异构多核平台上的共享资源争用分析与缓解：一项工业案例研究

Michael Bechtel、Heechul Yun

arXiv:2304.13110v3 [cs.PF] 8 Apr 2024

摘要——本文提出一种解决方案，应对 ARM 于 2022年提出的工业挑战。我们系统地分析了在异构多核平台 NVIDIA Jetson Nano 上，工业挑战中的增强现实抬头显示器（AR-HUD）案例研究应用受到共享资源争用的影响。我们对 AR-HUD应用进行配置，使其能够在该平台上以 20赫兹的实时速率处理传入的图像帧。我们使用微架构拒绝服务（DoS）攻击作为该挑战中的攻击者工作负载，并展示了这些攻击会显著影响 AR-HUD应用的延迟和准确率。尽管我们尽力通过缓存分区和对 AR-HUD应用进行实时调度来缓解这些影响，估算轨迹仍与已知真实值存在显著偏差。为应对这一挑战，我们提出 RT-Gang++，一种分区实时组调度框架，具备末级缓存（LLC）和集成 GPU 带宽节流能力。通过应用 RT-Gang++，即使存在满负载的干扰任务，我们也能使 AR-HUD应用达到预期性能水平。

索引词—工业挑战赛、实时、SLAM、微架构DoS攻击

1 引言

异构多核计算平台越来越多地应用于安全关键型信息物理系统（CPS），因为它们能够在满足尺寸、重量和功耗（SWaP）约束的同时，显著提升性能。然而，此类平台中计算单元之间对共享微架构资源（如共享缓存和主存）的争用仍然是一个重大挑战，因为它可能影响关键实时任务的执行时序，从而危及信息物理系统的安全。此外，恶意行为者也可能蓄意引发共享资源争用，以损害信息物理系统的性能和安全性。这类对手被称为微架构拒绝服务（DoS）攻击者 [9]，对于需要在单个多核平台上同时运行多个并发应用的高性能信息物理系统而言尤其棘手。鉴于信息物理系统日益互联的发展趋势（ARM 的 SOAFEE 计划 [4]便是例证）面向汽车行业，可以预见，此类拒绝服务攻击可能会被远程部署到未来的信息物理系统中。

近年来，理解并解决多核系统中的共享资源争用问题一直受到学术界和工业界的高度关注。特别是，ARM 于2022年发起了一项工业挑战赛，旨在解决共享资源争用问题 [3]。该挑战赛围绕汽车应用中的增强现实抬头显示器（AR-HUD）案例研究展开。该案例研究应用由两个主要组件构成：视觉同步定位与地图构建（SLAM）任务 [15] 和基于深度神经网络的驾驶员头部姿态估计任务 [16]。SLAM 任务由三个主要线程组成，全部运行在 CPU 上；而基于深度神经网络的头部姿态估计任务（下文简称深度神经网络任务）可能会使用图形处理器。该应用代表了一种计算密集型混合关键度实时系统，必须利用高性能异构多核嵌入式平台。因此，该挑战赛旨在寻找相应方法，以分析并优化此类关键实时任务的性能界限，即使存在可能在访问共享资源时与关键实时任务发生争用的“干扰任务”，也能实现这一目标。

本文首先研究共享资源争用对 ARM 工业挑战赛 AR-HUD案例应用性能的影响。通过这项研究，我们旨在回答以下问题：(1) 我们能否在具有代表性的异构片上系统（SoC）处理器上安全地整合 AR-HUD案例中的两个实时任务（同步定位与建图算法和头部姿态检测），并实现所需的实时性能（即满足截止时间）？(2) 两个 AR-HUD任务之间的共享资源争用是否会影响同步定位与建图任务所获得的位置/轨迹估计的准确率？(3) 在存在干扰任务的情况下——这些任务可能经过恶意设计，以造成高强度的共享资源争用——我们能否在不过度配置的前提下，保证 AR-HUD应用在准确率和延迟两个方面达到期望的性能水平？

为回答这些问题，我们在 NVIDIA Jetson Nano 上系统地开展了实验。该平台是具有代表性的异构嵌入式多核平台，配备四核 ARM Cortex-A57 CPU 和集成 GPU。我们的研究结果如下：我们能够配置

1. *Bechtel* 博士目前任职于 *Garmin*。该项工作是在他任职于堪萨斯大学期间开展的。电子邮箱：*mgbechte@gmail.com*2. *Yun* 博士任职于堪萨斯大学，是本文的通信作者。电子邮箱：*heechul.yun@ku.edu*

AR-HUD应用，使其能够以 20 赫兹的实时频率处理输入图像帧。然而，我们发现，两个实时任务之间的争用确实会显著影响 SLAM 任务的准确率。即使该任务能够以 20 赫兹的实时频率处理所有输入图像帧，估计轨迹仍会明显偏离真实值。此外，我们发现，缓存组感知型拒绝服务攻击 [7] 对 AR-HUD应用的准确率和实时性能影响尤为显著。具体而言，当缓存组感知型拒绝服务攻击任务以尽力而为（非实时）任务的方式，与 AR-HUD应用的实时任务共同调度，以充分占用系统资源时，由于争用导致延迟增加，SLAM 任务不得不丢弃大部分输入图像帧，甚至无法生成轨迹。

为应对这一挑战，我们提出 RT-Gang++，这是一种分区式实时组调度框架，具备集成 GPU 和末级缓存（LLC）带宽节流功能。RT-Gang++ 基于 [2]，并在此基础上扩展了以下功能：（1）增加对分区式组调度的支持，以允许多个不同优先级的实时组并发执行；（2）增加对末级缓存和集成 GPU 带宽节流的支持，以防止这些共享资源发生争用。这些新增功能对于应对 ARM 工业挑战赛问题至关重要。

借助 RT-Gang++，即使存在恶意拒绝服务攻击，我们也能够在 Jetson Nano 平台上安全整合 AR-HUD应用，并实现预期的实时性能和准确率。此外，我们还将 RT-Gang++ 移植到 Raspberry Pi 4 平台，并评估了其有效性。为便于复现和传播，我们以开源形式发布了 AR-HUD应用的配置，包括我们移植到 ROS2 的 OV^{2}SLAM、评估脚本以及 RT-Gang++ [1]。

本文其余部分的组织如下。第2节介绍ARM工业挑战赛问题。第3节介绍该挑战中的微架构拒绝服务攻击。第4节讨论我们的实验设置。第5节介绍对该挑战案例研究应用的实证评估。第6节介绍一种缓解方法。第7节介绍实验结果。第8节回顾相关工作，并在第9节总结全文。

2 ARM工业挑战赛 2022年：增强现实抬头显示器（AR-HUD）应用

解决共享资源争用的影响，对于机器人技术和汽车领域中的许多高性能信息物理系统（CPS）而言至关重要。鉴于在CPS中整合高性能混合关键性应用的重要性日益提升，这一点尤为突出。为推动对该主题的进一步研究，ARM在ECRTS 2022年发起了一项工业挑战赛。该挑战赛以汽车领域中的一个增强现实抬头显示器（AR-HUD）应用作为案例研究。作为一种高级驾驶辅助系统（ADAS），该应用为车辆驾驶员提供实时的额外警报和通知。具体而言，这些警报通过增强现实（AR）技术叠加在现实世界的物体上。所提出的AR-HUD应用主要由两个组件构成：一个视觉SLAM任务和一个头部姿态估计任务。下面我们将简要介绍并讨论这两个AR-HUD组件。

对于许多自主信息物理系统（CPS）而言，定位和三维地图生成是实现真实世界性能的重要步骤。越来越多的CPS采用同步定位与建图（SLAM）算法，在一个步骤中完成这两项操作。在SLAM 算法中，系统接收输入传感器数据，并利用这些数据估计系统在给定环境中的当前位置，同时生成或更新该环境的三维地图。视觉（摄像头）和测距传感器（如激光雷达、激光传感器和声纳）均可用于SLAM。利用视觉的SLAM算法类别如今被称为视觉*SLAM*。

在 ARM工业挑战赛中，建议将 OV^{2}SLAM 算 [15] 法作为 AR-HUD 案例研究 [3]的一部分。OV^{2}SLAM 是一种面向实时应用的视觉SLAM 算法，除了关注 SLAM 性能，也强调处理时间。该算法由四个主要组件构成，每个组件分配给一个独立线程：

1）前端线程负责对相机传感器进行实时姿态估计。它还负责创建关键帧，用于生成周围环境的三维地图，但不会为每个输入帧都创建关键帧。请注意，该线程会针对收到的每个输入帧运行，这意味着它本质上是一项周期性任务。对于我们的评估，我们将逐帧截止时间设为 50 毫秒，因为所用输入数据集以 20 赫兹的频率回放数据。2）建图线程使用前端 生成的关键帧来生成新的三维地图点。它主要通过对关键帧进行三角测量来完成此操作。如果没有新的关键帧到达，它还会执行局部地图跟踪，以尽量减少漂移。与前端不同，建图 线程是非周期性的，因为它由事件驱动，仅在生成新关键帧时运行。3）状态优化线程执行两项主要操作。首先，它运行局部光束法平差（BA），以优化相机姿态估计。其次，它执行关键帧筛选，以防止在后续 BA 操作中处理冗余关键帧。请注意，该线程也是非周期性的，因为它依赖于建图 线程的输入，因此同样由事件驱动。4）回环检测 线程执行在线词袋（BoW）操作，以检测系统给定轨迹中的闭环。不过，我们在案例研究中不使用此线程，因为目标 AR-HUD应用 [3]不需要它。

请注意，只有每个输入帧送入 OV^{2}SLAM 时，前端线程才会运行。其余线程只会在必要时运行，例如创建新的关键帧时。

1. https://github.com/CSL-KU/ArmArHudChallen ge

默认情况下，OV^{2}SLAM 算法可以以三种不同模式之一运行：准确、快速和平均。准确模式会执行上述全部四个步骤，包括回环闭合，旨在尽可能提高准确率，同时保持 20 赫兹的控制频率。另一方面，快速模式会牺牲一些准确率，以实现快得多的 200 赫兹控制频率。为此，快速版本采用更快（但准确率较低）的关键点检测算法，并且不执行回环闭合步骤。平均版本的性能则介于另外两个版本之间。换句话说，它的控制频率介于 20 和 200 赫兹之间，准确率低于准确版本，但高于快速版本。不过，与准确版本一样，平均版本也会执行回环闭合。工业挑战赛建议使用快速版本，因为它具有出色的实时性能和良好的准确率。

对于 AR-HUD应用而言，同样重要的是，为驾驶员提供的 ADAS 警报应以符合驾驶员视角的方式显示。为此，可以估计驾驶员的头部姿态，以便根据需要校正增强现实显示屏。因此，AR-HUD应用在其第二个组件中采用了头部姿态估计任务。Industrial Challenge 建议使用 HopeNet-Lite 头部姿态估计器 [16]，，因为它可以在许多嵌入式异构多核平台上实时运行。我们将在评估设置中进一步讨论该组件。

3 微架构拒绝服务（DoS）攻击

作为 Industrial Challenge 的一部分，ARM 还设想在增强现实抬头显示案例研究 [3]中加入攻击者工作负载。这些攻击者工作负载可能会与 AR-HUD应用共同调度，并可能争用共享资源。

对于我们的攻击者工作负载，我们采用了文献 [38], [9], [7]中所述的微架构拒绝服务（DoS）攻击。这些 DoS 攻击针对多核平台中的各种微架构资源（例如末级缓存、动态随机存取存储器），即使跨核实时任务运行在专用核心上并拥有专用末级缓存分区，也可能导致其执行时间显著延迟。在本文中，我们希望了解此类 DoS 攻击可能对 AR-HUD应用造成的影响。我们在评估中采用的具体 DoS 攻击如下：

- IsolBench 套件中的带宽基准测试 [38]。该基准测试旨在以顺序方式持续访问目标共享资源（例如末级缓存或动态随机存取存储器）。具体而言，它以缓存行粒度对一维数组进行顺序访问（即所有访问之间相隔 64B）。我们将基于此基准测试的拒绝服务攻击称为*Bw*。

- IsolBench 套件中的延迟-*MLP* 基准测试 [38]。与带宽 基准测试非常相似，延迟-*MLP* 会持续访问目标资源，但由于采用指针追逐，其访问模式有所不同。具体而言，它会在多个并行链表（PLL）中进行随机访问。我们将基于此基准测试的拒绝服务攻击称为*PLL*。

- 来自 [7]的缓存组感知攻击。与内存感知攻击 [6], 类似，这些攻击基于上述*PLL*攻击，但经过修改后只访问特定缓存存储体，以便在访问末级缓存时产生最大的缓存组争用。我们将此攻击称为*BkPLL*。

此外，这些拒绝服务攻击还可以从两个方面进行额外配置。首先，所有攻击都可以配置为执行读访问或写访问。因此，我们在测试中对这两种访问方式的拒绝服务攻击均进行了测试。其次，如上所述，攻击者可以配置为访问末级缓存或动态随机存取存储器，因此我们分别采用针对每种共享资源的拒绝服务攻击。请注意，我们将*BkPLL*攻击配置为仅访问末级缓存，因为它们是专门针对该资源设计的。综合起来，在命名方面，我们对拒绝服务攻击采用以下命名约定：

<*DoS* 攻击类型><访问类型>（<目标资源>）

其中，*DoS* 攻击类型 是上述列表中的一种攻击，访问类型 可以是读或写，而目标资源 可以是 LLC 或动态随机存取存储器。例如，执行针对 LLC 的读访问的*Bw* 攻击实例将被称为*BwRead(LLC)*。在评估中，我们总共采用了 10 个不同的 DoS 攻击任务。

4 实验设置

本节将介绍针对 AR-HUD 应用的案例研究的实验设置。

硬件平台： 对于硬件平台，我们使用 NVIDIA Jetson Nano 平台。该平台配备四核 Cortex-A57 核心，每个核心都有自己的专用一级指令和数据缓存，所有核心共享一个全局 L2 缓存。表 1 展示了该平台的基本特性。

平台	NVIDIA Jetson Nano
SoC	Tegra X1
CPU	4 个 Cortex-A57 核心 @ 1.43 赫兹
GPU	128 核 Maxwell
共享末级缓存（二级缓存）	2MB（16 路）
内存（峰值带宽）	4GB LPDDR4（25.6 GB/s）

表 1：NVIDIA Jetson Nano 硬件规格。

应用设置： 如第 2 节所述，AR-HUD 应用由两个主要组件组成：OV^{2}SLAM 视觉SLAM任务和 HopeNet-Lite 头部姿态估计任务。

对于 SLAM 任务，我们采用工业挑战赛 [3],中推荐的 OV^{2}SLAM 快速 设置，该设置由三个线程组成：前端、建图和状态优化。每当摄像头以固定的 20 赫兹频率提供新的图像帧时，都会调用前端 线程。当前端 线程生成新的关键帧时，会有条件地调用建图 和状态优化线程（详情请参见第 2 节。）

由于同步定位与建图任务对实时性至关重要 [3]，，我们使用 Linux SCHED_FIFO 实时调度器，并为其分配实时优先级 2。此外，我们将这三个线程全部分配给

同步定位与建图任务的线程被分配到两个 CPU 核心（核心 0 和核心 1）上；我们通过实验确定，当它们独立运行时，这样的配置足以满足需求。请注意，同步定位与建图线程观测到的每核心 CPU 利用率最高不到 70%，这意味着仍有额外的余量可供尽力而为任务使用。

对于 SLAM 任务的输入数据，我们使用 EuRoC 数据集 [10] 中的五个 Machine Hall（MH）场景。需要注意的是，EuRoC 数据集包含微型飞行器（MAV）的视觉惯性数据，其中包括立体图像、惯性测量单元（IMU）测量数据以及准确的运动与结构真值数据。在 MH 场景中，数据采集于穿越室内环境的过程中，该环境中分布着各种复杂程度不一的机械设备；其中，MH01 场景对 SLAM 而言最简单，而 MH05 场景最困难。

数据集中的图像帧通过 rosbag2 [17], 实例输入 SLAM 任务；该工具用于回放记录在 ROS bag 文件中的数据集，并运行在核心 2 上，实时优先级为 2，因此不会被任何尽力而为任务延迟。需要注意的是，所有 MH 数据集均以 20 赫兹的频率回放输入数据。观测到 rosbag2 的 CPU 利用率约为核心 2 的 ∼5-10 %。

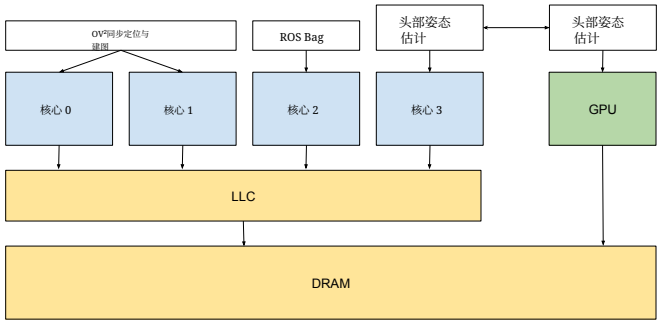


图 1：Jetson Nano 上 AR-HUD 案例研究的任务到核心分配。

对于头部姿态估计任务，我们使用推荐的 HopeNet-Lite DNN 模型 [16],，该模型是原始 HopeNet 模型的轻量级版本 [33]。需要注意的是，Jetson Nano 主要在图形处理器上处理 HopeNet-Lite 模型，但单个 CPU 核心（核心 3）也用于启动 GPU 内核并监控其进度。在 Jetson Nano 上，我们将 DNN 任务配置为以与 SLAM 任务相同的 20Hz速率周期性运行，并为其分配 1 的实时优先级，因为该任务被确定为非关键的高优先级任务 [3]。此外，我们将 HopeNet-Lite 任务固定到与 SLAM 任务核心不同的 CPU 核心（核心 3）上，以确保其所需的 CPU 操作（例如 CUDA 内核启动等）不会受到 OV²SLAM 线程的干扰。

图 1 直观展示了我们所采用的设置，以及如何将 AR-HUD 任务分配到目标平台的 CPU 核心上。此外，表 2 展示了 AR-HUD 案例研究中所有实时任务的任务/线程/核心映射关系及实时调度参数。

操作系统设置： 对于 Jetson Nano 的操作系统，我们运行 Ubuntu 18.04 和 Linux 内核 4.9，并通过 PALLOC [42] 进行修补，以支持末级缓存分区-

Task	线程	核心数	RT 优先级	Rate (Hz)
OV²同步定位与建图	前端	0,1	2	20
	建图			-
	状态优化			-
ROS Bag	-	2	2	20
头部姿态估计	-	3,GPU	1	20

表 2：增强现实抬头显示案例研究中 Jetson Nano 上的实时任务/线程/核心映射和调度参数。请注意，所有实时任务均使用 SCHED_FIFO 实时调度器进行调度，且优先级值越大表示实时优先级越高。

化。PALLOC 利用虚拟内存页转换，将页面分配强制限定到特定的页颜色。借助 PALLOC，我们将末级缓存划分为四个大小相等的分区（颜色），并将这些分区进行 2×2 划分。也就是说，OV²同步定位与建图算法获得两个末级缓存分区，而所有其他任务共享剩余的两个缓存分区。请注意，所有尽力而为任务（即使用 Linux 默认 CFS 调度器进行调度的任务）也共享后两个缓存分区，以最大限度地减少对同步定位与建图任务的性能影响，因为后者具有实时关键性。请注意，在 PALLOC 中，任务而非处理器核心可以映射到任意缓存分区。

5 分析共享资源争用的影响

在本节中，我们评估共享资源争用对 OV²同步定位与建图算法性能的影响。

5.1 协同调度拒绝服务攻击的影响

在本实验中，我们评估 DoS 攻击共运行任务的影响，以及它们是否能够有效降低给定场景下 OV²同步定位与建图的性能。需要注意的是，为了专注于同步定位与建图性能及其对拒绝服务攻击的敏感性，本实验不执行 HopeNet-Lite DNN任务。

实验设置如下：首先，我们使用 EuRoC 数据集 [10]中的 MH01 场景运行一个 OV²同步定位与建图任务实例。任务完成后，我们计算该算法相对于已知真实轨迹 [10]的 绝对轨迹误差（*ATE*）。随后，我们在所有四个可用处理器核心上运行 DoS 攻击者实例，重复进行实验。我们再次计算绝对轨迹误差，并将其与 Solo 情况进行比较，以确定攻击者是否产生了明显影响。

请注意，我们将 DoS 攻击作为尽力而为任务运行（使用 CFS 调度器进行调度），同时将 OV²SLAM 任务作为实时任务运行（使用 SCHED_FIFO）。由于 Linux 严格优先调度实时任务而非尽力而为任务，因此，拒绝服务攻击任务只能在未执行任何实时任务的处理器核心上运行。换言之，每当 SLAM 任务的线程准备就绪时，它们会立即抢占任何拒绝服务攻击者任务。还请注意，如前文所述，拒绝服务攻击任务与 SLAM 任务被分配到不同的末级缓存分区。这种隔离最大限度地减少了共同调度可能产生的负面影响，因为拒绝服务攻击任务无法驱逐缓存行

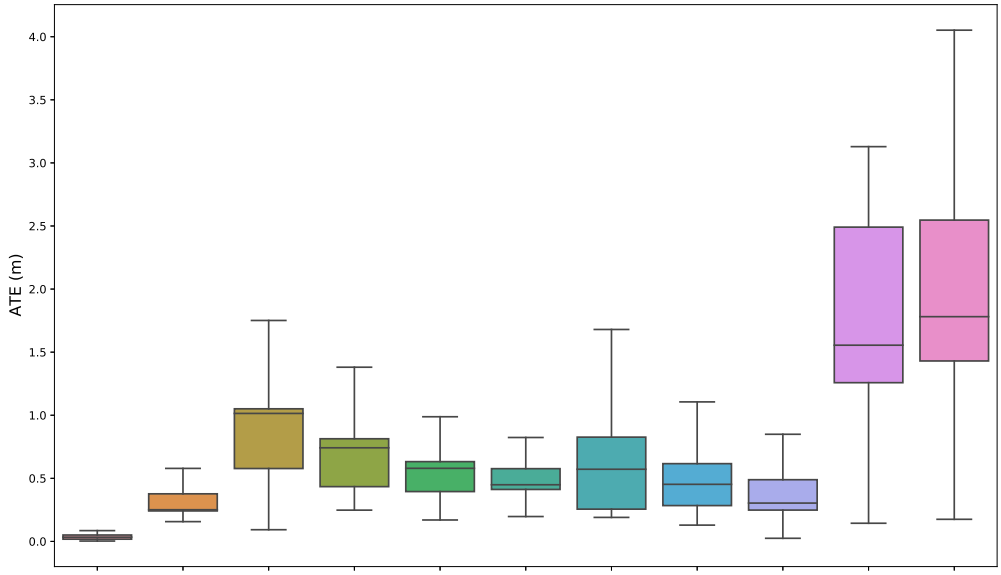


图 2：DoS 攻击对 OV²同步定位与建图生成轨迹的绝对轨迹误差（绝对轨迹误差）的影响。

SLAM 任务的缓存行。因此，观察到的任何延迟都不能归因于 CPU 调度或缓存空间争用。

图 2 展示了在整个 MH01 数据集持续时间内收集的 OV²同步定位与建图绝对轨迹误差箱线图，包括单独运行（Solo）以及分别与每种受测 DoS 攻击同时运行的情况（其余攻击位于 X 轴）。

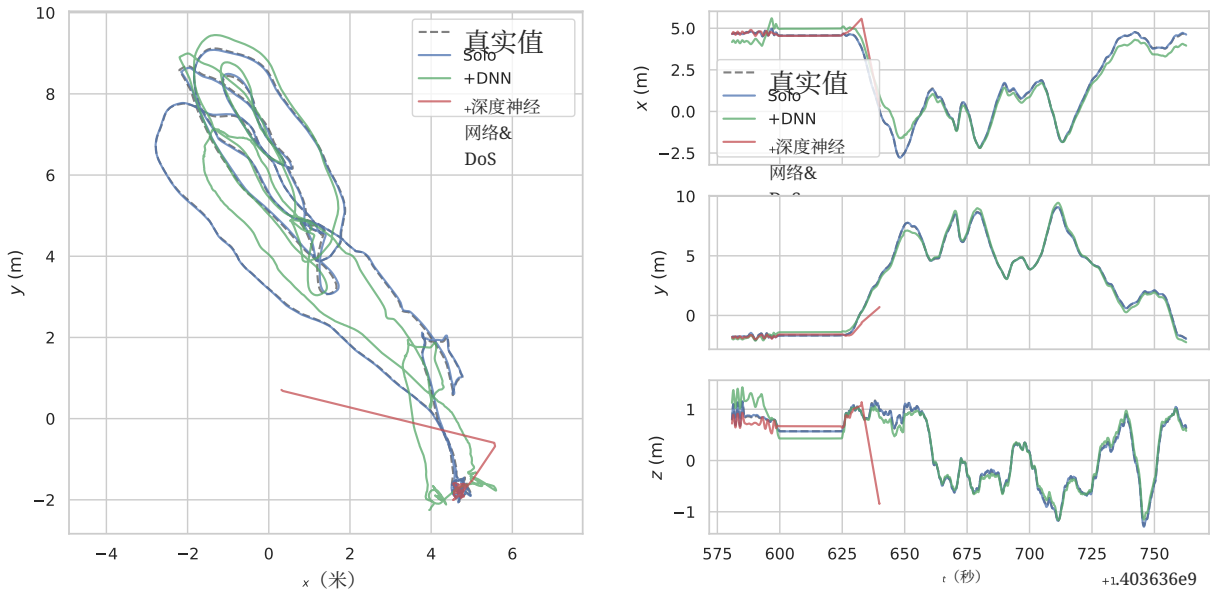
首先需要注意的是，所有受测 DoS 攻击都会对跟踪性能造成显著负面影响，使绝对轨迹误差最多增加 4.0 米。尽管 DoS 攻击无法抢占同步定位与建图任务，也无法驱逐其缓存行，但仍会出现这种情况。这是因为现代多核系统中还有许多其他共享硬件资源会影响执行时序。这些共享硬件资源包括动态随机存取存储器带宽 [43], [6],动态随机存取存储器 存储体 [42], 缓存内部缓冲区/队列 [38], [9], 以及缓存存储体 [7]。X 轴上的 DoS 攻击旨在使这些共享资源产生最大争用。需要注意的是，在此场景下， ∼0.3 或更高的绝对轨迹误差表示与真实值存在显著偏差，这在现实世界中可能导致故障（例如崩溃） [23]。

具体而言，我们观察到， [7], 中最近提出的缓存存储体感知型拒绝服务攻击分别针对读取和写入，被称为 BkPLL-Read（LLC）和 BkPLLWrite（末级缓存），在影响绝对轨迹误差方面尤其有效。具体来说，*BkPLLRead*（*LLC*）攻击将绝对轨迹误差的中位数提高到 1.7 以上（相对于 Solo 增加 49 倍），而 *BkPLL- Write* （*LLC*）攻击则将其提高到 1.9 以上（增加 55 倍）。简单来说，这些攻击导致 OV² SLAM 检测到的轨迹偏离真实值轨迹，中位数为两米，最坏情况下超过四米。这两种缓存存储体感知型拒绝服务攻击都经过专门设计，可向特定的末级缓存存储体生成大量并发访问，从而导致该存储体发生争用。Cortex-A57 的共享二级缓存由两个标签存储体组成，每个标签存储体包含四个可并行访问的数据存储体 [7]。通过将并发访问定向到一个数据存储体，SLAM 任务对缓存存储体的访问会被延迟，进而延迟 SLAM 任务的执行。

5.2 协同调度 HopeNet-Lite 对集成图形处理器的影响

在本实验中，我们评估将 HopeNet-Lite 头部姿态估计器与 OV²同步定位与建图任务进行协同调度对其性能的影响。我们比较以下配置：*Solo*、 *+DNN* 和 *+DNN&DoS*。在*Solo*配置中，OV²同步定位与建图单独运行；在 *+DNN* 配置中，HopeNet-Lite DNN 与 OV²同步定位与建图协同调度；在 *+DNN&DoS* 配置中，HopeNet-Lite 和 DoS 攻击任务均与 OV² 同步定位与建图协同调度。需要注意的是，DoS攻击者属于尽力而为任务，而 OV²同步定位与建图和 HopeNet-Lite 均属于实时任务。这意味着 DoS攻击者无法抢占 OV²同步定位与建图或 HopeNet-Lite。根据第 5.1 节的研究结果，我们选择 *BkPLLWrite*（末级缓存） 攻击作为攻击者工作负载，因为它在降低同步定位与建图任务性能方面最为有效。

图3展示了真实值以及生成的三条轨迹。首先，在*Solo*模式下，生成的轨迹几乎与真实值完全重合，表明 OV²SLAM 的准确率很高，ATE 中位数为 0.03 米，也就是说观测轨迹误差仅约为 3 厘米。然而，在 +深度神经网络模式下，加入 HopeNet-Lite 深度神经网络任务显著影响了 OV² SLAM 的准确率，ATE 中位数增至 0.8 米以上（约为 Solo 模式下的 27 倍）。这是因为 OV²SLAM 与 HopeNet-Lite 深度神经网络任务会争用共享硬件资源，尤其是共享的动态随机存取存储器；CPU（执行 SLAM）和集成 GPU（执行深度神经网络）都会使用该存储器。最后，在+深度神经网络*&DoS*模式下，当 HopeNet-Lite 与 DoS 攻击同时运行时，OV²SLAM 的性能急剧下降，导致其无法生成完整的



(a) XY平面中的轨迹　　(b) X、Y和Z位置随时间的变化

图3：OV²SLAM与两个*BkPLLWrite*（末级缓存） DoS攻击者及一个基于GPU的DNN应用同时运行时的轨迹。

轨迹。这是由于 OV²SLAM 无法跟上输入数据，导致大量输入相机帧被丢弃。我们的分析显示，在+深度神经网络&*DoS*模式下，仅处理了约 26% 的输入图像帧，而*Solo* 或+深度神经网络 模式下的处理比例均超过 97%。再次说明，共同调度任务之间对共享硬件资源的争用会导致性能下降，尤其是共享动态随机存取存储器（由深度神经网络任务引发的争用）和 LLC bank（由 DoS 攻击引发的争用）。

总之，尽管通过划分CPU和缓存空间来保护OV²SLAM任务，但由于共享硬件资源的争用，协同调度HopeNet-Lite DNN和/或BkPLLWrite（末级缓存）DoS攻击会显著影响OV²SLAM的准确性。

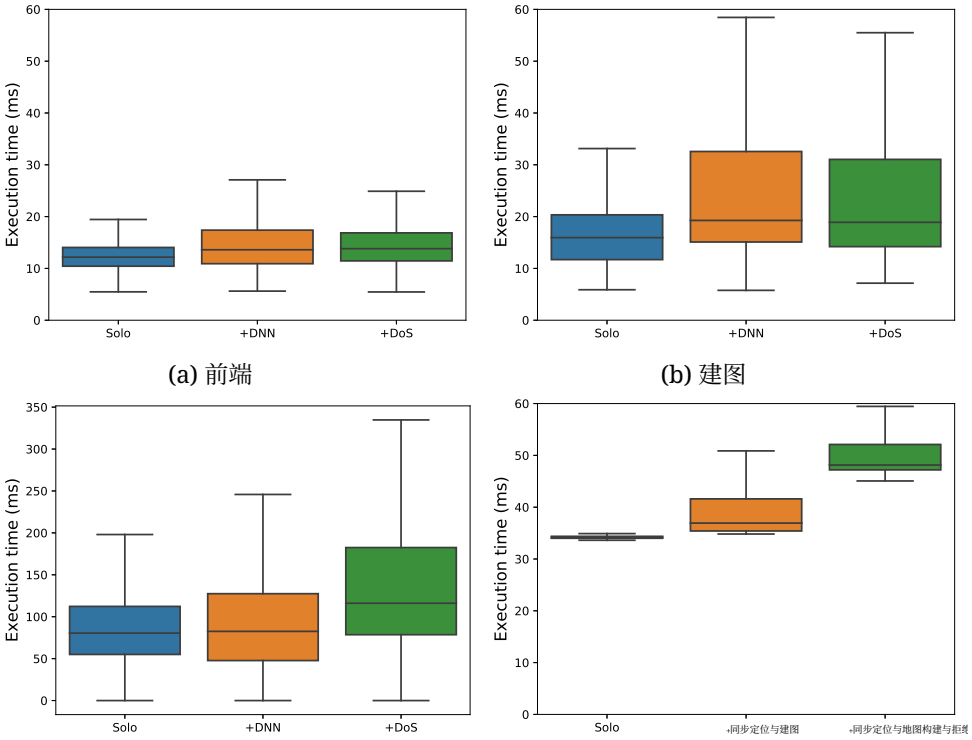
5.3 OV²SLAM和HopeNet-Lite的运行时分析

为进一步研究共享资源争用对 AR-HUD 应用的影响，我们对 OV²SLAM 任务的三个 CPU 线程 —— 前端、建图 和 状态优化线程——以及基于深度神经网络的头部姿态估计器 HopeNet-Lite，进行了详细的执行时间分析。

对于 OV²SLAM（视觉SLAM），我们测量并记录任务中各线程执行其主要计算循环时的执行时间（例如，前端 接收到新的输入帧时，建图 接收到新的关键帧时，等等）。我们在以下三种场景下重新进行实验：单独运行，SLAM 任务及其三个线程单独运行；+*DoS*，将 *BkPLLWrite*（末级缓存）DoS 攻击任务与 SLAM 一同调度；以及 +深度神经网络，将 HopeNet-Lite 任务与 SLAM 一同调度。

对于 HopeNet-Lite（头部姿态估计），我们测量其在 1000 个输入帧上的推理时间。随后，我们计算所有任务和线程的执行时间分布。以确定它们是否会因共同运行任务干扰而出现执行延迟。我们在以下三种场景中重新进行实验：*Solo* 单独运行时， +*SLAM* 与 OV²SLAM任务共同运行时，以及+*SLAM&DoS* 与 OV²SLAM 任务和*BkPLLWrite*（末级缓存）DoS 攻击共同运行时。

图4展示了所有实时任务（OV²SLAM 和 HopeNet-Lite DNN）及其线程在每种测试场景下的执行时间分布。首先，对于 OV²SLAM 任务，我们观察到，当存在共运行任务（HopeNet-Lite DNN 或 DoS 攻击）时，三个线程的执行时间均有所增加。对于 *FrontEnd* 线程（插图(a)），无论哪种共运行任务存在，其执行时间增加都相对较小。对于 *Mapping*线程（插图(b)），DNN 和 DoS 攻击者都会显著增加建图线程的执行时间。另一方面，对于 *State Optimization* 线程（插图(c)），DoS 攻击者在增加其执行时间方面比 DNN 共运行任务更为有效。请注意，DNN 和 DoS 任务分别在不同的共享硬件资源上造成争用，即 DRAM（带宽）和 LLC（bank）。因此，我们可以推断：（1）Front End 线程对这两种共享资源都不太敏感；（2）Mapping 线程同时对 DRAM 和 LLC 敏感；（3）State Optimization 线程对 LLC 更为敏感。这些由争用导致的执行时间增加，尤其是 Mapping 和 State Optimization 线程的执行时间增加，会进一步导致我们在存在 DNN 或 DoS 攻击任务时观察到的 ATE 增加。需要注意的是，与仅存在 DNN 共运行任务时相比，存在 DoS 攻击者时观察到的 ATE 损失要高得多（相对于 Solo，ATE 增加了 55 倍，而不是 27 倍），这表明 State Optimization 对 OV²SLAM 算法的整体准确率至关重要，这与先前的研究结果 [23]一致。



(a) 前端 (b) 建图

(c) 状态优化 (d) 头部姿态估计

图 4：AR-HUD中所有实时任务的执行时间分布：（a）-（c）OV^{2}同步定位与建图；（d）HopeNet-Lite

其次，插图(d)展示了 HopeNet-Lite DNN 任务的执行时间分布。请注意，当它与 SLAM（运行在不同的核心/图形处理器上）一起运行时，其平均推理时间从 ∼34 ms（单独运行）增加到∼37ms（+SLAM）。然而，当进一步加入 DoS 攻击（+SLAM&DoS）后，平均推理时间再次增加至 ∼48ms。由此我们可以推断，DNN 任务本身同时对 SLAM 任务和 DoS 攻击任务敏感，尤其是对其 Mapping 线程敏感，因为该线程可能会与 DNN 争用 DRAM 带宽；而 DoS 攻击任务则会在特定的 LLC bank 上造成争用。

6 缓解共享资源争用

本节将介绍一种缓解方案，用于在存在干扰任务（即 DoS 攻击）的情况下保护实时AR-HUD应用程序。

6.1 RT-Gang++

在本研究中，我们利用 RT-Gang 调度框架。RT-Gang 是一种实时组调度器，以_Linux 内核 [2]中的 SCHED FIFO 实时调度器扩展形式实现。RT-Gang 支持一种简单的实时组调度策略，允许所有处理器核心上同一时间仅运行一个并行实时组。此外，RT-Gang 支持对尽力而为任务进行内存带宽节流，以保护当前正在运行的实时 Gang 任务。换言之，每当任意处理器核心上运行实时 Gang 任务时，RT-Gang 都会对执行尽力而为任务的处理器核心进行节流在系统中。另一方面，如果系统中没有调度实时组，那么尽力而为任务即可完全访问内存带宽。

当我们尝试应用 RT-Gang 来缓解 ARM工业挑战赛中 AR-HUD应用所面临的共享资源争用问题时，遇到了以下挑战。首先，原始 RT-Gang 一次仅支持单个 Gang 任务。当我们在原始 RT-Gang 中将实时任务作为独立的 Gang 任务进行调度时，由于这些任务未实现完全并行化，无法充分利用所有可用的处理器核心和图形处理器，因此无法满足必要的实时要求。其次，即使我们将这些任务组合起来形成“虚拟组”任务 [1]以提高资源利用率，RT-Gang 也没有提供任何用于缓解虚拟组内部实时任务之间争用的机制。这意味着，当优先级较低、运行在集成 GPU 上的 DNN 实时任务与其协同调度时，无法减小其对优先级较高的 OV^{2}同步定位与建图任务造成的负面性能影响。最后，当我们将缓存组感知型拒绝服务攻击 [7] 部署为尽力而为的干扰任务时，RT-Gang 的内存带宽节流能力在保护实时任务方面变得无效，因为这些干扰任务的攻击目标是末级缓存组争用，并不消耗任何内存带宽。

为应对这些挑战，我们对原始 RT-Gang 做出了三项扩展：(1) 分区式实时 Gang 调度能力；(2) 集成 GPU 带宽限制；(3) LLC 带宽限制。由此得到的系统称为*RT-Gang++*。

6.2 分区组调度

基线 RT-Gang 的一项主要特性是，在多核 CPU 的所有核心上，同一时间只允许一个实时 Gang 任务运行。虽然这种设计能够防止实时任务之间发生共享资源争用，有利于提高可预测性，但从可扩展性的角度来看，这也是一种限制，因为并非所有任务都能从大量核心中受益，而 CPU 中的核心数量也在不断增加。RT-Gang 通过支持所谓的“虚拟 Gang”在一定程度上缓解了这一问题。虚拟 Gang 是由多个周期和优先级均相同的实时任务组成的集合，这些任务整体上的行为类似于一个 Gang 任务；但当任一实时任务的优先级或周期与其他任务不同时，便无法使用该机制。此外，许多现代多核 CPU 通常由多个集群组成，每个集群可能拥有与其他集群不共享的不同计算资源和内存资源。例如，许多 ARM 多核 CPU 采用 big.LITTLE 架构，其中一个集群由高性能“大”核心组成，另一个集群由高能效“小”核心组成。这些集群不仅包含不同类型的 CPU 核心，而且通常还拥有集群私有共享资源，这些资源不会跨集群共享，从而降低了严格遵循 RT-Gang“一次只能运行一个 Gang 任务”策略的必要性。

在 RT-Gang++ 中，我们支持多个分区，其中每个分区由一组静态确定的处理器核心组成。随后，每个分区分别采用一次仅运行一个 Gang 任务的策略，而不是全局采用该策略。这意味着，只要分配到不同的分区，多个具有不同优先级和/或周期的 Gang 任务就可以同时运行。对于表 2 中的 AR-HUD 案例研究，我们将 OV^{2}SLAM 和 EuRoC 回放任务分配为一个虚拟组任务，并将其分配到一个 Gang 分区中，该分区由核心 0、1 和 2 组成；同时，我们将深度神经网络任务（头部姿态估计）分配到另一个 Gang 分区中，该分区由核心 3 和集成 GPU 组成。通过这种方式，系统可以同时运行两个具有不同优先级和周期的活动 Gang 任务。

6.3 集成 GPU 内存带宽节流

然而，当多个实时任务共同运行时，它们不可避免地会争用共享资源。正如第 5.2 节所观察到的那样，尤其是协同调度深度神经网络任务会对更关键的 OV

[2]同步定位与建图任务的准确率。遗憾的是，基线 RT-Gang 未提供任何方法来解决两个共同调度的实时任务之间的争用。在本研究中，我们利用了所用评估平台的硬件级 GPU 带宽节流功能。具体而言，Jetson Nano 平台中的 Tegra X1 SoC 在其内存控制器上支持多种 QoS 特性，其中一项是对访问内存控制器的部分硬件组件进行硬件级节流 [30]。内存控制器的节流功能提供了 32 个可编程节流级别，可用于对集成 GPU 进行节流。我们利用这一功能对深度神经网络任务进行节流，因为该任务通过 GPU 进行的内存访问会影响 OV^{2}同步定位与建图任务的性能。

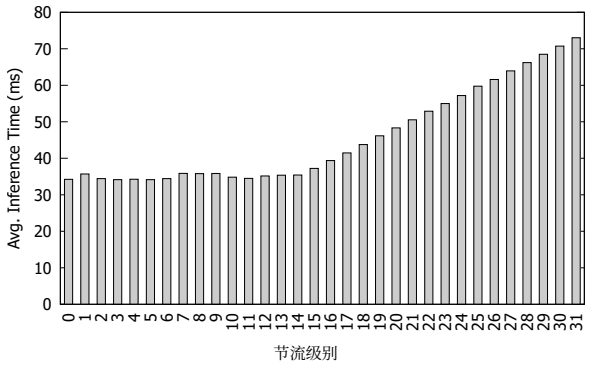


图 5：GPU 节流对 HopeNet-Lite DNN 的影响。

图 5 展示了 HopeNet-Lite 在单独运行、与其他任务隔离时，在 32 个节流级别下的平均推理延迟。请注意，在节流级别达到约 15 之前，GPU 节流不会影响深度神经网络任务的性能，这表明在此之前带宽足够。此后，更强的节流会影响深度神经网络的性能。当节流级别为最高级别 31 时，平均推理延迟增加到 ∼73 毫秒，约为未节流时的两倍。请注意，随着对 GPU（后者使用 GPU）进行更强的节流，SLAM 任务的准确率与深度神经网络任务的延迟之间存在权衡：更强的节流有助于提高 SLAM 任务的准确率（降低 ATE），但会增加深度神经网络任务的延迟。因此，需要为目标应用找到一个“最佳平衡点”。在本研究中，我们通过实验选择了级别 20，因为这是仍能使深度神经网络推理任务达到 20 赫兹实时性能的最高节流级别。不过，如果相比于以 20 赫兹处理深度神经网络任务，实现 SLAM 任务的最高准确率更为重要，也可以选择更激进的节流级别（例如级别 31）。在我们的测试中，将 GPU 节流级别设为 31，可使 SLAM 任务实现近乎完全的隔离，但代价是深度神经网络任务的延迟翻倍。

6.4 末级缓存带宽节流

对于我们的案例研究应用，OV^{2}SLAM 和 DNN 实时任务都必须受到保护，避免受到以尽力而为方式调度的共同调度干扰任务的干扰（即，调度到任何未执行 RT 任务的处理器核心上）。如第 5.1 节所述，当 DoS 攻击者被用作干扰任务时，即使由于共享资源争用，DoS 攻击者无法抢占 SLAM 任务，OV^{2}SLAM 算法的性能仍会显著下降。具体而言，我们发现，缓存组感知型拒绝服务攻击 [7] 对 SLAM 任务的准确率有特别显著的负面影响。然而遗憾的是，基线 RT-Gang 的内存带宽节流能力无法防御缓存组拒绝服务攻击，因为该攻击会产生末级缓存命中，而不会消耗任何内存带宽。

在 RT-Gang++ 中，我们通过利用 CPU 核心的 L1-D 缓存未命中性能计数器（L1D_CACHE_REFILL），增加了对末级缓存带宽节流能力的支持

用于跟踪并限制尽力而为型 DoS 攻击任务所使用的末级缓存（L2）带宽。需要注意的是，该节流实现基于 MemGuard [43]，并且在启用末级缓存节流且调节间隔为 1ms 时，我们使用通过实验确定的 100 MB/s 末级缓存带宽阈值。

7 评估结果

在本节中，我们在两个常用的嵌入式多核平台 Jetson Nano 和 Raspberry Pi 4 上评估 RT-Gang++ 的性能。

7.1 Jetson Nano

基本实验设置与第 4 节所述相同：即在核心 0 和 1 上执行 SLAM 任务，在核心 2 上执行 EuRoC 数据集回放任务，并在核心 3 和集成 GPU 上执行深度神经网络任务。

为进行评估，我们重复第 5.2 节中的实验，但使用 RT-Gang++。回顾一下，在该实验中，我们将 BkPLLWrite（末级缓存）DoS攻击者作为攻击者共同调度到所有 CPU 核心上，以引发共享硬件资源上的争用，具体而言是在单个 LLC bank 上产生争用。因此，我们希望了解在 DoS 攻击者存在的情况下，RT-Gang++ 能在多大程度上保护 SLAM 和深度神经网络任务的性能。

7.1.1 结果

图6展示了 OV^{2}SLAM 任务生成的轨迹，以及轨迹随时间变化的 X、Y 和 Z 位置。请注意，*Co-run* 表示未启用 RT-Gang++的基线配置，而 *RT-Gang++* 表示启用了 RT-Gang++ 的相同配置。如前所述，在 *Co-run*中，OV^{2}SLAM 因争用而无法生成有效轨迹。然而，在*RT-Gang++*中，OV^{2}SLAM 生成的轨迹有效，且与*Solo* 和真实值更为接近，这表明 RT-Gang++能够有效保护 SLAM 任务的性能。

配置	平均推理时间（毫秒）
Solo	34.23
Co-run	36.32
RT-Gang++	48.69

表3：深度神经网络任务的平均推理延迟

表3展示了 HopteNet-Lite 深度神经网络任务在三种不同测试配置下取得的平均推理时间。在*Solo*中，深度神经网络任务单独运行并与其他任务隔离，每个图像帧的平均推理延迟为 ∼34 毫秒。在*Co-run*中，SLAM 和 DoS攻击者与深度神经网络任务共同调度运行，平均延迟略有增加，从 ∼34毫秒升至 ∼36毫秒。另一方面，在*RT-Gang++*中，深度神经网络任务的平均延迟增加至 ∼49 毫秒。这是因为 iGPU限流限制了 iGPU 对动态随机存取存储器带宽的使用，使深度神经网络任务运行得更慢，从而有助于保护 SLAM 任务的性能，但代价是降低深度神经网络任务的性能。尽管如此，需要注意的是，深度神经网络任务仍能够达到所需的 20Hz速率（见表 2）。

Core	Solo	Co-run	RT-Gang++
0	8241	2413	956
1	8470	1304	792
2	7643	5628	1211
3	7496	156	119

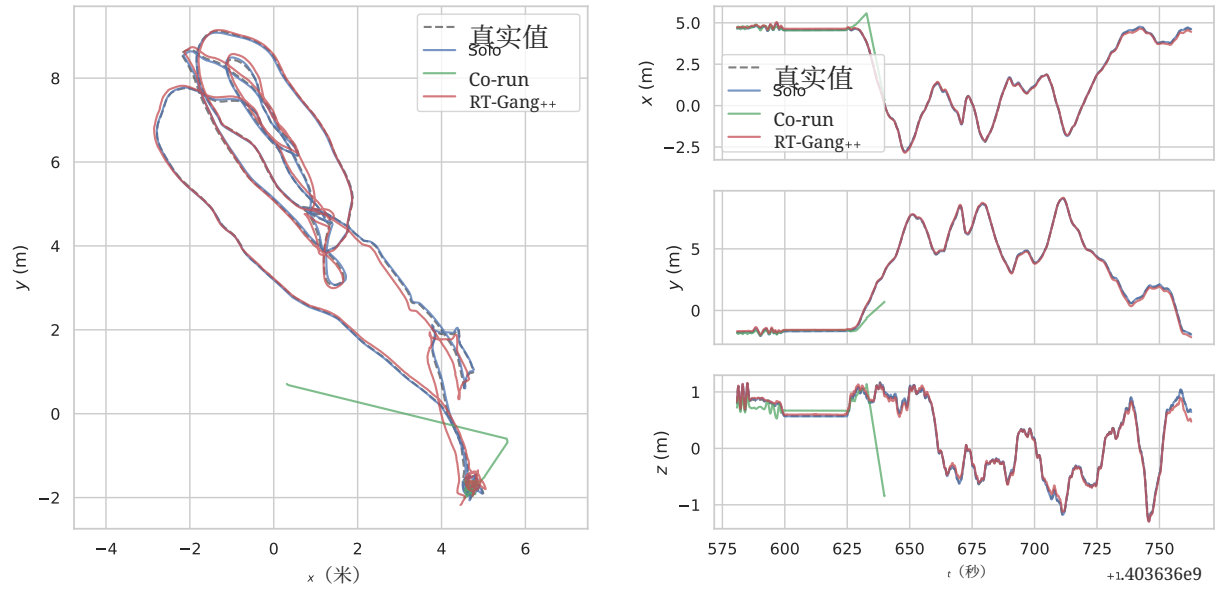
表4：所有尽力型*BkPLLWrite*（末级缓存）攻击者消耗的平均 LLC带宽（单位：MB/s）。

表4展示了尽力而为任务 DoS攻击者在每个核心上的平均末级缓存带宽：首先是在*Solo*中单独隔离运行时的带宽，其次是在没有和启用 RT-Gang++的情况下，与 SLAM 和深度神经网络实时任务共同运行时的带宽，分别对应*Co-run*和*RT-Gang++*。请注意，RT-Gang++会对尽力而为任务的末级缓存带宽进行限流，以限制 LLC组竞争。因此，*RT-Gang++* 的平均末级缓存带宽数值远低于*Co-run*或*Solo*，这符合预期。不过，需要注意的是，这些尽力而为任务 DoS攻击者仍能够使用显著高于 100MB/s 设定阈值的末级缓存带宽（该阈值根据第 6.4 节通过实验确定）。这是因为，在 RT-Gang++中，只有当平台的任一 CPU 核心上当前调度有实时任务时，末级缓存带宽限流才会动态启用。当没有活跃的实时任务时，由于实时任务可能早于截止时间完成，这种情况经常发生，末级缓存带宽限流会自动禁用，从而允许尽力而为任务 DoS攻击者在不受限流的情况下充分利用全部末级缓存带宽，直到任一实时任务再次变为活跃状态。这是

数据集	Solo	RT-Gang++（无iGPU限流）	RT-Gang++
MH01	0.03	0.36	0.11
MH02	0.04	0.47	0.07
MH03	0.08	0.24	0.20
MH04	0.15	2.86	0.29
MH05	0.12	0.80	0.23

表 5：OV^{2}SLAM 在 MachineHall场景中的 ATE 中位数（单位：米），数据来自 EuRoC 数据集。

表 5 展示了 OV^{2}SLAM 在 EuRoC 数据集的全部五个机器大厅场景中生成轨迹的 ATE 中位数。请注意，由于争用导致 OV^{2}SLAM 无法生成完整轨迹，因此不包含*Co-run* 。即使启用了 RT-Gang++，在未启用 iGPU 带宽节流时，我们仍然观察到报告的 ATE 值显著上升。另一方面，在*RT-Gang++* 中同时启用末级缓存和 iGPU 带宽节流后，我们观察到 ATE 显著改善。改善程度因场景而异。例如，MH04 场景属于难度最高的场景之一，对 iGPU 限流尤其敏感：未启用 iGPU 限流时，SLAM 的 ATE 高达 2.86 米，而启用后仅为 0.29 米。结果表明，iGPU 限流和末级缓存带宽限流对于保障 SLAM 任务的性能都很重要。请注意，这些场景要求 ATE 值低于 0.3 米 [23],，而在 RT-Gang++ 完全启用的所有情况下，我们都能满足这一要求。



(a) X-Y 平面轨迹

(b) X、Y 和 Z 随时间变化的位置。

图6：RT-Gang++ 对 Jetson Nano 上 OV²同步定位与建图性能的影响。

7.2 Raspberry Pi 4

为了展示 RT-Gang++的通用性，我们还在 Raspberry Pi 4 平台上对其进行了评估。

7.2.1 硬件和软件设置

Raspberry Pi 4 配备博通 BCM2711 片上系统，其中包括采用 1MB 共享二级缓存的四核 ARM Cortex-A72 CPU、VideoCore 6 GPU 和 4GB LPDDR4 SDRAM。需要注意的是，Raspberry Pi 4 的 Cortex-A72 CPU 核心比 Jetson Nano 的 Cortex-A57 核心更先进、性能更强。不过，Pi 4 的 VideoCore 6 GPU 弱于 Nano 的 128 核 Maxwell GPU，并且缺乏用于 GPU 卸载的软件框架支持。表 6 展示了 Raspberry Pi 4 的硬件规格。

平台	Raspberry Pi 4
SoC	BCM2711
CPU	4 个 Cortex-A72，频率为 1.5 赫兹
GPU	VideoCore 6（未使用）
共享末级缓存（二级缓存）	1MB（16 路）
内存（峰值带宽）	4GB LPDDR4（25.6 GB/s）

表 6：Raspberry Pi 4 硬件规格。

基本应用设置与表 2 中的 Jetson Nano 相同，但由于 Pi 4 的 GPU 缺乏深度神经网络软件支持，HopeNet-Lite DNN任务完全在 CPU 上运行。

在软件方面，Pi 4 运行搭载 Linux 内核 5.15 的 Raspberry Pi OS，并应用了 PALLOC 补丁 [42]以支持 LLC 空间分区，这与 Jetson Nano 平台的情况相同。我们采用了与 Jetson Nano 相同的 2/2 LLC 分区设置，不过由于 Pi 4 的 L2 缓存大小是 Nano 的一半，因此每个缓存分区的大小也减半。

最后，我们将 RT-Gang++ 移植到 Pi 4 的 Linux 内核上，其中包括分区式协同调度和 LLC 带宽限流功能。不过，由于前文所述未使用集成 GPU，因此未实现 iGPU 限流。

7.2.2 结果

首先，我们重复第 5.1 节中的实验，以了解各种 DoS 攻击对 Pi 4 平台的影响。

图 7 展示了结果。与图 2 中 Jetson Nano 上的结果类似，由于争用增加，所有 DoS 攻击都会增加同步定位与建图生成的轨迹中的绝对轨迹误差分数。还需注意的是，BkPLLWrite（末级缓存）再次成为最有效的 DoS 攻击，与 Jetson Nano 平台上的情况相同，但其 ATE 中位数增幅为 1.3，略小于我们在 Jetson Nano 平台上观察到的 1.9。由此可见，Pi 4 的 Cortex-A72 中央处理器核心同样容易受到 LLC bank 争用的影响，就像 Nano 的 Cortex-A57 一样。因此，在后续实验中，我们再次使用 BkPLLWrite（末级缓存）作为 DoS 攻击者任务。

接下来，我们重复第5.2节中的实验，以评估在存在 DoS攻击和HopeNet-Lite DNN任务的情况下，RT-Gang++ 保护OV²同步定位与建图性能的能力。

图 8 展示了在 Pi 4 平台上有无 RT-Gang++ 时，OV² SLAM 随时间生成的轨迹和 XYZ 位置。首先需要注意的是，与 Jetson Nano 平台上的结果（图 6）一样，我们发现共同调度的拒绝服务攻击和深度神经网络任务产生的争用确实会影响 SLAM 的性能。然而，与 Jetson Nano 上的情况不同，尽管 ATE 显著增加，SLAM 任务仍能够生成完整的轨迹；在 *Co-run* 情况下，生成的轨迹与真实值或 *Solo* 情况下的轨迹相差甚远。这主要是因为 Pi 4 上的深度神经网络任务运行在 CPU 核心上，而不是图形处理器上。在 Pi 4 上，运行于 CPU 核心的深度神经网络任务不会产生同样多的 DRAM带宽争用

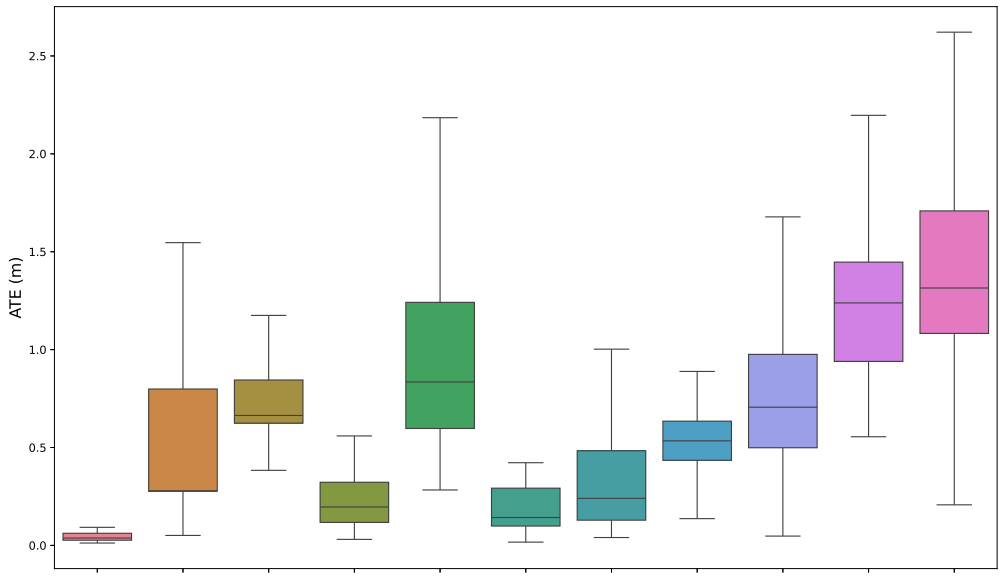
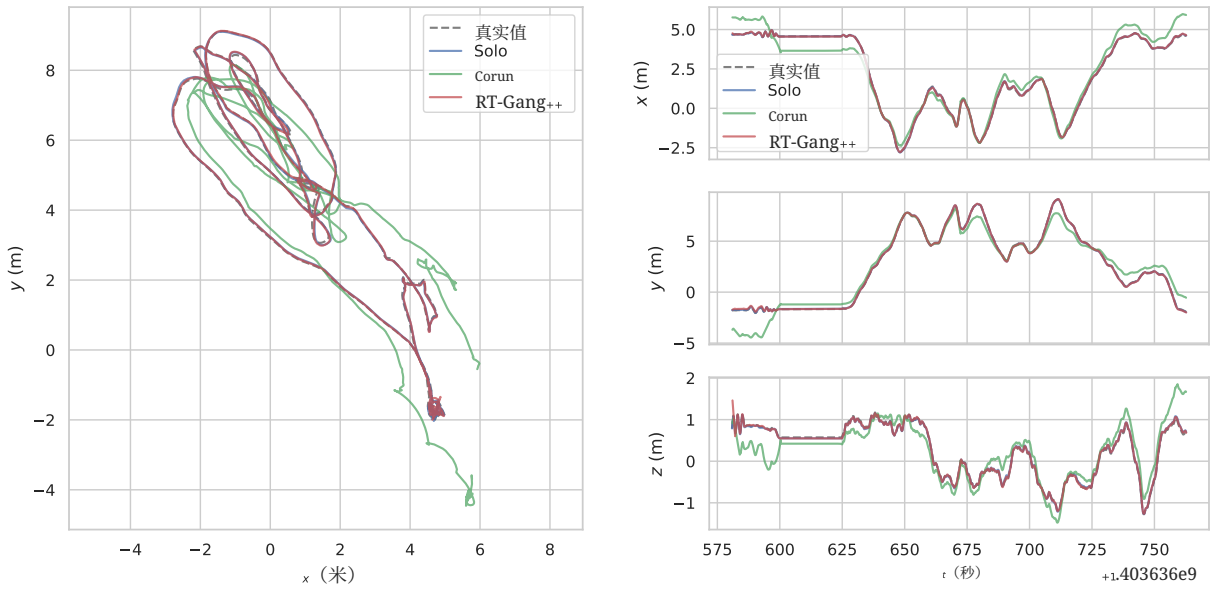


图 7：DoS 攻击对 Pi 4 上 OV^{2}同步定位与建图生成轨迹的绝对轨迹误差（ATE）的影响。



（a）X-Y 平面轨迹

（b）X、Y 和 Z 随时间变化的位置。

图 8：RT-Gang++对 Pi 4 上 OV^{2}同步定位与建图（SLAM）性能的影响。

相比之下，如果在图形处理器上运行，则会产生更多争用。其次，RT-Gang++ 能够有效缓解深度神经网络任务和拒绝服务攻击产生的争用，并保护 SLAM 任务的性能。具体而言，RT-Gang++ 将 ATE 中位数降低至仅 ∼0.05 米，非常接近 *Solo* 情况下的 ∼0.04 米 ATE 中位数。

8 相关工作

同时定位与建图（SLAM）算法是许多机器人应用（包括高级驾驶辅助系统和自动驾驶系统）的核心，因为它们被用于用于确定自车相对于周围环境的位置和姿态。值得注意的 SLAM 算法包括 LSD-SLAM 算法 [12],ORB-SLAM 算法的多次迭代 [28], [29],[11],以及本文所采用的 OV^{2}SLAM 算法 [15], 。这些算法会检测相机图像中的特征，并跟踪这些特征以确定其在世界坐标中的位置。最近，Li 等人观察到，SLAM 算法的性能可能对执行时序延迟较为敏感，并在 SLAM 中提出了一种自适应策略，以最大限度地降低性能下降 [23]。相比之下，我们的

工作提出了一种系统级解决方案，无需更改 SLAM 算法，并对资源争用进行了深入的微架构分析。

微架构拒绝服务攻击是专门设计用于引发高度资源争用的软件攻击。研究人员已对针对多种共享资源的拒绝服务攻击进行了研究和评估。Moscibroda 等人提出了一种针对共享 DRAM 控制器中 FR-FCFS 调度算法的拒绝服务攻击 [31] [27]。Keramidas 等人展示了针对共享末级缓存空间的拒绝服务攻击，并为应对此类攻击，提出了一种缓存替换策略，使攻击者只能访问较少的末级缓存空间 [20]。Woo 等人在模拟环境中研究了针对总线带宽和共享缓存空间的拒绝服务攻击 [39]。Valsan 等人和 Bechtel 等人证明，拒绝服务攻击可以针对 LLC 内部硬件结构 [9], [8], [38]。基于 [9], , Iorga 等人提出了一种用于测试拒绝服务攻击的统计方法 [19]。Li 等人采用机器学习来更好地优化拒绝服务攻击者，从而导致最坏情况执行时间减慢 >400X [24]。研究人员也研究了基于 GPU 的拒绝服务攻击。Yandrofski 等人通过生成各种对抗性程序，系统地研究了独立 Nvidia GPU 上的共享资源争用 [41]。Bechtel 等人针对 Intel 集成 GPU 实施了拒绝服务攻击，因为它们也会访问 LLC。

实时系统研究领域投入了大量精力来解决多核系统中的共享资源争用问题。共享资源的分区，尤其是共享缓存 [26], [21], [22], [14], [40],[32] 和 DRAM 存储体 [22], [14], , 已经得到了广泛研究。带宽节流 [43], [40], [37], [34], [35], [45],[13] 是另一种广受欢迎的方法。MemGuard [43]使用每核硬件性能计数器来限制每个核心的带宽使用量，这已成为后续许多研究中的标准节流技术。这些机制用于实现更严格的多核最坏情况时序分析。关于多核时序分析的全面综述可参见 [25]。最近，Intel 和 ARM 也引入了用于共享资源分区和节流的硬件支持 [18], [5], , 但其为实时系统提供隔离的有效性仍然不足 [44], [36], [7]。例如，有报道称，即使在最高节流级别下，Intel RDT 仍无法保护关键实时任务，因为受到节流的核心仍能够产生大量流量，足以在最坏情况下导致实时任务的性能下降 80% [36]。尽管如此，这些硬件能力彼此正交，并且可以轻松集成到我们的框架中。

在大多数此类研究中，核心会在实时核心与尽力而为核心之间进行分区。例如，Saeed等人提出了一种基于内存利用率的动态节流系统，通过限制其他尽力而为核心的内存带宽使用量，来保护单个实时核心 [34], [35]。Seals等人还提出了一种动态节流系统，该系统也能够限制集成 GPU 的内存带宽，从而保护单个实时核心 [13]。然而，这种方法可能会严重降低计算资源的利用率。例如，在 AR-HUD 工业挑战赛问题中，一个实时核心并不足够，因为该应用由一个多线程 CPU 任务和一个 GPU 任务组成，二者都需要实时保证。RT-Gang [2], [1] 提供了更加灵活的调度潜力，因为所有核心都可以同时用于实时任务和尽力而为任务。这是因为，只有当多核系统中的任意核心数上运行实时任务时，操作系统才会自动限制尽力而为任务。然而，它无法为不同核心上的实时任务与集成 GPU 之间提供相互保护。在本文中，我们采用 RT-Gang，并通过增加集成 GPU 节流、CPU缓存带宽节流和分区式协同调度来解决其局限性，从而在真实的异构多核平台上，面对负载完全饱和的干扰任务时确保性能，并成功整合 ARM工业挑战赛应用。

9 结论

本文介绍了针对 ARM 在 2022 [3] 中提出的工业挑战赛问题的解决方案。我们系统地分析了在异构多核平台上，资源争用对工业挑战赛中的增强现实抬头显示器（AR-HUD）案例研究应用的影响。我们使用微体系结构拒绝服务（DoS）攻击作为挑战赛的干扰任务，结果表明，此类干扰任务会显著影响 AR-HUD 应用的延迟和准确率。尽管通过对 AR-HUD 应用采用缓存分区和实时调度，尽最大努力减轻其影响，但这仍可能导致估计轨迹与真实值之间出现显著偏差。为解决这一问题，我们提出了 RT-Gang++，通过结合末级缓存和 iGPU 带宽节流，减轻来自相应资源的共享资源争用。部署 RT-Gang++ 后，我们能够有效保护关键 SLAM 任务的性能，使其达到接近 Solo 情况的性能，而无需对系统进行过度配置。

未来的工作中，我们计划在性能优于 Jetson Nano 的平台上开展类似的案例研究，例如 Jetson Xavier 或 Jetson Orin 系列嵌入式平台，以评估这些平台对共享资源争用的易受影响程度，以及所提出的 RT-Gang++ 框架的通用性。

致谢

本研究部分由美国国家科学基金会 CNS-1815959、CPS-2038923 项目以及美国国家安全局安全科学倡议合同编号 H98230-18-D-0009 资助。

参考文献

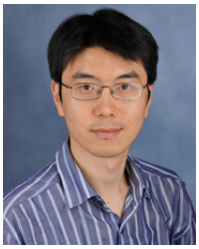
[1] W. Ali、R. Pellizzoni 和 H. Yun。并行实时任务的虚拟组调度。载于*DATE*，第270–275页。IEEE，2021年。[2] W. Ali 和 H. Yun。RT-Gang：面向安全关键系统的实时组调度框架。载于*RTAS*，2019年。[3] M. Andreozzi、G. Gabrielli、B. Venu 和 G. Travaglini。工业挑战赛2022：基于Arm的高性能实时案例研究。载于*ECRTS*，2022年。[4] SOAFEE。https://www.soafee.io/。[5] ARM。*Arm*架构参考手册补充文件：内存系统资源分区与监控（*MPAM*），*DDI:0598B.b*，2020年。[6] M. Bechtel 和 H. Yun。多核实时系统中共享缓存上的内存感知型拒绝服务攻击。《计算机汇刊》，2021年。[7] M. Bechtel 和 H. Yun。多核ARM处理器上的缓存存储体感知型拒绝服务攻击。载于*RTAS*，2023年。

[8] M. G. Bechtel、E. McEllhiney、M. Kim 和 H. Yun。DeepPicar：低成本深度神经网络自主车。载于RTCSA，2018年。[9] M. G. Bechtel 和 H. Yun。多核共享缓存上的拒绝服务攻击：分析与防护。载于RTAS，2019年。[10] M. Burri、J. Nikolic、P. Gohl、T. Schneider、J. Rehder、S. Omari、M. W. Achtelik 和 R. Siegwart。EuRoC微型空中飞行器数据集。《国际机器人研究杂志》，2016年。[11] C. Campos、R. Elvira、J. J. G. Rodr´ıguez、J. M. Montiel 和 J. D. Tard´os。ORB-SLAM3：用于视觉、视觉惯性和多地图SLAM的准确开源库。《IEEE机器人学汇刊》，2021年。[12] J. Engel、T. Sch¨ops 和 D. Cremers。LSD-SLAM：大规模直接单目SLAM。载于ECCV，2014年。[13] H. Y. Eric Seals、Michael Bechtel。Bandwatch：面向异构多核的系统级内存带宽调节系统。载于RTCSA，2023年。[14] F. Farshchi、P. K. Valsan、R. Mancuso 和 H. Yun。确定性内存抽象及其多核系统架构支持。载于ECRTS，2018年。[15] M. Ferrera、A. Eudes、J. Moras、M. Sanfourche 和 G. Le Besnerais。OV²SLAM：面向实时应用的全在线且通用的视觉SLAM。《IEEE机器人与自动化快报》，2021年。[16] HopeNet-Lite。https://github.com/OverEuro/deep-head-pose-lite。[17] rosbag2。https://github.com/ros2/rosbag2。[18] Intel。Intel®资源调度器技术（Intel® RDT）框架。
h
t
t
p
s://
w
w
w.
i
n
t
e
l.
c
o
m/
c
o
n
t
e
n
t/www/us/en/architecture-and-technology/resource-director-technology.html。[19] D. Iorga、T. Sorensen、J. Wickerson 和 A. F. Donaldson。缓慢而稳定：多核干扰的测量与调节。载于RTAS，2020年。[20] G. Keramidas、P. Petoumenos、S. Kaxiras、A. Antonopoulos 和 D. Serpanos。防止共享CMP缓存中的拒绝服务攻击。载于SAMOS，2006年。[21] H. Kim、A. Kandhalu 和 R. Rajkumar。多核实时系统中实用的操作系统级缓存管理的协调方法。载于ECRTS，2013年。[22] N. Kim、B. C. Ward、M. Chisholm、J. H. Anderson 和 F. D. Smith。通过结合硬件管理与混合关键性资源配置，攻击多核一出M问题。《实时系统》，2017年。[23] A. Li、H. Liu、J. Wang 和 N. Zhang。从时序变化到性能下降：理解并缓解软件执行时序对SLAM的影响。载于IROS，2022年。[24] A. Li、M. Sudvarg、H. Liu、Z. Yu、C. Gill 和 N. Zhang。PolyRhythm：用于时序干扰的多通道攻击模板自适应调节。载于RTSS，2022年。[25] C. Maiza、H. Rihani、J. M. Rivas、J. Goossens、S. Altmeyer 和 R. I. Davis。多核实时系统时序验证技术综述。《ACM计算调查》（CSUR），52(38)，2019年。[26] R. Mancuso、R. Dudko、E. Betti、M. Cesati、M. Caccamo 和 R. Pellizzoni。面向多核架构的实时缓存管理框架。载于RTAS，2013年。[27] T. Moscibroda 和 O. Mutlu。内存性能攻击：多核系统中的内存服务拒绝。载于USENIX安全研讨会，2007年。[28] R. Mur-Artal、J. M. M. Montiel 和 J. D. Tardos。ORB-SLAM：通用且准确的单目SLAM系统。《IEEE机器人学汇刊》，2015年。[29] R. Mur-Artal 和 J. D. Tard´os。ORB-SLAM2：用于单目、双目和RGB-D相机的开源SLAM系统。《IEEE机器人学汇刊》，2017年。[30] NVIDIA。Tegra X1移动处理器技术参考手册（修订版1.3p），2019年。[31] S. Rixner、W. J. Dally、U. J. Kapasi、P. Mattson 和 J. Owens。内存访问调度。载于ACM SIGARCH计算机体系结构新闻，2000年。[32] S. Roozkhosh 和 R. Mancuso。中间的可编程逻辑潜力：缓存漂白。载于RTAS，2020年。[33] N. Ruiz、E. Chong 和 J. M. Rehg。不使用关键点的细粒度头部姿态估计。载于CVPR，2018年。[34] A. Saeed、D. Dasari、D. Ziegenbein、V. Rajasekaran、F. Rehm、M. Pressler、A. Hamann、D. Mueller-Gritschneder、A. Gerstlauer 和 U. Schlichtmann。基于内存利用率的动态带宽

多核平台中用于时间隔离的宽度调节。载于*RTAS*，2022年。[35] A. Saeed、D. Hoornaert、D. Dasari、D. Ziegenbein、D. Mueller-Gritschneder、U. Schlichtmann、A. Gerstlauer 和 R. Mancuso。基于内存延迟分布的多处理器片上系统时间隔离调节。载于实时系统欧洲微型会议（*ECRTS*）。Schloss Dagstuhl-Leibniz 信息学中心，2023年。[36] P. Sohal、M. Bechtel、R. Mancuso、H. Yun 和 O. Krieger。深入探究 Intel 资源调度器技术（RDT）。载于*RTNS*，第127–139页，2022年。[37] P. Sohal、R. Tabish、U. Drepper 和 R. Mancuso。E-WarP：用于内存带宽分析与管理的系统级框架。载于*RTSS*，2020年。[38] P. K. Valsan、H. Yun 和 F. Farshchi。驾驭非阻塞缓存以提升多核实时系统中的隔离能力。载于*RTAS*，2016年。[39] D. H. Woo 和 H. Lee。分析片上多处理器因资源拒绝服务攻击而导致的性能脆弱性。载于*CMP-MSI*，2007年。[40] M. Xu、L. T. X. Phan、H.-Y. Choi、Y. Lin、H. Li、C. Lu 和 I. Lee。多核实时系统的整体资源分配。载于*RTAS*，2019年。[41] T. Yandrofski、J. Chen、N. Otterness、J. H. Anderson 和 F. Smith。在 NVIDIA 图形处理器上树立强大的敌人。载于*RTSS*，2022年。[42] H. Yun、R. Mancuso、Z.-P. Wu 和 R. Pellizzoni。PALLOC：面向 DRAM 存储体的内存分配器，用于多核平台上的性能隔离。载于*RTAS*，2014年。[43] H. Yun、G. Yao、R. Pellizzoni、M. Caccamo 和 L. Sha。MemGuard：用于高效性能隔离的多核平台内存带宽预留系统。载于*RTAS*，2013年。[44] M. Zini、D. Casini 和 A. Biondi。从时间可预测性的角度分析 ARM 的内存分区与监控。*IEEE*《计算机汇刊》，72(1):168–182，2022年。[45] A. Zuepke、A. Bastoni、W. Chen、M. Caccamo 和 R. Mancuso。MemPol：在处理器核心之外监管核心内存带宽。载于*RTAS*，2023年。



Michael Bechtel 于2017年获得堪萨斯大学计算机科学理学学士学位，并于2023年获得堪萨斯大学计算机科学博士学位。他的研究兴趣包括实时嵌入式系统、计算机体系结构和信息物理系统。他的研究成果发表于RTAS等顶尖嵌入式实时系统会议，并获得了RTAS'19杰出论文奖。他目前是Garmin的软件工程师。



Heechul Yun 是堪萨斯大学电气工程与计算机科学系副教授。他的研究兴趣包括操作系统、计算机体系结构和实时嵌入式系统，尤其关注解决安全关键型信息物理系统（例如自动驾驶汽车和无人机）中的实时性、安全和安全性相关问题。他的研究成果发表于RTAS、ECRTS和《计算机汇刊》等顶尖嵌入式实时系统领域刊物；并曾获得多项著名论文奖项（RTSS'20最佳论文奖、RTAS'19杰出论文奖、RTAS'16最佳论文奖、2016年IEEE《计算机汇刊》年度编辑精选奖）。他于2013年获得伊利诺伊大学厄巴纳-香槟分校计算机科学博士学位。在攻读博士学位之前，他曾在三星电子担任高级软件工程师。