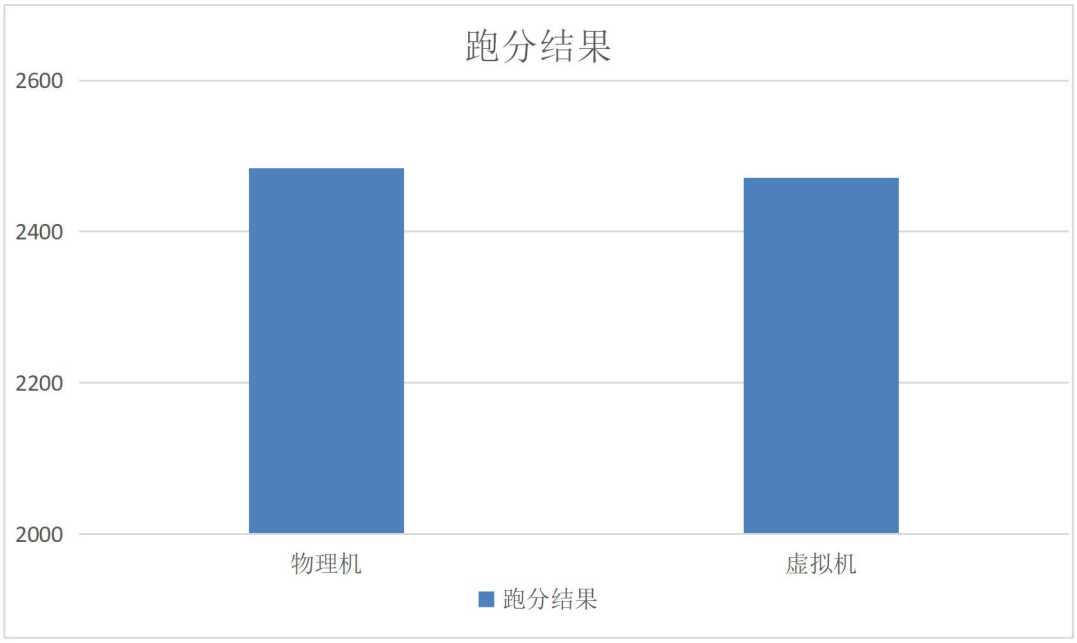

图 5.8 X86 下虚拟机和物理机 unixbench 跑分结果

首先由于 intel 的 Xeon E5-2603v2 处理器的性能远高于 RK3399，因此跑分结果几乎是 RK3399 的四倍，这一点没有什么比较值得评判的地方，然后在 X86 下的物理机和虚拟机的跑分对比结果非常惊人，物理机和虚拟机的跑分结果差距只有十几点的差距，反映出几乎没有性能的损耗的结果，效率几乎在 99%以上，说明 ARM 下的 KVM 性能还有待继续优化改善。

5.5 本章小结

本章首先介绍了 KVM 在 ARM 下的部分运行原理，以及 KVM 在 ARM 下运行的特点。然后针对 KVM 未支持实验平台所选用的 CPU 型号的特殊优化问题，通过修改相关代码，解决会被认为是通用 V8 处理器的而导致虚拟化效率不高的问题，最后通过各项测试来证明这种优化方式是有效的，并且与 X86 的 KVM 进行对比发现 ARM 下的 KVM 效率不及 X86 下的，仍有一定的优化空间。

6 虚拟机双屏异显的实现

在工业系统、车载系统上经常遇到需要双屏异显的情况，也就是一个屏幕显示操作系统界面或者仪表，另一个屏幕显示视频、终端等，本章将双屏异显应用到虚拟机领域，实现一个屏幕显示操作系统界面，一个屏幕显示虚拟机操作系统界面的效果。

6.1 双屏异显的硬件需求

6.1.1 双屏异显的原理

双屏异显其实利用的是 linux 下 framebuffer 设备来的实现[20]，由于在 linux 下，不可能像 DOS 那样直接调用硬件中断对屏幕进行直接写入操作，因此只能采用 framebuffer 方式实现。

Framebuffer 是 linux 在 2.6 以上版本提供的一个设备文件，是显示器和计算机之间的一个内存地址空间，系统将需要显示的内容写入该缓冲区，然后在显示器上面显示出来，这个操作可由 GPU 完成，但也可以由用户通过命令或者编写程序向该文件写入数据，但想做到双屏异显，还是需要多显卡支持，因为一块显卡就是一个显存空间，映射到 framebuffer 文件，对应的就是/dev/fb0、/dev/fb1 等文件[21]。

Framebuffer 的设备文件工作原理图 6.1 所示。

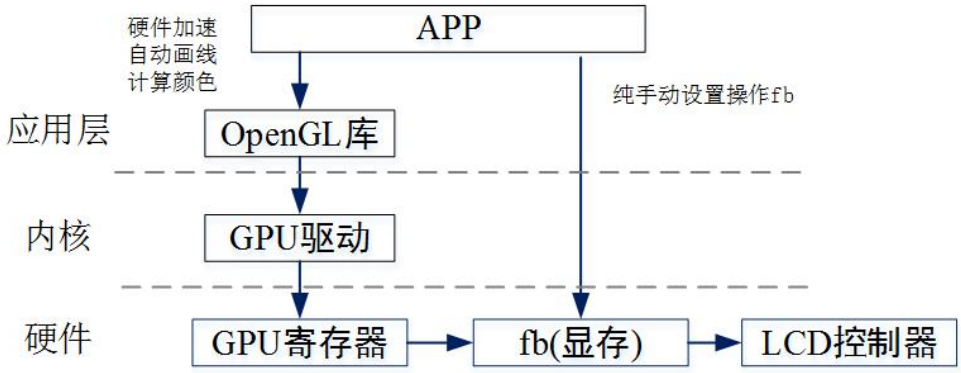

图 6.1 linux 下 framebuffer 设备的工作原理

按照上图的介绍，需要多个显卡硬件才能实现双屏异显功能，然而在嵌入式设备中，由于功耗，晶片面积等原因，不可能有多个显卡（显示适配器、图形处理器）来支持双屏异显功能。

不过，瑞芯微公司为了方便嵌入式厂商实现双屏异显功能[22]，通过修改内核驱动的方式，提供了一个双屏异显的硬件解决方案，具体的技术参数 RK 公司并未透露，但是目前知道的是，瑞芯微公司通过划出 2 块内存空间建立 2 个 framebuffer 文件来对每一块屏幕进行指定输出[23]，开发者只需要往指定文件写入数据即可做到双屏异显。

6.2 双屏异显的难点

在 ARM 开发板上实现双屏异显功能有以下的难点：

（1）由于 QEMU 项目组仅仅提供了自己的源代码，而没有提供这些代码的技术文档和工作原理，所以想实现双屏异显，必须查阅 QEMU 显示设备原代码，从中分析出显示设备的工作原理，并且将显示设备的关键数据输出，分析其数据的结构和含义，找到实现双屏异显所需要的数据。

（2）获取数据后，需要将数据转化为 linux 下的 framebuffer 文件可以读取的数据，并且要配合其数据结构，如何转换，也是一个问题。

6.3 双屏异显的实现

通过写入 framebuffer 的方式实现，首先需要从 QEMU 中，获取到当前屏幕的数据信息，但由于 QEMU 官方并未公布这些显示设备的详细文档信息，只有开源代码，因此想读取 QEMU 虚拟机的屏幕信息是一件非常困难的事情。只能通过分析设备文件的 C 语言代码来分析其具体工作原理，找到屏幕数据信息的具体数据结构，再将其分析读取，写入设备文件。

首先查看 QEMU 中，提供给 virt 虚拟机的的显示设备有多个，如表 6.1 所示。

表 6.1 QEMU 下 virt 虚拟机支持的显示设备

设备名	设备介绍
Bochs-display	（未知设备）
Ramfb	内存缓冲独立设备（Ram framebuffer standalone device）

VGA	传统 VGA 显示器设备
virtio-gpu-device	virtio 图形加速设备[24]，接入 virtio 总线[25]
virtio-gpu-pci	virtio 图形加速设备，接入 pci-e 总线

其中 ramfb 设备是 QEMU 3.0 版本新增的一个设备[26]，从字面意思上看，大致是划出一块内存空间作为缓存，将需要显示的内容放入缓存区，然后再输出，这一点和 linux 下的 framebuffer 文件原理有点类似。本章将对 ramfb 设备进行详细分析。

Ramfb 文件在 qemu 源码的 hw/display 目录下，名为 ramfb.c 文件，头文件 ramfb.h 在 include/hw/display 目录下，打开代码后发现，该设备有 2 个数据结构，名为 RAMFBCfg 和 RAMFBState，经过分析后各参数的类型、用途如表 6.2、6.3 所示。

表 6.2 RAMFBCfg 的数据结构

参数名	参数类型	参数用途（推测）
addr	uint64_t	Framebuffer 内存地址
fourcc	uint32_t	（未知）
Flags	uint32_t	设备标志编号
Width	uint32_t	设备分辨率（宽）
Height	uint32_t	设备分辨率（高）
Stride	uint32_t	（未知）

表 6.3 RAMFBState 的数据结构

参数名	参数类型	参数用途（推测）
*ds	DisplaySurface	显示对象指针
*framebuffer	uint8_t	显示缓冲区指针
width	uint32_t	设备标志编号
height	uint32_t	设备分辨率（高）
cfg	RAMFBCfg	ramfb 设备的参数设置

其中 RAMFBState 结构中的成员变量 framebuffer 疑似指向存储为显示数据内容的指针，但想获取其数据信息，还需要先知道其运行原理。

Ramfb.c 文件下有 ramfb_setup、ramfb_fw_cfg_write、ramfb_display_update 三个

函数，通过源码分别对这三个函数进行具体分析：

（1）ramfb_setup 函数是启动虚拟机时对设备的加载，首先获取虚拟机参数信息，判断模拟的虚拟机是否支持 DMA（直接存储器访问）功能，如果不支持则提示错误终止虚拟机的运行，若支持则加载设备固件，并且划分内存空间给设备使用，同时将 ramfb_fb_cfg_write 函数入口提供给虚拟机调用。

（2）ramfb_fb_cfg_write 函数的功能是在虚拟机显示参数发生变化的时候被调用，当虚拟机的显示分辨率或者色彩模式等发生变化时，获取虚拟机的分辨率、色彩信息，将 RAMFBState 内的信息更新，由于分辨率发生变化，内存空间需求也会发生变化，因此，该函数在执行时，会在物理机上重新划分内存缓冲区，同时将缓冲区地址发送给 qemu 的图形压缩函数 qemu_drm_format_to_pixman()，推测是用于在 spice 或者 vnc 远程传输时降低对带宽的需求，最后将分辨率等信息发送给本地监视窗口，窗口根据分辨率大小调整显示界面的长宽比。

（3）ramfb_display_update()函数在显示屏幕需要刷新的时候被调用，如果虚拟机需要更新屏幕，则会调用这个函数，调用时会读取设备状态，如果该设备已不存在（查看设备数据结构中的宽高信息是否为空）则会停止刷新直接结束，如果该设备仍然存在则将屏幕刷新一次，因此该函数几乎会被频繁调用。

为了更直观的表现这三个函数的功能，使用一张表来表示三个函数各自的功能，如表 6.4 所示。

表 6.4 ramfb 设备的三个函数的功能分析

函数名	调用情况	功能描述
ramfb_setup	启动虚拟机时	在虚拟机启动时，先判断 DMA，再将设备参数初始化
ramfb_fb_cfg_write	屏幕分辨率参数有变时	当虚拟机配置的屏幕分辨率、色彩发色数等参数发生变化时，重新划分内存缓冲空间
ramfb_display_update	虚拟机请求刷新屏幕时	当虚拟机需要刷新屏幕时，读取设备状态，将内存缓冲空间内容进行刷新

根据对这三个函数的代码分析，以及控制台输出测试，推断整理出了这三个函数的启动顺序以及猜测的设备运行逻辑流程，并使用流程图表示，如图 6.2 所示。

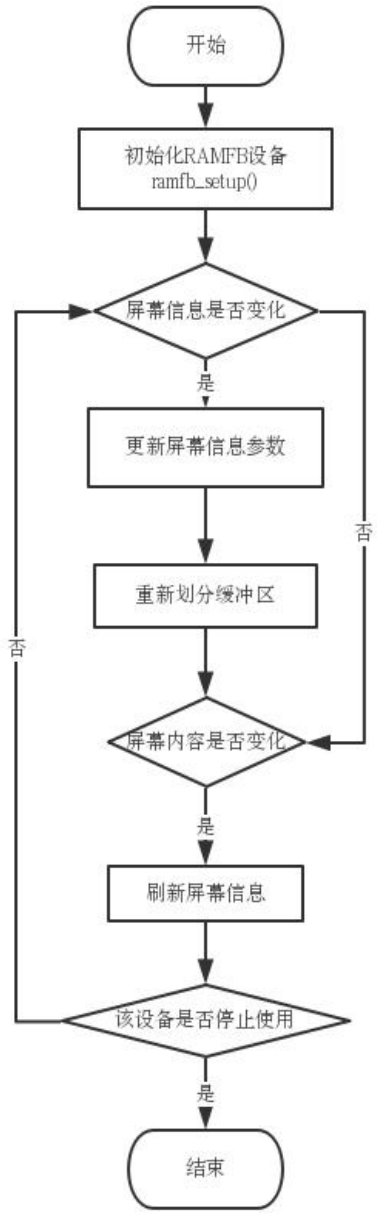

图 6.2 Ramfb 设备的运行流程图

根据上图的流程图来看，如果想获取图形数据，必须从 ramfb_display_update 函数入手，然而根据 RAMFBState 的数据结构，里面的 framebuffer 是类型为 uint8 的一个指针，并不知道该指针所指向数据内容是一个什么样的模式，因此在该函数内，添加一个控制台输出，由于指针是 uint8 类型的，因此每个数应该是一个 8 位无符号数（0-255），将该指针所指向的内存地址输出到控制台后，发现数据如图 6.3 所示。

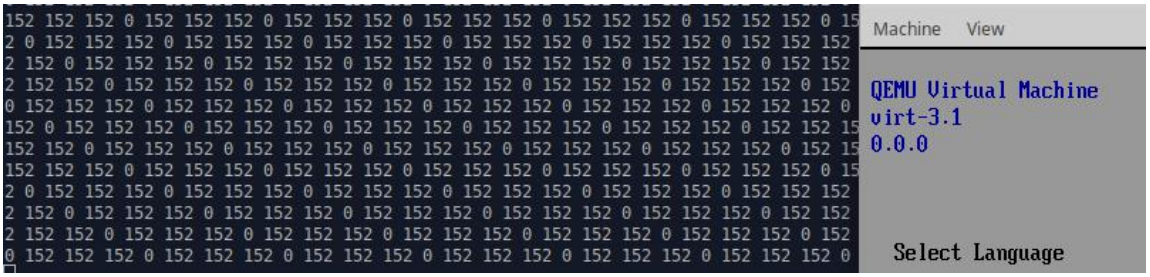

图 6.3 输出的数据结果与屏幕对比

由于 framebuffer 的内存区数据量较大，若要输出满会导致控制台卡顿，因此仅输出屏幕中的一行数据，可以看出，“152 152 152 0”疑似为一个 RGB+通道的数值，上图右侧的屏幕是一个灰色的屏幕，所以尝试将“152 152 152”三个数作为 RGB 值来对待，通过在线 RGB 工具将该数值输入后，发现输入该值输出的颜色与该行屏幕的颜色几乎一致，如图 6.4 所示。

图 6.4 数值转换为 RGB 值与虚拟机界面对比

可以看出，该数据极有可能就是 RGB 值，根据这个结果，就可以推断出该指针所指的内存空间的数据结构，是一个 RGBA 的数据，具体结构通过图形来描述更直观，如表 6.4 所示。

表 6.4 猜测的 framebuffer 指针所指向的数据结构

Red	Green	Blue	Alpha
红色分量	绿色分量	蓝色分量	通道分量

为了保证该猜测的准确性，尝试将该数据输出为一个 bmp 位图文件（相当于屏幕截图），结果发现屏幕输出与输出的位图文件完全一致，因此完全可以确信，RAMFBState 的成员变量 framebuffer 指针所指向的内存空间所存储的数据结构是一个 RGBA 的结构，即“红-绿-蓝-通道（透明度）”的一个结构，由于目前透明度参数在代码分析和双屏异显的实现中没有意义，因此不考虑该值的用途。

获取了该设备数据结构和提取数据的方法后，就要将该数据写入屏幕中，前面已经讲过 linux 下也有一个 framebuffer 的设备文件，但是通过查找文档得知，此设备的数据结构与该函数获取的数据结构并不完全兼容，因此需要对数据格式进行转换，才可以在屏幕中输出该数据。根据这个方案，将 ramfb 的设备运行流程做了更改，在 ramfb_display_update()刷新数据后，将 RAMFBState 的成员变量 framebuffer 指针所指的内存空间输出到 framebuffer 文件即可。

Linux 中 framebuffer 设备文件中的数据结构为 unsigned int 32[27]，而 ramfb 的数据结构为每个颜色由一个 uint8 表示，红绿蓝三色外加一个 alpha 通道（透明度）组成，所以要将这些颜色通过每 8 位左移，按位相加的方式来将 4 个 uint8 格式的数据转换为一个 uint32 位的数据[28]。

根据相关文档，得知这个 uint32 位的数据格式如表 6.5 所示。

表 6.5 linux 系统下的 framebuffer 文件数据结构

位数	0-7	8-15	16-23	24-31
定义	Alpha	Red（红色）	Blue（蓝色）	Green（绿色）

根据上图的结构，那么只用将颜色数据存放进对应的位数即可[29]，将该数据输出到 framebuffer，即可及时刷新屏幕，具体实现的部分代码如代码 6.1 所示。

代码 6.1 双屏异显实现部分关键代码

```
void hust510_write(uint8_t *framebuffer,uint32_t width,uint32_t height){
    int fd=-1;
    int ret=-1;
    unsigned int *pfb;
    struct fb_fix_screeninfo finfo;
        struct fb_var_screeninfo vinfo;
    fd=open("/dev/fb1",O_RDWR);
    if(fd<0){
        return -1;
    }
    ret=ioctl(fd,FBIOGET_FSCREENINFO,&finfo);
    if(ret<0){
        perror("ioctl");
        return -1;
    }
    ret=ioctl(fd,FBIOGET_VSCREENINFO,&vinfo);
```

```
    if(ret<0){
        perror("ioctl");
        return -1;
    }
    int screensize=vinfo.xres*vinfo.yres*vinfo.bits_per_pixel/8;
    pfb=(unsigned int
*)mmap(NULL,finfo.smem_len,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    if(NULL==pfb){
        perror("mmap");
        return -1;
    }
    for(int j=0;j<height;j++){
        for(int i=0;i<width*4;i+=4){
            //Todo：创建临时 UINT32 变量，将 RGBA 四个 8 位分量分别写入变量中，再写入
文件缓冲区
        }
    }
    close(fd);
}
```

该实现代码是一个函数，在 ramfb_display_update 执行时调用，将 framebuffer 指针和当前屏幕参数的宽和高信息传入，打开 framebuffer 文件，若无法打开则返回失败退出 QEMU，若成功则根据屏幕分辨率的宽和高，创建一个对 framebuffer 文件写入的缓冲区，缓冲区大小就是宽和高的乘积，例如 800*600，再将 framebuffer 指针所指向的数据进行读取。创建一个临时的 32 位长度的无符号整数变量，将 RGBA 数据每 8 位填入该变量对应的 RGBA 定义位中，再将该变量填入文件缓冲区，当所有信息全部填入后，再全部写入 framebuffer 文件中即可，最后关闭文件，完成本次屏幕刷新。

6.4 本章小结

本章首先从技术层面上对 framebuffer 设备文件进行分析，并分析了想实现双屏所需的硬件要求，以及介绍了开发平台提供的硬件驱动支持以及如何在开发板上开启双屏异显的支持。

其次，对如何从 QEMU 中读取设备图形信息进行了研究分析，由于 QEMU 项目组并未提供详细代码文档，因此要从设备实现代码中发现其运行原理并且找到其数据结构的意义，然后写测试代码将数据输出到控制台，分析输出的数据有何意义，最后确定显示设备中的 framebuffer 数据结构。

最后将分析的数据试图转换为 linux 的 framebuffer 设备文件的格式，并且将数据输出到该设备，达到双屏异显的效果。

7 虚拟机硬件直通的实现

在第二章介绍了虚拟化中的一个重要技术：硬件直通技术，该技术目前广泛应用于 X86 服务器下，可以帮助企业解决很多现实问题[30]，然而对于在 ARM 下的硬件直通技术，尚处于研究阶段，本章将通过 ARM 开发板来测试 ARM 下平台的硬件直通是否可行。

7.1 实现方案及难点

和 X86 平台相比，ARM 平台没有 intel 公司提供的 VT 虚拟化技术[31]，因此如果需要对硬件设备实现直通，则需要通过不同的方式来实现。

查询 RK3399 的官方参数后得知，RK3399 内部支持 PCI-E 总线，支持的标准版本为 2.1，因此在理论上，完全可以通过该设备调通 PCI-E 设备并且直通到虚拟机上，由于 ARM 板没有 vt-d 技术，所以尝试使用 vfio 方式来实现。

由于实验设备平台并未提供类似 PC 主机上的 PCI-E 4x 接口，但提供了一个 M.2 的 NGFF 接口，该接口实际上也是接到 PCI-E 总线上的，但是需要独立供电接口（PCI-E 设备的功耗通常较高），因此需要通过转接板来转接标准 PCI-E 接口[32]，并且单独接入 PC 电源（PC 电源需要短接 14 号引脚[33]），再接入 PCI-E 设备，设计方案如图 7.1 所示。

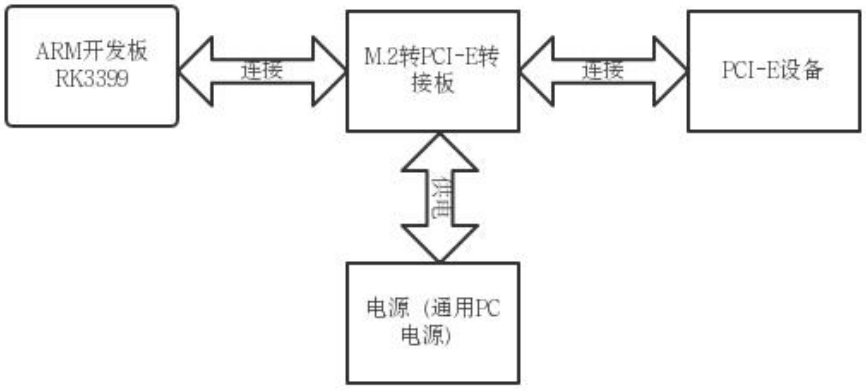

图 7.1 Firefly-RK3399 接入 PCI-E 设备的方案原理图

7.2 具体实现

想实现硬件直通，首先需要在开发板内核上开启 PCI-Express 支持以及 vfio 功能，在 PC 机上进入开发板 menuconfig，找到 Bus support 并进入[34]，将框内的所有选项全部勾选，即开启对 PCI-E 设备的支持，如图 7.2 所示。

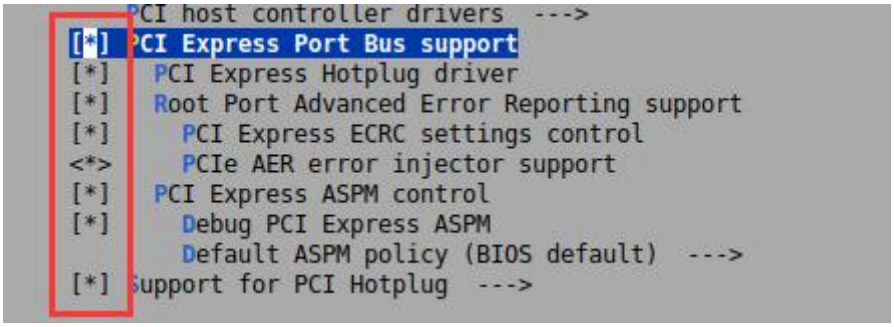

图 7.2 在内核中开启 PCI-E 支持

然后返回上一菜单，选择 Device drivers，找到 VFIO Non-Privileged userspace driver framework，点击进入后，勾选 VFIO support PCI devices，如图 7.3 所示。

图 7.3 在内核中开启 VFIO 支持

选择 Save 保存配置文件，并且编译成 kernel.img 固件，与第三章一样的方式将编译好的固件烧写进入开发板内，此时开发板已经可以支持 PCI-E 设备的接入。

接着将 M.2 转 PCI-E 转接板接入开发板 M.2 接口，用螺丝固定，然后，将三亚科技的加密卡接入转接板，并外接电源，然后对开发板通电，启动开发板系统。

然而，通电后发现，一旦接入该 PCI-E 设备后，发现开发板无法正常启动，更换为 intel 的服务器网卡也不能启动，通过串口 debug 输出启动日志，发现启动过程中出现了 crash 导致内核启动失败，如图 7.4 所示。

```
[    0.450286] pci_bus 0000:00: root bus resource [mem 0xfa000000-0xfa5fffff]
[    0.450913] pci_bus 0000:00: root bus resource [io  0x0000-0xfffff] (bus address [0xfa600000-0xfa6
[    0.452168] pci 0000:00:00.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.454853] pci 0000:01:00.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.468203] pci 0000:02:00.0: bridge configuration invalid ([bus 02-ff]), reconfiguring
[    0.468931] pci 0000:02:01.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.469644] pci 0000:02:02.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.470365] pci 0000:02:03.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.471086] pci 0000:02:04.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.471799] pci 0000:02:05.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.472520] pci 0000:02:06.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.473241] pci 0000:02:07.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.473962] pci 0000:02:08.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.474674] pci 0000:02:09.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.475394] pci 0000:02:0a.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.476115] pci 0000:02:0b.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.476836] pci 0000:02:0c.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.477548] pci 0000:02:0d.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.478269] pci 0000:02:0e.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.478990] pci 0000:02:0f.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.479702] pci 0000:02:10.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.480423] pci 0000:02:11.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.481143] pci 0000:02:12.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.481863] pci 0000:02:13.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[    0.482575] pci 0000:02:14.0: bridge configuration invalid ([bus 00-00]), reconfiguring
```

图 7.4 firefly-RK3399 接入 PCI-E 设备的报错信息

将此问题在必应中搜索发现，不少海外开发者在不同的开发板上也遇到过同样的这种问题[35]，虽然设备都不相同，经过分析后只能认为可能是开发板的问题（因为有公司推出过 RK3399 方案的，带有标准 PCI-E 接口的开发板，所以该方案是绝对支持标准 PCI-E 设备的）。

为了解决这个问题，通过联系厂家后，获得了一个可以使用的设备，如图 7.5 所示。

图 7.5 可用于 Firefly-RK3399 的 PCI-E 设备

该设备可以将开发板上面的 M.2 接口转换为 SATA 接口，从而可以支持接入标准 2.5 寸固态硬盘或者机械硬盘。

将该设备接入，并连接硬盘后，启动开发板，进入终端输入 lspci，显示出该设备“ASMedia Technology Inc. ASM1061 SATA IDE Controller”，说明该设备已经添加成功,同时，在 sys/bus/pci 下找到 vfio-pci 文件夹，说明 vfio 模块已经安装到位，可以开始使用 vfio 进行测试了[36]。

在 linux 中 sys/bus/pci 目录下找到 devices 目录，打开后发现一个名叫“0000:01:00.0” 的文件夹，该文件夹就是该设备对应的文件目录，输入命令“echo vfio-pci > /sys/bus/pci/devices/0000:01:10.0/driver_override ”，向 vfio 设备绑定该设备，同时，该设备属于 AHCI 设备，因此在/sys/bus/pci/drivers 目录下找到“ahci”目录，并找到该设备的驱动文件，其中有一个“unbind”的文件，用途是将设备从物理机上卸载，执行该程序卸载设备。最后，将该设备重新挂载，就会挂载到 vfio 上[37]。

在 QEMU 启动时添加如下信息：“-device vfio-pci,host=0000:01:01.0”将该设备挂载启动。启动成功说明该设备已经成功挂载[38]。

7.3 实验结果与分析

将该设备成功挂载后，在虚拟机的 fedora server 中发现该硬盘已经成功挂载进虚拟机[29]，如图 7.6 所示。

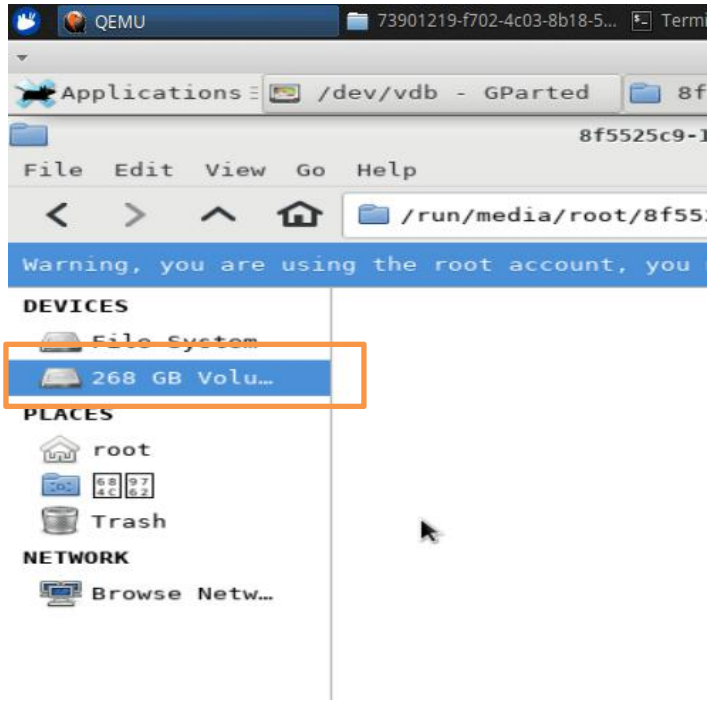

图 7.6 QEMU 虚拟机挂载 PCI-e 设备的效果

将文件读写进该目录内，传输速度与物理机几乎相同[39]，最终结果说明在 ARM 下同样支持虚拟机硬件直通技术。那么根据这个结果也可以得出，可以在 ARM 虚拟机接入网卡等其他 PCI 设备，甚至可以做到显卡直通，来解决 ARM 下的图形加速问题。

7.4 本章小结

本章通过参考 INTEL 下设备直通的功能，尝试在 ARM 下实现 PCI 设备的硬件直通。

首先在内核中开启 PCI-E 设备支持以及 vfio 内核设备直通功能，然后将 PCI-E 设备连接到 ARM 开发板上，然后再通过 vfio 方式将设备直通至 qemu 虚拟机中，并且对实验的结果进行分析，证实了在 ARM 下的虚拟化同样可以支持硬件直通。

8 总结与展望

本章对所做的工作进行总结，归纳文章的创新点，并分析文章的不足之处，提出进一步的研究目标。

8.1 工作总结

首先，分析了当前 ARM 平台的性能发展情况与基于 ARM 平台虚拟化技术的发展现状，根据当前技术条件下，通过 RK3399 的开发平台，将 KVM 虚拟化调通并且使用 QEMU+KVM 的方式安装虚拟机操作系统，并在此基础上，实现一些功能扩展，具体的工作内容如下：

（1）将 UEFI 移植到 QEMU-ARM 指定的虚拟机上，并且使其成功启动并且引导系统。

（2）修改 KVM 使其可以识别 Cortex-A72 而不是误认为通用 V8 处理器的问题，并且通过一系列方式使用针对 Cortex-A57 的性能优化，使得 RK3399 平台上的虚拟机可以有更高的性能加速，最后通过测试证明这种确实有明显的加速效果。

（3）在 QEMU 上做出改动，针对 QEMU 推出的 ARM 下显示设备 ramfb 的源代码进行分析和修改，通过读取设备中的缓冲区图形信息，将该信息进行分析并且处理后，写入 linux 下的 framebuffer，从而实现双屏异显的效果。

（4）在 ARM 下尝试实现类似 intel 虚拟化下的硬件直通，将 PCI-E 设备接入开发板并且直通进入虚拟机内，从而扩展虚拟机的功能。

8.2 研究展望

目前 ARM 下的虚拟化技术，尚处于初级阶段，这需要 ARM 公司、下游 SOC 芯片厂商、开源软件平台公司、开源软件开发者等多方努力的技术探索和代码贡献，因此任重而道远，而在文章中提出的几个实现方案，其实都还有可以继续深度研究的地方：

（1）根据 ramfb 设备可以读取图像设备数据的特性，就可以考虑在虚拟机出现

故障时，将当前故障附近时间点的屏幕录制下来，便于开发或者运维人员调查故障原因，从而更顺利解决故障[39][40]。

（2）目前所有的实现都是基于 linux 系统的，由于微软开始与高通公司合作，推出了 windows 10 ARM 版[41]，并且已经推出了多款设备，例如基于骁龙 835 的华硕骁龙本和基于骁龙 850 的华为 MateBook，微软甚至基于 windows 10 ARM 版本推出了对应的 windows server 2016，且均通过 UEFI 来启动，也就是说，虚拟机完全可以运行 windows 10 ARM 或者 windows server 2016，但是目前问题在于驱动尚未移植，但是由于 QEMU 的设备和驱动均为开源，这个可以在后期去移植实现，而对于第三方设备的驱动，也可以由第三方设备厂商去实现。

（3）目前讨论的虚拟化仅仅是 CPU、内存和 I/O 三部分，然而在开发和实验中也发现，在 ARM 平台上运行的虚拟机，有一些图形性能方面的损耗[42]，由于目前的虚拟化图形加速技术都是在 X86 下的，因此图形加速是 ARM 虚拟化研究的方向之一，既可以考虑像第五章中实现的 PCI-E 直通的方式实现，也可以考虑像 virtio-gpu 的方式去实现（值得注意的是，该模式暂时对 ARM 的图形芯片无效，只能达到类似 VGA 显示的效果）。

（4）当前的虚拟化技术仅仅在 KVM 下讨论，但是其他的技术，例如 Xen、VMware 等虚拟化技术，其实也可以借此机会跟进，通过 KVM 下的虚拟化研究显示，KVM 在 ARM 下的运行效率相对于 X86 来说，仍有一定优化空间，况且现在有 ARM 架构的大型服务器面世，相信未来 ARM 服务器将会出现相当数量的市场份额，未来前景十分可观。

致　　谢

两年的研究生学习生涯很快就要过去，在这两年，通过了很多项目的实战和研究，使我对计算机技术这个专业，相对于本科时代，有了更多更深的认识和了解，也丰富了我的人生阅历，回首过去，感慨万千，在这里，我将感谢我的家人、同学、老师，如果没有他们的支持与帮助，我将不会顺利完成这两年的学业。

首先，我要感谢我的导师付才教授，这两年来，不论是工作、学习还是生活上，付才老师都给予了我很大的帮助。在平时的科研工作中，对我要求严格，每周定期与我们开会讨论，交流想法，同时给我很多项目让我有机会锻炼自己的业务能力，在困难的时候给予我很大的帮助，有不懂的问题也会及时给出指导性的意见，也给我提供了很多实验所需的设备材料，使我可以完成很多实验。

其次，我要感谢信息安全实验室的韩兰胜、刘铭老师，在平常的生活工作中也给予了我很大的支持，也经常给出指导性的建议，也监督我平时的工作，让我更加严格要求自己。

然后，我要感谢信息安全实验室的各位同学，在平时的工作中相互帮助，相互鼓励，共同交流想法，交流技术，有时还共同参与项目，在项目研发的过程中不仅积累了相互合作的经验，也加深了同学之间的友谊。

特别的，我要感谢 Raspi System Dev 树莓派嵌入式开发群的各位群员们，群内对嵌入式有深入了解的成员非常多，所以在嵌入式方面有很多不懂的问题，都能得到群员们的专业回答，没有你们的耐心解答和技术交流，相信我也不会在短期内能够理解 ARM 架构的技术特性，也就无法完成论文的编写工作。

同时，我要感谢我的父母，在我读研期间给予生活上的关心和大力支持，你们是我能够顺利完成研究生学习生涯的坚强后盾，感谢你们为我付出的一切。

最后，衷心感谢各位专家评委百忙之中抽出时间对我的论文进行评审，感谢你们为我提出的宝贵意见。

参考文献

[1] 任永杰，单海涛. KVM虚拟化技术：实战与原理解析：principles and practices[M]. 机械工业出版社, 2015. 38~67.

[2] 王中刚, 薛志红, 项帅求. 服务器虚拟化技术与应用[M]. 人民邮电出版社, 2018. 60~77.

[3] Chisnall, David. The definitive guide to the Xen hypervisor [M]. Upper Saddle River, N.J. : Prentice Hall, 2008.

[4] Barham P, Dragovic B, Fraser K. Xen and the art of virtualization[C]// Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003: 164~177.

[5] 石磊, 邹德清, 金海. Xen虚拟化技术[M]. 华中科技大学出版社, 2009.30~52.

[6] 王春海. VMWare虚拟化与云计算. VSphere运维卷[M]. 中国铁道出版社, 2018. 20~42.

[7] Dave Mishchenko.VMware ESXi : planning, implementation, and security[M]. Boston, Mass. : Course Technology, Cengage Learning, 2011. 40~79.

[8] Kelbley, John,Windows server 2008 R2 Hyper-V: insiders guide to Microsoft's Hypervisor [M]. Indianapolis, Ind. : [Sybex]/Wiley Pub, 2010. 120~133.

[9] 马博峰.Windows Server 2012 Hyper-V虚拟化部署与管理指南[M]. 机械工业出版社, 2014. 50~77.

[10] 谭会生, ARM 嵌入式系统原理[M]. 西安电子科技大学出版社, 2017. 58~62.

[11] Yeung,Alfred , Partovi, Hamid , Harvard, Qawi , Ravezzi,Luca,Ngai, John , Homer, Russ,Ashcraft, Matthew , Favor, Greg. 5.8 A 3GHz 64b ARM v8 processor in 40nm bulk CMOS technology [J]. Applied Micro, Sunnyvale, CA,2014: 131~171.

[12] Christoffer Dall; Shih-Wei Li; Jin Tack Lim; Jason Nieh; Georgios Koloventzos. ARM virtualization [J]. ACM SIGARCH Computer Architecture News, 2016,31(2): 172~215.

[13] G. Wang, Q. Liu, J. Wu, and M. Guo. Monitoring of malefactor's activity in

virtualized honeypots on the base of semantic transformation in Qemu hypervisor [J].IEEE 2009, 30(5): 120~171.

[14] Muench, Daniel , Paulitsch, Michael , Herkersdorf, Andreas. Temporal Separation for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using PCIe SR-IOV [J]. Architecture of Computing Systems (ARCS), 2014 27th International Conference on, 2014,5(3): 120~212.

[15] Wang C, Wang Q, Ren K, et al. Platform Device Assignment to KVM-on-ARM Virtual Machines via VFIO[J]. Embedded and Ubiquitous Computing (EUC), 2014, 62(2): 324~481.

[16] 戴正华. UEFI 原理与编程[M]. 机械工业出版社, 2015. 60~77.

[17] Wolfgang Ecker, Wolfgang Müller, Rainer Dömer.. Hardware-dependent Software [electronic resource] : Principles and Practice[M]. Dordrecht : Springer Netherlands, 2009. 60~77.

[18] 韩德强, 马骏, 王宗侠, 高雪圆. 基于ARM平台的UEFI开发与移植 [J].电子技术应用，2014, 40(04): 11~14.

[19] 高志君. Linux 系统管理与服务器配置:基于 CentOS7 [M]. 电子工业出版社, 2018. 20~78.

[20] 姜先刚, 刘洪涛. 嵌入式Linux驱动开发教程 [M]. 电子工业出版社, 2017. 30~71.

[21] 吴国伟, 姚琳, 毕成龙. 深入理解Linux驱动程序设计 [M]. 清华大学出版社, 2015. 57~60.

[22] 瑞芯微Rockchip基于RK3399高性能计算芯片 [J]. 集成电路应用，2017, 32(02): 20.

[23] Rockchip,RK3399 - Rockchip open source Document [DB/OL], http://opensource.rock-chips.com/wiki_RK3399, 2018.

[24] Shye-Tzeng Shen , Shin-Ying Lee , Chung-Ho Chen. Full system simulation with QEMU: An approach to multi-view 3D GPU design[C]//IEEE International Symposium on Circuits and Systems (ISCAS), Proceedings of 2010 , 2010.

[25] Zhang B, Wang X, Lai R, et al. A Survey on I/O Virtualization and Optimization[M]. Communication and persuasion . 中国传媒大学出版社, 2010: 117~123.

[26] QEMU forum,Documentation/Platforms/ARM [DB/OL], https://wiki.qemu.org/Documentation/Platforms/ARM, 2018.

[27] 梁志，郭惠婷，吴跃前. 图像显示之Linux Framebuffer [J]. 电子世界, 2018, (19): 81~83.

[28] Jon Masters, Richard Blum. Professional Linux programming [M]. Indianapolis, Ind. : Wiley, 2007. 79~82.

[29] Peter van der Linden, Expert C programming : deep C secrets [M]. 人民邮电出版社, 2015. 61~79.

[30] 王春海, 虚拟化技术实践指南 : 面向中小企业的高效、低成本解决方案 [M]. 机械工业出版社, 2017. 73~27.

[31] Xiao-Feng GU, Wang J. Full-Virtualization Implementation of XEN Based on Intel VT-x[J]. Computer Technology & Development, 2009.

[32] 王齐, PCI Express体系结构导读 [M]. 机械工业出版社, 2010. 30~90.

[33] 韩雪涛, 图解计算机主板维修快速入门 [M]. 人民邮电出版社, 2009. 70~97.

[34] Robert Love, Linux kernel development [M]. China Machine Press, 2011. 10~27.

[35] k-a-z-u, PCIe-port not working on RK3399[EB/OL]. https://github.com/rockchip-linux/kernel/issues/116, 2018.

[36] 鸟哥, 鸟哥的Linux私房菜. 基础学习篇 [M]. 人民邮电出版社, 2018. 60~77.

[37] Eric Auger , KVM PCIe/MSI Passthrough on Arm/Arm64[J/OL]. https://www.linaro.org/blog/kvm-pciemsi-passthrough-armarm64/, 2016.

[38] Richard Petersen, Beginning Fedora Desktop : Fedora 20 Edition [M]. Berkeley, CA : Apress : Imprint: Apress, 2014. 63~79.

[39] 庞丽萍. 操作系统原理(第三版)[M]. 华中科技大学出版社, 2000. 30~67.

[40] 胥峰, 杨俊俊. Linux运维最佳实践[M]. 机械工业出版社, 2016. 52~79.

[41] 郭强. Windows 10深度攻略[M]. 人民邮电出版社, 2018. 82~109.

[42] Virtual Open Systems,The Virtual Open Systems video demos to virtualize ARM multicore platforms[DB/OL]. http://www.virtualopensystems.com/en/solutions/demos/kvm-full-virtualization-cortex-a15-vexpress/, 2014.