華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

课程报告

Common Counters: Compressed Encryption Counters for Secure GPU Memory

学号	Y202102005
姓名	王道宇
类别	硕士
教师	施展 童薇
院系	研究生院

2022 年 1 月 7 日

Common Counters: GPU 内存安全加密计数器

王道宇 [1)]

[1)](武汉数字工程研究所-709 所，武汉 430074)

摘 要 基于硬件的可信执行为实现云计算的安全提供了新的机遇。然而，当前的可信执行环境仅限于传统的基于 CPU 的计算，而越来越多的关键工作负载（如机器学习）需要使用 GPU。对于 GPU 计算的安全性，最近的研究提出通过为 GPU 提供可信的执行环境，将关键的 GPU 操作与操作系统隔离开来。然而，先前 GPU 可信执行研究中缺少了针对 GPU 优化的基于硬件的内存保护技术。CPU 的最新内存保护技术使用计数器模式加密，其中每个块计数器用作生成一次性密钥。计数器的完整性由计数器树保护。本研究调查了 GPU 的硬件内存加密，并确定计数器缓存未命中是关键性能瓶颈之一。为了减少计数器缓存未命中的开销，本文提出了一种称为公共计数器的新技术，用于高效的计数器模式加密和计数器完整性保护。该技术利用了通用 GPU 应用程序的独特特性。在 GPU 应用中，大部分的应用内存都是 CPU 初始传输数据时写一次，GPU 应用程序对主要数据结构的写入次数常趋于一致。利用统一写入特性，本研究建议使用公共计数器表示来补充现有的逐块加密计数器。使用公共计数器技术几乎可以消除由计数器缓存未命中引起的性能开销，将性能开销降低到 2.9%。

关键词 可信计算；GPU 内存安全；加密计数器

1 引言

基于硬件的可信执行环境(TEE)，如 Intel Software Guard Extension (SGX)和 ARM TrustZone 提供了在不可信的远程云服务器上安全执行用户应用程序。这种硬件 TEE 通过隔离和保护用户执行环境，使其免受特权软件和物理攻击，为可信云开辟了一个新的方向。然而。当前 TEE 支持的一个关键限制是缺乏对广泛使用的基于 GPU 的计算的考虑。随着机器学习(ML)工作负载的广泛部署，GPU 计算已经成为计算系统的重要组成部分，特别是在提供基于 ML 的服务的云计算中。为了保护基本的 GPU 计算，必须将 TEE 的范围扩展到 GPU 计算。

为了解决 GPU TEE 缺乏的问题，最近的研究提出了针对 GPU 执行环境的基于硬件的加强技术。Graviton 将关键的 GPU 管理操作从 CPU 端驱动程序中分离出来，将它们隔离在 GPU 端命令处理单元中，保护 GPU 关键操作不受恶意特权软件的影响。但是 Graviton 需要改变 GPU 的命令处理子系统，才能在 GPU 内部提供这种隔离的 GPU 管理。另外，HIX 建议保护 PCIe IO 子系统免受特权软件的影响，并使用 CPU TEE 保护 GPU 驱动程序。虽然 HIX 不需要改变 GPU，但它的威胁模型比 Graviton 弱，因为 HIX 不能提供对直接物理攻击的保护。

尽管之前的研究提出了可信计算的 GPU 系统设计，但是一个关键的没有解决的问题是针对 GPU 的内存保护优化技术。Graviton 的前提是 3D 堆叠 DRAM 在 GPU 上可用，并假设 3D 堆叠的 DRAM 中的内存内容不会被泄露。然而，不仅许多独显使用 GDDRx 内存，很多集显也使用传统的 DDRx 内存。因此，为了给 GPU TEE 提供通用支持，硬件内存保护也必须对 GPU 内存可用。然而，现有的基于硬件的内存保护方案需要昂贵的加密和完整性验证。图 1 (a)描述了在 Graviton 中提出的带有可信堆叠内存的安全 GPU 执行系统，图 1 (b)显示了传统非可信内存的目标 GPU。

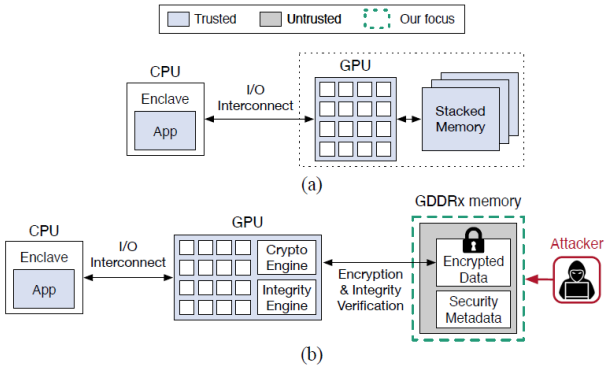

图 1

最近，基于硬件的可信 CPU 内存保护有了显著的改进。最先进的内存加密使用基于计数器的加密，它为给定的加密密钥、地址和计数器生成一次性密钥(OTP)。每个缓存行单元都有一个单独的加密计数器，每次数据写回到内存时，计数器都会递增，以保证其新鲜度。计数器缓存在计数器缓存中，只要片上计数器缓存中有上一级缓存（LLC）中丢失数据的计数器，就可以在数据从内存返回之前准备丢失数据的 OTP，需要使用 OTP 进行简单的异或解密操作。完整性验证对每个缓存行数据使用消息身份验证码（MAC），计数器值验证使用计数器树。

然而，即使处理器内存加密和完整性保护最近有所改进，GPU 内存的保护仍有待研究。本文采用了 Graviton 提出的 GPU 可信执行模型，但内存在传统 GDDRx 中是不可信的。本研究是第一个评估硬件 GPU 内存保护对性能影响的研究，并提出了一种新的技术来减少性能开销。研究表明，计数器缓存缺失会导致 GPU 应用程序的显著性能下降，因为计数器块的空间局部性比较差。由于 GPU 是为了高性能而使用的，因此安全性方面的性能开销可能会延缓 GPU 的可信计算的快速和广泛采用。

为了减少内存保护导致的性能开销，本文提出了一种新的 GPU 计数器表示技术，称为 COMMON COUNTER，利用 GPU 应用程序的独特特性。对于 GPU 应用程序，大部分应用程序内存通过 CPU 内存的初始数据传输写入一次。此外，多个 GPU 内核的执行往往会对主要数据结构的每个元素产生相同数量的写操作，因此在一个内核执行后，一个公共数据结构的每个内存块的计数器值往往具有相同的值。因此，当 GPU 应用程序包含多个具有数据依赖的内核执行时，这些计数器的值将在整个内核

执行流中均匀递增。

利用 GPU 应用中计数器值的一次写入和一致进度特性，本文提出了一种公共计数器表示方法，对现有的逐缓存行计数器管理方法进行了扩展。本文提出的机制可以有效地识别哪些内存块在内核执行或从 CPU 端代码传输数据后具有相同的缓存行数据单元计数器值。在内核执行期间，如果一个缺失的缓存行的计数器可以从几个已知的公共计数器值中得到服务，那么计数器缓存将被绕过，请求的计数器将直接由公共计数器提供。因为相应的计数器值从少量的片上普通计数器中获得，对于大多数 LLC 缺失，计数器缓存缺失明显减少。

为了启用公共计数器，每个应用程序的 GPU 上下文必须使用不同的内存加密密钥，因为应用程序内存的所有计数器都必须在上下文创建过程中重置。一旦受信任的 GPU 命令处理器创建了 GPU 上下文，该上下文的内存页面就会被每个上下文的加密密钥加密，并且为该上下文分配的内存块的计数器值将被重置。这个计数器的重置并不违反安全要求，因为新上下文的内存页是由新密钥加密的。

基于 GPGPU-Sim 仿真的评估表明，所提出的计数器管理技术几乎可以消除由于计数器缓存缺失而带来的性能开销。当与之前的 MAC 优化相结合，COMMON COUNTER 将一组 GPU 应用程序的内存保护性能开销降低到 2.9%，而最新的改进使用 Morphable 计数器可以在相同的 MAC 优化开销低 11.5%。

作为第一个基于硬件的 GPU 内存保护的研究，本文的贡献如下：

- 本文将现有的基于 CPU 的内存保护技术应用到可信 GPU 执行模型中。它指出计数器缓存缺失是 GPU 安全执行中性能下降的主要原因之一。
- 为了减少计数器缓存丢失的开销，本文利用 GPU 应用中大量的只读数据。
- 本文指出了 GPU 应用中计数器值的一种独特的更新行为，在这种情况下，大多数的计数器值通常与应用中少数的公共值一起进展。
- 本文提出了一种有效的机制来识别公共计数器，并为 LLC 缺失处理提供公共计数器，从而有效地绕过计数器缓存。

2 原理和优势

2.1 可信执行的内存保护

内存保护需要支持保密性和数据完整性驻留在不受信任的外部 DRAM。当一个内存块被带入片上缓存时，该块被解密并必须验证其完整性。当被修改的缓存行被逐出时，内存块被加密，完整性元数据也被写入加密的数据中。用于保密的常用硬件加密采用基于计数器的一次性密钥 （OTP）.。

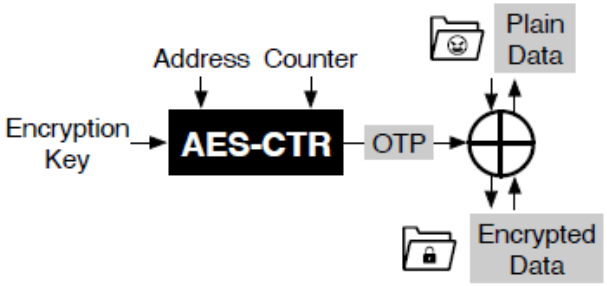

图 2

如图 2 所示，加密是通过以下方式完成的：1)从加密密钥、地址和计数器值创建 OTP，2)用被驱逐的缓存行对 OTP 进行异或。使用像 AES 这样的对称分组密码来生成 OTP。对于内存块的每个缓存行单元，必须维护一个单独的计数器。而且，每当从片上缓存的脏驱逐更新内存块时，计数器的值就会增加。即使使用相同的加密密钥，每个块的计数器也保证加密数据的新鲜度。

完整性验证检测任何损坏的数据。对于完整性验证，在之前的研究和商业处理器中使用了 Merkle 树的变体。一个简单的设计是从整个内存块创建一个哈希值树。这棵树的根不长叶子安全处理器，但其余的散列树节点可以存在于不安全内存中。为了提高性能，可以将哈希树的一部分缓存到处理器内部的哈希缓存中。当 LLC 缺失将内存块带入处理器时，将根据树中存储的哈希值来验证该内存块的计算哈希值。如果对应的哈希值在哈希缓存中不存在，则必须先从内存中读取哈希节点，并由哈希值的父节点进行验证。

通过将哈希树与反模式加密相结合，研究了对完整性树的改进。原始 Merkle 树的高度可能很高，因为树的叶子节点必须覆盖整个数据存储区域。改

进的 Bonsai Merkle 树(BMT)只创建内存块计数器值的哈希树。对于每个内存块，生成一个键控哈希值或消息验证码(MAC)，用于完整性验证。另外，筹码的完整性树(Bonsai Merkle tree)保证了筹码的新鲜度，防止了潜在的重放攻击。由于 BMT 仅为计数器覆盖的内存区域比原来的 Merkle 树要小得多，所以它的高度要短得多。

BMT 有了更多的改进，可以将更多的计数器打包到中间树节点的单个块中，如图 3 所示。计数器块通常按照数据缓存行的大小组织。由于一个计数器块可以存储更多的计数器，不仅提高了计数器缓存的效率，而且降低了计数器完整性树的高度。拆分计数器将计数器值分解为次要计数器和主要计数器。在一个计数器块，每个计数器有一个次要计数器，同一个计数器块中的所有计数器共享一个主计数器。如果次要计数器达到限制，主计数器将增加，导致相应的数据块重新加密。VAULT 采用了改进的计数器完整性树树的每个级别都有不同的值，以便在计数器的紧凑表示和由于较小的计数器溢出而导致的重加密成本之间取得平衡。变形计数器提出了更紧凑的计数器表示，每个 64B 计数器块包装 128 个计数器。它动态地更改计数器表示，以匹配应用程序的计数器更新行为。

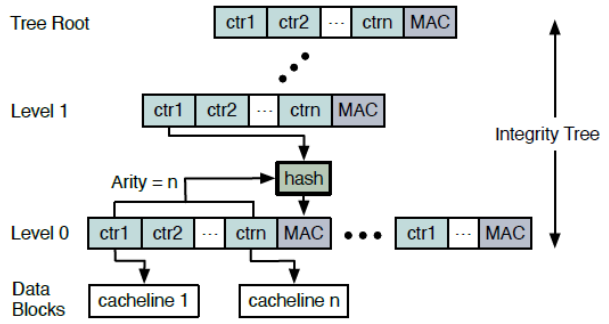

图 3

2.2 GPU 应用程序的写特性

本文的主要目标是降低计数器缓存未命中导致的性能开销。为了实现这一目标，本文利用 GPU 应用程序在内存写的独特特性，并开发了一个高效的加密引擎，以确保 GPU 的安全。观察到 GPU 通常以流的方式移动数据，这促使作者发现了 GPU 应用程序的一个独特属性，在不显著影响性能的情况下设计内存安全的 GPU 架构。GPU 应用程序倾向于以相当一致的方式创建内存写入到分配的内存区域。GPU 上下文中的大部分分配区域通常会收到相同数量的写操作，要么是 1)从主机上拷贝初始内存，要么是 2)如果应用程序写数据，扫描分配的内存，则会有超过一次的写操作。

方法：为了进行分析，作者使用真实的 GPU 进行实验，使用 NVBit 来执行 GPU 应用程序的二进制仪表并收集内存跟踪。NVBit 可以捕获所有的负载和存储内存访问，并记录访问的虚拟地址。然而，NVBit 缺乏以下能力：1)分析 L2 缓存上的访问是否丢失，2)识别脏缓存行的回写是否为缓存丢失服务。本文使用 GPU 核心的内存访问跟踪来分析写入行为，而不是使用 L2 遗漏跟踪。

为了研究内存访问计数中存在多少一致性，首先将上下文的内存空间划分为一组固定大小的数据块。对于每个块，跟踪块中的缓存行是如何更新的。如果块中的缓存行以统一的方式更新，则该块被表示为统一更新的块。在本节中，将分析统一更新的块与总内存大小的比率。
使用两种场景。首先，GPU 的基准分析使用了主评测中使用的基准套件。在这一节中，本文将在一个真实的系统上运行它们，而不是使用模拟器。其次，评估 7 个真实世界的应用程序，以进行更复杂的设置。

GPU 基准分析：图 4 报告了通过从 32KB 到 2MB 的块大小变化，统一更新的块在所有块上的比率。每个条被分解成两个类别；1) Read-only (solid)表示仅由主机首次写操作更新的内存块。2)非只读(虚线)显示块中每个缓存行有多个写操作。平均而言，当使用 32KB 块大小时。61.6%的块是均匀的更新块。当使用 2MB 的块大小时，27.5%的块被统一更新。随着块大小的增加，一个块中所有缓存行具有相同计数器值的可能性降低，因为该块覆盖了更多数量的缓存行，它们在内核执行期间很可能出现分歧。大量统一更新的块具有只读数据。然而。对于几个基准测试应用程序(fdtd-2d、sssp、pr、lib、hotspot 和 srad_v2)，很大一部分内存块会更新多次。

图 5 显示了 GPU 基准测试的整个执行之后，统一更新的块的不同计数器值的数量。对于仅具有只读数据的基准测试应用程序，不同的计数器值的数量是 1，因为统一更新的块只写入一次。但是，对于具有许多非只读数据的基准测试，不同的计数器值的数量可以是 2 和 3。请注意，该图显示了不同的计数器值的数量，不同基准测试中的实际计数器值差异很大。结果表明，在存在统一更新块的情

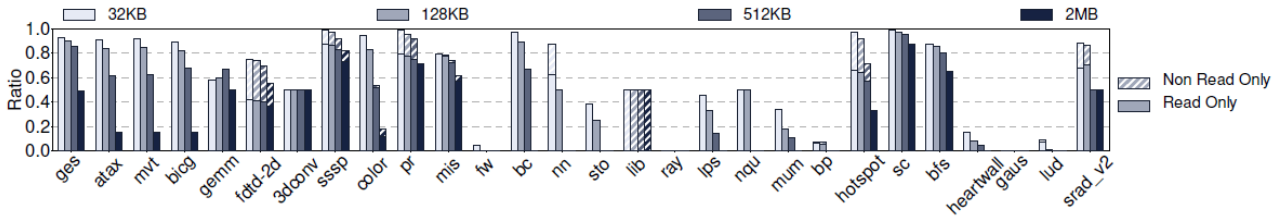

图 4

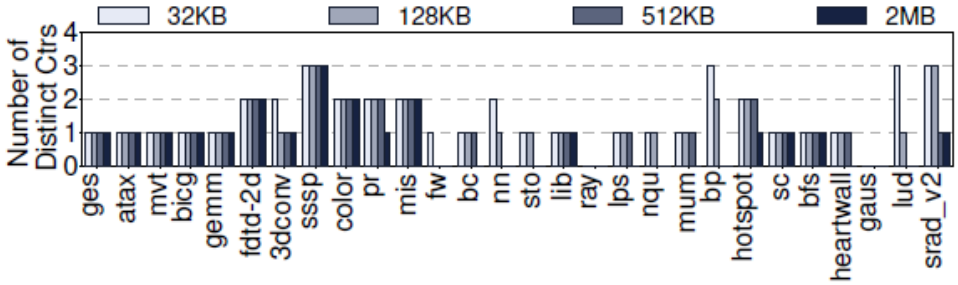

图 5

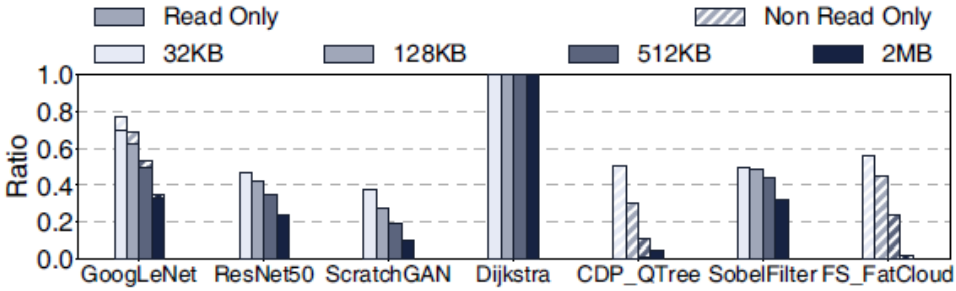

图 6

况下，少数常用的计数器可以覆盖统一更新块所使用的计数器值。

真实世界的应用分析：为了演示真实世界的 GPU 应用的属性，本文选择了 7 个成熟的 GPU 应用程序：2)对 ScratchGAN 进行 DNN 训练迭代运行，3)Dijkstra 算法寻找最短路径，4)使用 CUDA 动态并行度将 2D-map 转换为 CDP-QTree，5) SobelFilter 算法进行边缘检测，6)脂肪云的三维流体模拟(FS FatCloud)。

图 6 显示了真实应用程序中，随着数据块的大小从 32KB 增加到 2MB，统一更新的数据块在所有数据块中的比例。虽然现实世界的应用程序倾向于表现出比 GPU 基准测试更低的统一更新数据块比率，但相当一部分数据块仍然表现出统一更新写属性。在 googlet 中，根据块的大小，所有块中 34.5%到 84.4%是统一更新的块。ResNet-50 和 ScratchGAN 具有比 GoogLeNet 更复杂的模型结构，因此具有更低的统一更新内存块比率。平均而言，当使用 32KB 块大小时，所有块的 59.6%被统一更新。当使用 2MB 的块大小时，29.3%被统一更新。googlet、ResNet-50、ScratchGAN、Dijkstra 和 SobelFilter 大部分是只读的，而 CDP_QTree 和 FS Fatcloud 大部分是非只读的。

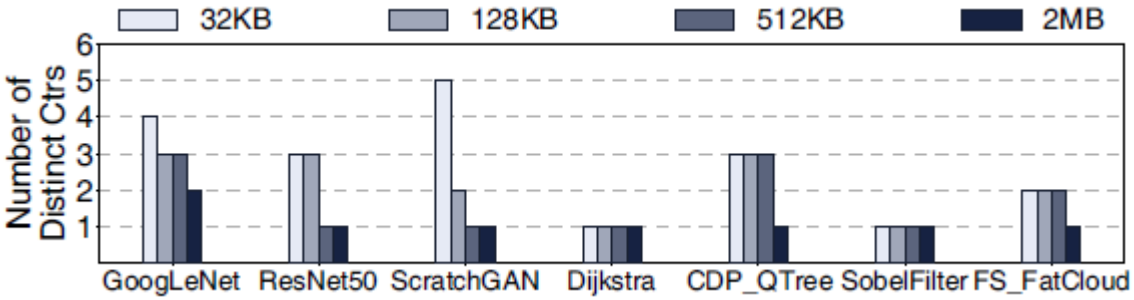

图 7

图 7 显示了真实应用程序执行程序后一致更新的块的不同计数器值的数量。与 GPU 基准相比，有更多不同的计数器值，最多可达 5。通过对 GPU 基准测试和实际应用的分析，发现一般的 GPU 应用程序倾向于均匀地写入内存，不同的计数器值的数量很少，可以存储在少量的片上存储器中。

3 研究进展

3.1 概述

为了减少 GPU 中计数器缓存丢失造成的性能损失，本文利用了在前一节中讨论的通用 GPU 应用程序的统一内存写行为。如果应用程序的计数器在应用程序初始化期间被重置，大多数 GPU 应用程序内存有几个可能的计数器值之一。这种被称为 Common Counter 的技术提供了内存地址到多个内存块共享的公共计数器值的有效映射。GPU 为每个上下文维护一些公共计数器(公共计数器集)。对于一个 LLC 未命中，加密引擎可以为未命中请求使用一个公共计数器值，如果所请求的地址对应的内存块的计数器是公共计数器值之一。因此，如果请求的 miss 可以由一个公共计数器提供服务，那么它不会访问计数器缓存。

为许多内存块支持这样的共享公共计数器。有几个变化是必要的。Common Counter 必要的硬件修改如图 8 所示。首先，每个上下文应该有一个单独的内存加密密钥。当安全 GPU 命令处理器为上下文分配新页面时，处理器用密钥重新加密内存页面，并将计数器值重置为 0。其次，GPU 硬件维护一个状态表，称为公共计数器状态映射(CCSM)，它告诉加密引擎请求的地址是否可以由一个公共计数器服务。

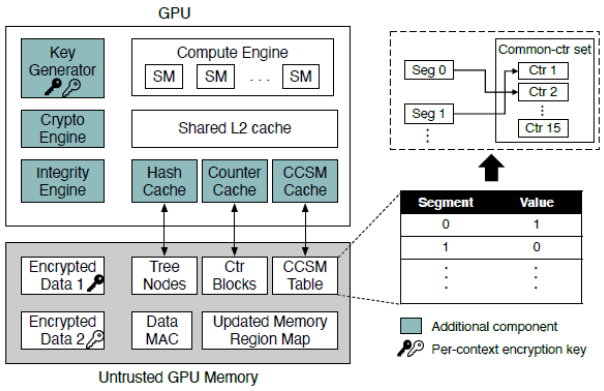

图 8

请注意，公共计数器表示形式补充了现有的每个缓存行加密计数器，因此仍然必须维护每个缓存行计数器和计数器完整性树。当许多页面具有相同的计数器值时，普通计数器只是简单地返回具有高度紧凑计数器表示的内存块的计数器值。因此，如果 GPU 内核更新了一个内存块，对应的内存块就不能再使用常用的计数器，因为内存块中的计数器值与当前时刻有偏差。因此，对于带有更新数据的 LLC 驱逐，原始计数器值也必须更新。此外，CCSM 标记内存块不能再使用公共计数器。

图 9 描述了建议的 COMMON COUNTER 保护方案的执行流程。在两个事件中标识公共计数器。第一个事件是从主机内存中传输初始数据。从主机内存中拷贝新数据时，数据以密文形式到达 GPU，该密文由 CPU 应用程序和 GPU 之间的共享密钥加密。GPU 对加密后的数据进行解密，并更新 GPU 内存。GPU 应用程序的许多内存页可能不会在初次更新后进一步更新，称之为初使写入一次。数据

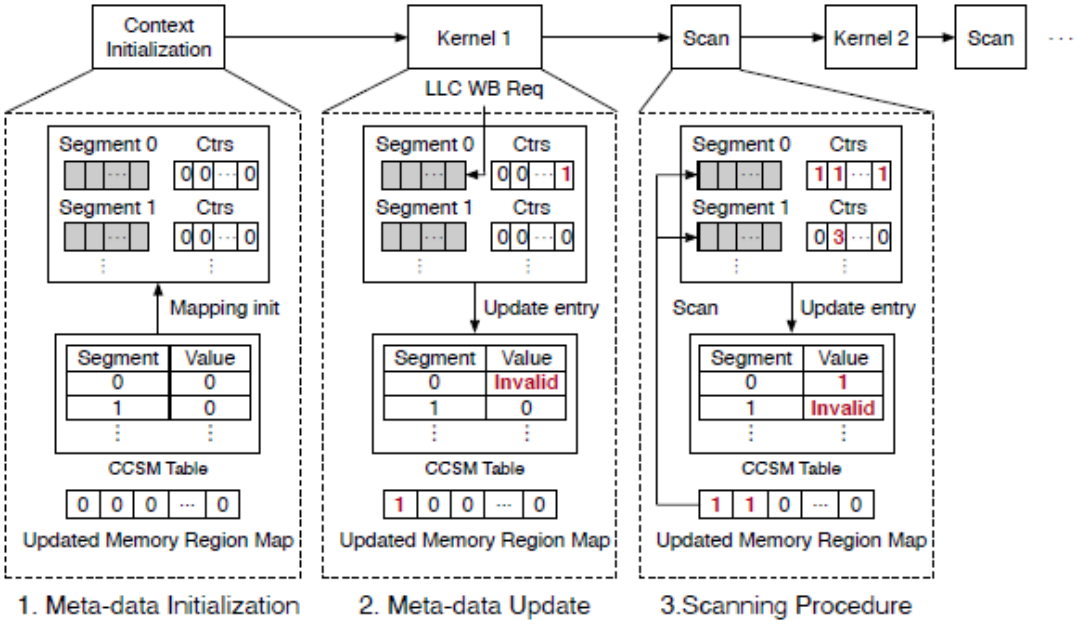

图 9

传输完成后。通过扫描更新内存块的实际计数器值来识别公共计数器。在这种情况下，计数器的值通常都加 1，因为内存副本只更新内存页一次。

其次，当内核执行完成时，将检查内核更新的内存页的所有计数器值。通过扫描计数器，可以识别具有相同计数器的内存块，并更新状态映射(CCSM)，以便下次内核执行时访问更新后的内存。

安全保障：使用普通计数器不会影响内存保护的保密性和完整性。基于计数器的加密和完整性树维护与常规机制相同。Common Counter 只查找由现有机制设置的公共计数器值，并为它们提供一个压缩的表示。它需要为每个新上下文重置计数器。为了避免计数器重置的潜在泄漏，每次重置都会使用一个新的加密密钥。

为了支持大多数内存页面的共享公共计数器，它需要以下额外的元数据：

逐上下文加密密钥：当一个新的 GPU 应用程序创建上下文时，一个内存加密密钥为新上下文生成。硬件内存加密引擎根据所请求数据地址的上下文更改内存加密密钥

GPU 范围的公共计数器状态图(CCSM)：对于一个内存块，状态图跟踪内存块是否使用了一个公共计数器。在文献资料方面，将 CCSM 管理的映射单元表示为段。请注意，段大小不需要匹配虚拟内存支持的页面大小。这个映射表是通过物理地址寻址的。本文使用 128KB 的段大小，每个段 4 位的 CCSM。如果 CCSM 表项的值无效，则该段不使用公共计数器。

逐上下文公共计数器集：对于每个上下文，维护一组公共计数器值。为了减少存储开销，本文只使用 15 个普通计数器。当 GPU 执行一个上下文，公共计数器集必须加载到片上存储快速访问。CCSM 中条目的值是上下文的公共计数器集的索引

更新的内存区域映射：这个映射记录了 CPU 或内核执行数据传输过程中更新的内存区域。这只是为了减少数据传输或内核执行后计数器值的扫描开销。本文采用粗粒度的映射，每个 2MB 区域使用 1 位。

3.2 可信 GPU 执行

可信 GPU 模型遵循了 Graviton 提出的相同设计，尽管模型假设外部 DRAM 是脆弱的。在可信 GPU 模型中，CPU 端用户应用程序运行在 SGX enclave 中。并与 GPU 建立信任。在初始化过程中，用户应用程序通过与远程 CA(证书颁发机构)验证 GPU 使用的签名来验证 GPU 本身。一旦认证完成。“user enclave”和“GPU”共用一个密钥。后续用户 enclave 与 GPU 之间的加密通信将使用该公钥。GPU 端命令处理器负责执行关键的管理操作，它是 TCB(可信计算基础)的一部分。部分 GPU 内存(隐

藏内存)预留给安全元数据，安全元数据由安全命令处理器和加密引擎独家可见和访问。

上下文初始化：在初始设置之后，用户 enclave 可以安全地请求创建一个新的 GPU 上下文。GPU 端命令处理器创建并初始化上下文。请注意，GPU 端命令处理器独立于任何特权 CPU 软件(包括操作系统)工作。因此，操作系统不能恶意干预操作。对于内存分配，GPU 命令处理器更新应用的 GPU 页表。在页表更新过程中，命令处理器确保不同的 GPU 上下文不共享物理页面，强制上下文之间的内存隔离。

对于所提议的内存保护技术，需要额外的步骤。对于上下文的创建，将生成一个新的内存加密密钥。当上下文计划在 GPU 中运行时，它的内存必须通过密钥进行加密和解密。当为上下文分配内存页时，内存将由新密钥重新加密。但是，这个初始的重新加密步骤不会产生任何额外的成本。因为即使在当前的 GPU 中，为了安全，新分配的页面的内存内容也必须被擦除。在本文的方案中，擦洗步骤生成相同的内存写操作，这些写操作使用硬件引擎加密的零值。此外。为新分配的页面重置相应的 CCSM 条目，而不使用公共计数器。

3.3 更新常用计数器状态

要启用常用计数器，必须标识内存中常用的计数器值。在这一步骤中，必须更新 CCSM 和公共计数器集。命令处理器为两个事件启动公共计数器更新任务：1)数据从 CPU 传输到 GPU 内存的完成，2)内核执行的完成。对于这两种类型的事件，将扫描更新内存段的计数器值。对于在此步骤中找到的新的公共计数器，公共计数器集将被更新，以插入或修改相应的条目。此外，CCSM 也更新了部分，以指向新的部分。

跟踪更新的内存区域：因为扫描整个物理内存的所有计数器块会导致巨大的开销。数据传输或内核执行维护一个粗粒度的更新内存映射，每 2MB 区域 1 位。对于 32GB 内存，只使用了 16KB 内存，而且它的空间局部性非常高，因为只更新了一小部分内存。它们被缓存在最后一级缓存中。虽然可以进一步优化跟踪更新的内存区域，但本研究使用这种简单的方法，因为它不会造成任何显著的开销。

在计数器块的扫描步骤中，首先检查更新后的内存区域。只有当区域被更新时，相应的计数器块才会被扫描以找到公共计数器。如果一个段中所有的计数器值相等。相应的 CCSM 条目被更新为指向新的公共计数器值。如果新值在公共计数器集中不存在，则将新值添加到该集中。

3.4 使用常用计数器

本节介绍常用计数器的使用方法 LLC 缺失，以及如何处理写操作。对于快速访问 CCSM，一个小的缓存添加到 GPU。

CCSM 缓存：由于公共计数器状态映射(CCSM)必须覆盖 GPU 的整个物理内存，所以不能将整个映射存储在片上存储中。代替。该技术使用专用于 CCSM 数据的缓存。与计数器缓存相比，CCSM 缓存的效率要高几个数量级。对于计数器缓存，128B 计数器缓存块分别占用 16KB 和 32KB 的数据内存。但是，一个 128B 的 CCSM 缓存块可以覆盖 32MB 的数据内存，比 128 位的计数器块的缓存效率高 2048 倍。拟议的公共计数器的改进源于这种存储效率。本研究使用了一个小型的 CCSM 缓存。其大小为 1KB。

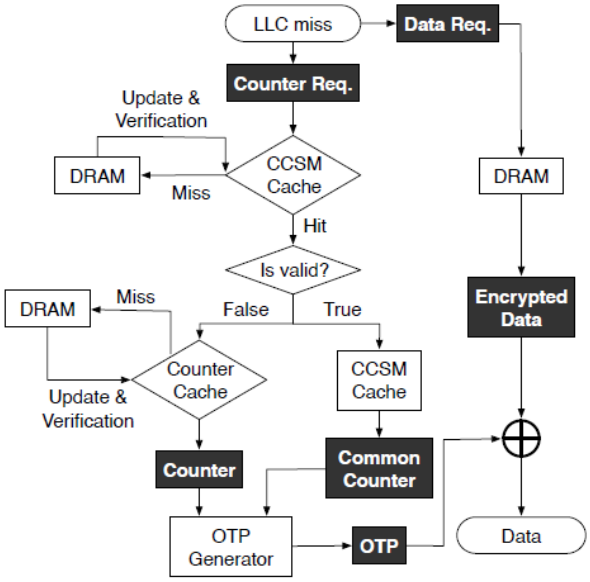

图 10

失误处理流程：考虑图 10。对于一个 LLC 缺失，一个数据请求被发送到内存控制器，同时，缺失的地址首先在 CCSM 缓存中检查，以确定 CCSM 状态。如果对应的条目不在 CCSM 缓存中，则必须从 GPU 的隐藏内存中检索 CCSM 条目。然而，由于映射效率很高，在 CCSM 缓存中这种缺失是很少

见的。一旦检查了 CCSM 条目，miss 处理程序就知道它是否可以为该地址使用一个公共计数器。

如果 CCSM 条目具有对上下文的公共计数器集的有效索引，则使用该集合中的公共计数器值。因为它保证了公共计数器值等于计数器块中的实际计数器值。如果 CCSM 表项将相应的段标记为无效(都是 1)，则访问计数器缓存以查找相应的计数器表项。如果计数器缓存丢失发生，应该像基线内存加密引擎那样处理它。一旦计数器值从公共计数器集或计数器缓存中可用。生成 miss 的 OTP。当数据从内存到达时，使用生成的 OTP 对数据进行解密。

处理写：将讨论写流，假设写回/写分配 LLC。尽管可以支持其他写策略。对于写操作，需要两次元数据更新。1)首先，与传统的反模式加密类似，当脏缓存行最终从最后一级缓存中被驱逐到内存时，相应的计数器条目必须被更新。普通的计数器是计数器值的一种快捷方式，但是每个内存块的实际计数器仍然被维护。2)其次，CCSM 条目现在被更新为无效的，因此任何对同一个地址的后续读/写都不允许使用公共计数器，因为计数器的值现在是递增的。请注意，一旦内核执行完成，并且公共计数器状态更新过程为内存段找到统一的计数器值，更新后的内存段的 CCSM 条目将被重置为一个公共计数器。

3.5 硬件开销

元数据大小：CCSM 必须与其他安全元数据一起分配到 GPU 内存的固定位置。CCSM 的大小取决于 GPU 内存的大小，每个上下文的公共计数器的数量，段的大小。段大小设置为 128KB，每个上下文的公共计数器数量为 15，每个段需要 4 位。对于每个 1GB 的 GPU 内存，需要 4KB 的 CCSM 容量。

片内存储：当每个上下文的公共计数器数量为时，公共计数器集需要 15 × 32 位 15. 如果多个上下文可以在 GPU 中同时运行。多个公共计数器组需要添加在上。芯片存储。除非两个上下文同时运行，否则公共计数器集将保存在上下文元数据内存中，并由 GPU 调度器恢复。

该架构需要额外的片上缓存，包括 1KB CCSM 缓存、16KB 计数器缓存和 16KB 哈希缓存。注意，计数器缓存和哈希缓存也包含在前面的内存保护技术中。例如 SC_128 和变形计数器。作者用仙人掌 6.5 估算额外片内缓存的面积和功率开销。本文发现所有片上缓存的面积为 0.11 mm2，是 TITAN X Pascal (GP102)模区面积的 0.02%，漏电功耗为 11.28 mW。

3.6 评估

模拟器：在 GPU 模拟器 GPGPU-Sim 上开发公共计数器架构。表一给出了详细的 GPU 配置，其模型为 NVIDIA TITAN X Pascal (GP102)。评估的图形处理器型号使用的是 GDDR5X DRAM，没有使用 HBM2 等 3D 堆叠内存。对 GPGPU-Sim 模拟器进行了修改，以模拟内存保护技术，包括加密/解密和完整性验证逻辑，以 sc128 为基础开发了 Common Counter 方案。此外，本文对变形计数器进行了更高的评估，每个缓存行有 256 个计数器。

数据 MAC 验证：考虑了两种不同的数据 MAC 验证方法。第一种方法是将数据与 MAC 一起从内存中取出，这对芯片外内存带宽造成了更大的压力。第二种方法是在一个鼓舞人心的前期工作中提出的技术 Synergy，它提出使用 ECC 芯片作为 MAC 的载体，实际上提供了完全消除 MAC 引起的性能开销。Common Counter 不是为 MAC 诱导的开销减少而设计的。这使得使用协同更有利于我们的技术。我们报告两种情况的结果。

性能改进：图 11 显示了三种不同内存保护方案的归一化 IPC：(1)分割计数器技术(SC 128)，(2)变形计数器(Morphable)，(3) Common Counter 技术。图 11 (a)提供了数据和 MAC 从芯片外内存中检索的情况下的结果，而图 11 (b)显示了 MAC 读取与 Synergy 建议的 ECC 读取内联时的结果。

图 11 (a)和(b)之间的差距表示当从内存中单独读取 MAC 时，数据 MAC 验证开销导致的性能损失。COMMON COUNTER 降低了这两种情况下安全内存的性能下降。但在协同 MAC 的设计下，COMMON COUNTER 的性能比 sc128 和 Morphable 有更高的提升。与不安全基线相比，COMMON COUNTER 的性能开销平均从(a)的 13.9%降低到(b)的 2.9%。从(a)到(b)的不可忽略的跳跃表明 Common Counter 技术与有效的数据 MAC 验证的集成 Synergy 等技术可能会提供显著的性能改进。

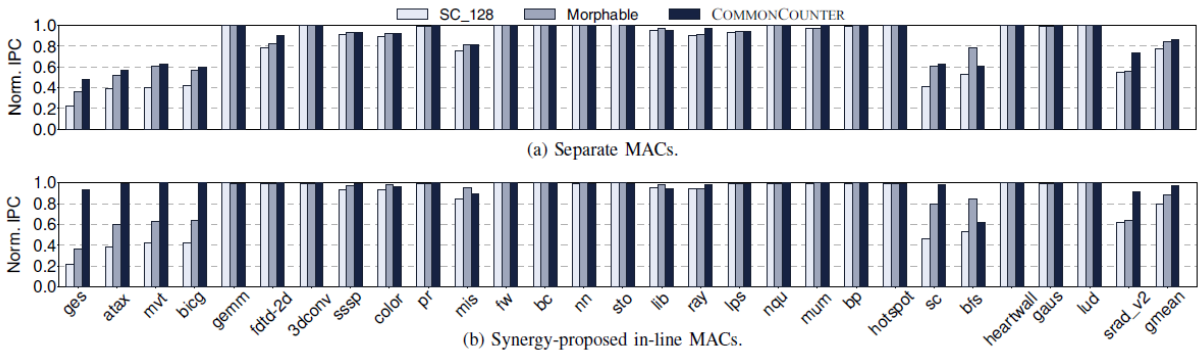

图 11

正如在第二节 a 中讨论的，它表明需要减少计数器缓存丢失和 MAC 流量，以提高整体性能。

在图 11(b)中，可以看到 SC_128 和 Morphable 技术平均有 20.7%和 11.5%的性能开销，而 Common Counter 只增加了 2.9%的性能开销。显著的减少来自于这样一个事实，即所提议的缓存子系统使用公共计数器有效地服务于 per 缓存行计数器请求。具有高性能降级的基准测试应用程序 ges、atax、mvt、bicg、sc 和 srad v2)得到了显著的改进。性能改进范围从 sradv2 优于 SC 128 和 Morphable 的 46.4%和 42.4%，到 ges 的 326.2%和 156.4%。尽管 COMMON COUNTER 在大多数工作负载上提供了比 SC 128 和 Morphable 更好的性能。对于 lib 和 bfs, Common Counter 的性能低于 Morphable。对于这两个应用程序，由于它们的更新模式，有许多 LLC 缺失没有得到公共计数器的服务，Morphable 的 256-arity 计数器块比 Common Counter 使用的 128-arity 计数器块提供了更好的性能。然而。Common Counter 可以通过在 Morphable 上添加公共计数器来改进。增加其计数块的基数。

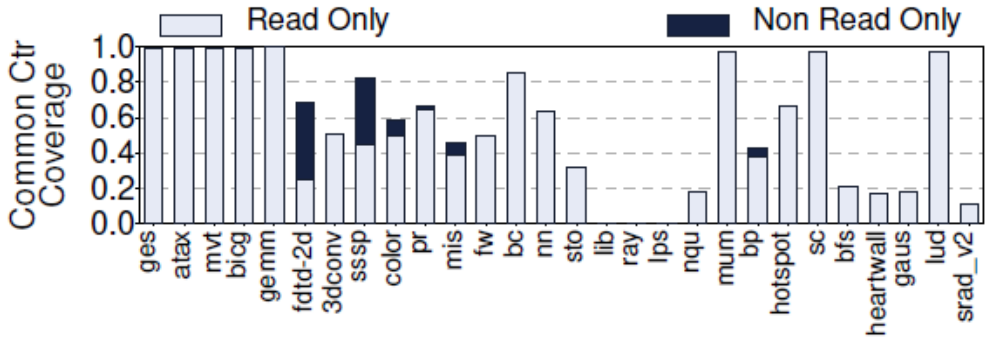

图 12

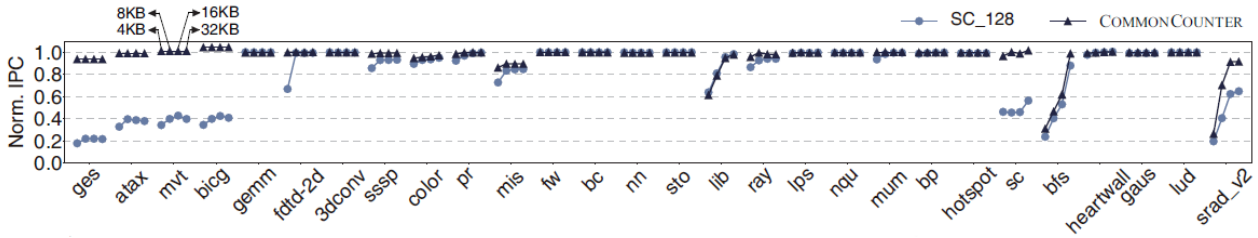

图 13

上面讨论的性能改进可以归因于有效使用公共计数器。图 12 显示了普通计数器服务的芯片外内存访问占所有计数器请求的比例。每个条被分解为只读数据(浅灰色)和非只读数据(深灰色)。注意到，对于在图 13(即 ges、atax、mvt、bicg 和 sc)中看到

巨大性能优势的基准测试，公共计数器的内存访问覆盖率接近 100%，这显示了公共计数器的有效性和性能增益之间的相关性。

计数器缓存大小的敏感性：图 13 显示了性能如何随着计数器缓存大小的变化而变化。一般来说。与 sc128 相比，Common Counter 受计数器缓存大小变化的影响较小，因为有很多 LLC miss 请求由公共计数器提供服务，即使相关的计数器在计数器缓存中不可用。例如，当使用 Common Counter 时，与非安全 GPU 相比，sc 几乎没有性能损失，因为计数器缓存大小减小。相反，对于 SC 128，当计数器缓存大小为 32KB 时，SC 表现出 43.6%的退化，当计数器缓存大小缩小到 4KB 时，SC 的损失急剧增加，达到 53.7%。可以观察到，对于某些基准测试，Common Counter 对计数器缓存大小的变化也很敏感。例如，当计数器缓存大小减小时，库会经历显著的性能损失，这是因为这个基准测试很少有机会使用常见的计数器，如图 12 所示。

扫描开销：为了识别能够使用普通计数器而不是常规计数器的段，建议的机制需要在每次内核执行的边界上扫描更新的内存区域。如果扫描开销很大，那么使用公共计数器带来的性能好处将会减少。为了粗略估计扫描开销，运行模拟来确定更新内存区域的大小。对于已识别的更新内存大小，本文使用一台配备了 GTX 1080 的真实机器来测量内存区域扫描延迟。表 III 显示了端到端应用程序运行期间执行的内核数量、需要扫描的更新内存区域的总大小，以及扫描开销占总执行时间的比例。扫描开销比从 0.372% (3dconv)下降到 0.004% (bfs)，这表明扫描开销几乎可以忽略不计。请注意，在所有性能实验中都包含了报告的扫描开销。

4 总结与展望

本文探讨了支持基于硬件的内存保护对 GPU 计算的性能影响。本文在分析 GPU 应用统一写特性的基础上，提出了使用公共计数器，几乎完全消除了计数器缓存丢失的影响。本文利用了 GPU 独特的内存更新行为，研究表明这种基于硬件的内存保护对于高性能 GPU 计算是可行的。

集成图形处理器：虽然本文研究的是具有自己的非信任内存的独立图形处理器，但 COMMON COUNTER 经过适度的更改可以扩展到集成图形处理器模型。在 GPU 集成模型中，由于 CPU 和 GPU 通过共享内存控制器共享内存，CPU 和 GPU 不仅可以共享受保护内存，还可以共享内存加密和完整性保护引擎。要为这样一个集成的 GPU 模型启用公共计数器，需要为 CPU 和 GPU 的每个上下文分别设置一个单独的加密密钥。此外，每个上下文的计数器是单独管理的，在上下文的初始化中重置，这与当前安全处理器中的通用内存加密和计数器管理不同。这种方法是 Rogers 等人提出的，将硬件内存加密与虚拟内存支持相结合。研究集成 GPU 的内存保护设计作为未来的工作。

支持多个 GPU 共享内存：如果运行单个应用程序上下文的多个 GPU 共享它们的内存，那么这些 GPU 必须为上下文共享相同的加密密钥。此外。计数器缓存的一致性必须保持，因为不同的 GPU 可以潜在地写入共享数据，更新相应的计数器。尽管对于任何共享内存受保护的多芯片安全处理器，也需要这种支持，但在 GPU 上进行研究可以作为未来的工作。