Key notes

武成岗老师 计算所

隔离机制与隔离原语：

混合内核：宏内核 + 隔离层，性能和安全性均合适，不过可能还有待验证

隔离原语：

1. SFI: Intel MPX
2. Domain-based Isolation: Intel MPK
3. 普通指令和特殊指令之类的

三大研究思路：

1. 设计专用的硬件支持
2. 挖掘现有硬件能力
3. 基于持续随机的隔离性

方向1:

MPK应用的进程内隔离 这里有两个工作，ERIM和Hondor

Donky: 扩展RISC-V，使得某个地方有权限

IMIX: SMOV指令被引入

HFI: 通过硬件来实现SFI

SecureCells: VMA为粒度划分若干个Cell，不同SD对于不同CeII有独立访问权限

方向2:

Chcore，利用Intel MPK提升微内核IPC速度，把推到用户态的放在内核态处理

SeCage, xMP，利用EPT做划分和隔离

Nested Kernel, 把区域权限设置成只读权限，要写的时候，则翻转CR0的WP位，让Nested Kernel有写权限

SEIMI: 利用了SMAP机制和SMEP机制，开启关闭的时间只要8.6个时钟周期

希望隔离用户态的一部分数据进行隔离。把关键数据放到U page，利用SMAP进行防护

CETIS: 挖掘CET应用在进程内的隔离潜力 CET机制是为了实现Shadow Stack机制，阻止ROP攻击的新增硬件特性

把敏感数据放在影子栈的下面，其他指令没法访问敏感数据，只有WRSS指令能访问

面临的挑战是地址位置对齐上的限制

ARM Memory Domain机制

ARM TTBRx应用在内核内的隔离

PANIC: 利用ARM的PAN机制，和Intel的SMAP机制很像，增加了Load/Store unprivileged(LSU)

把应用程序放在内核态下，依然存在攻击危险，在ARM中换了一套机制来缓解这个问题，在用户空间下使用TTBR0寄存器

问题

分割隔离机制为什么会有生命力？

- 被动防御手段的参差不齐

隔离开销容忍程度？

- 取决于应用需求

将用户态放到内核态是否会增加TCB？

- 已经利用隔离机制隔开，对于内核关键组分影响不大

李振华 清华大学

移动模拟器的虚拟化研究

移动模拟器，从华为安卓模拟器到鸿蒙模拟器

发现绝大部分情况下，手机都在等待宿主机在渲染，就会出现卡顿和停机

发现OpenGL中只有7条指令画三角形，剩下993条是做虚拟机上下文转换的。

为了减少重型游戏停机问题，就不要再用GPU了，手机没有这个东西，不如让手机CPU直接来干这件事。

字节跳动的参与，希望探索ARM架构的高保真、轻量级模拟器，帮助他们自动化发现抖音等软件的安全缺陷

大规模移动的虚拟化测试： 在开放移动市场系统，各手机厂商各有生态，要一起做测试是有难度的。

抖音团队的做法：字节跳动搞了一个物理设备农场，把各种各样的手机一起采购过来。

清华的做法：还是希望用云的方式来做，因此去做了一下这个调研，不过当前主流市场中的云还是有问题的，他们自己的测试流水线是不完备的，这是最大的问题，没法进行定制化测试。

结论：目前基于云的移动测试是不适用于国际APP的。

有没有办法自己搭建一个云端平台，既有虚拟化，效率还可以？以前确实有人提出这样的想法，不过很多人认为还是做不成。

在395台ARM服务器中，部署了5918台虚拟设备。

使用catofish，不使用google的安卓模拟器（有很多hook，而且会改变原来的行为）

设备模拟放在KVM，Kernel这边的层

测试与调试工具： 使用字节跳动的基于模型的自动化测试机制，效率比较高，也不会死循环

在虚拟设备上，测出415k次测试故障，在硬件设备中，则测出390k次测试故障，说明保真性还是做不到。

发现一件事，如果设计比较合理的话，能够达到较高保真率

又发现一件事：手机上自己的一些稀奇古怪小东西并不会对第三方APP产生影响，就硬件层面的保真性。

反而是大名鼎鼎的做驱动的，他们的GPU驱动中会有很多bug，会导致物理设备在渲染上出问题，虚拟设备上没问题。因为虚拟设备没法模拟。

锤子手机有用个锤子。

CTS与VTS只能做每个模块的单元测试，但是跨越模块的做不到。

每一个手机设备视频格式的解释是有微妙的区别 -> virtio上的分配有所区别

地区上的问题：比如我国，应用生态比较流氓，操作系统比较激进，喜欢杀进程。只好适应一下生态。

在模拟器上，把激进的杀进程机制也添上。

那GPU驱动的毛病怎么办？人不给修，采用二进制污点分析和补丁机制来告诉别人，你TM乱写。人真给改了。

在完成了以上的工作时，达到了抖音的要求。

问题：

怎么快速确定bug的具体位置？这边有很多层，运营商，应用等等。

- 由表到里，先从安卓java层，再从kernel层找。如果这也找不到的话，那可能是运营商私有模块的问题，需要用污点回溯问题，一点点去跟踪。
- 当然，就和医生看病一样，越往下深入，越来越难。

Session 1: Arch

樊树霖 IPADS

对ISA资源提供一个管控机制，过去的研究对于ISA资源尚没有足够的重视

对ISA资源做混合粒度的隔离，通过混合权限检查来做。

不同Domain所能够使用的ISA资源是不同的，切换Domain，通过0/1来确定是否可执行的选项。

采用了bit来做这边的细粒度控制，采用位操作来做其对应的权限检查。

模拟平台：RISCV和X86 -> rocket core & gem5

强调了ISA资源的重要性

冯寅潇 清华大学

芯连互联网络上的工作

成本危机，NRE即设计成本同样增加

芯粒架构，现有的芯粒架构大体上是一个平面拓扑，存在潜在问题

有没有办法用单个芯粒来做一个系统？

在片上网络，怎样才能做到拓扑可变？希望能够是一个软件定义可变。

涉及到片上网络，建议看完caaqa后进行略读。

评估方法是通过时钟级别精度的C++仿真器来做

问题：

软件定义接口如果真正使用的话，是可适配的，从实现技术来说，支不支持？

- 物理层上是独立的，软件层上做了一个抽象，以实现绑定在一起

王嵩岳 计算所

xAMU：为大规模乱序内存访问的加速单元

相对于传统的DRAM，现在的一些内存访问单元延迟比较慢，比如CXL。

这个缓慢对于现代处理器来说是不可以容忍的

Modern CPU is not Ready for Far Memory

ROB经常会出现满载的情况，提高内存延迟时，会以指数级来增长

能否直接通过关键资源的项数来减少延迟，不太能够：开销太大，不可以无限上涨

如何在现有的Out of order情况下来屏蔽更长的延迟？

异步预取是可行的吗？

- 不太可行，会引入很多新的问题，比如stateless

idea1: AMI，添加异步内存指令

通过异步手段缩短内存指令运行时间

idea2: AMU，通过一个ScratchPad Memory（SPM）来进行管理，是动态地在L2级Cache中分割出来的

idea3: 使用更多内存semantics，利用aset等其他新增指令

借用协程的思想，通过aload astore指令来做改写，运行的时候，利用一个协程调度器，来查询协程是否完成了工作。

问题：

与DMA edging有什么联系

- 通过新指令，能够更快启动。

异步上的内存访问以后是否会替换掉当前同步内存访问？

- 异步有他的适用场景，不过主要来看并行场景，有些东西确实优化起来比较困难。完全替代可能做不到。

欠缺的问题，手动识别循环与数据是否可以并行。将来做编译器，可能还是用OpenMP来标记，这可能做不到完全的适配，有隐式的问题

那个异步语义保障能否做对？

- 可以锁的机制来显式控制，来保证语义上的正确性，并且开销也不是很大。

田博宇 清华大学

Near-Data Processing的问题

热数据会吸引过来很多的任务，这种负载不均衡的问题需要解决。

工作目标：同时优化动态数据caching和computation scheduling

1. task-based programming model
2. Travelling cache
3. 同时满足上述要求的特殊调度策略

Baseline:

Task-based Model，采用pagerank，来找到核心关键点

Adopt a task-based programming and execution model, similar to Swarm

硬件架构：不再赘述，我拍照了

Traveller Cache:

Distributed DRAM caching

有限位置，更高效的实现

Hybrid Task Scheduling Policy:

打分公式，有数据信息和数据位置载荷，还有一个是workload情况，还有一个折扣Hybrid Scheduling Weight

问题

Distributed Shared Memory System? consistency与conherency问题怎么解决？

- 每一个Task有一个时间戳以防止冲突
- conherency粒度还是比较大

如果有很多副本的话，会不会降低全局内存利用率？

- 这也是为什么我们设置一个区域限制机制，用于cache的部分只有1/8或者1/16，稍微牺牲了一点DRAM开销，利用率的影响不算特别严重

周可行 北京大学

https://eecs.pku.edu.cn/info/1040/5917.htm

英雄出少年

改变仿真顺序，重排顺序，从而尽可能减少内存访问的次数，降低仿真开销

里头的优化思想充满着形式化和流水线优化的韵味，汗流浃背了（指令重排

采用了一些建模和形式化上的分析，以把问题梳理地更加清晰

工作做的比较完整，在多个硬件仿真上均做了优化处理，能够得到不错的结果。

能够把多个pipeline缩减成一个pipeline，性能提高比较显著

单线程和多线程的性能均做了评估，虽然Khronos是一个单线程仿真器，但是他在部分demos上有与多线程仿真器上有相当的仿真性能

多线程仿真器并不总是比单线程更快，其往往采用悲观的方法来仿真