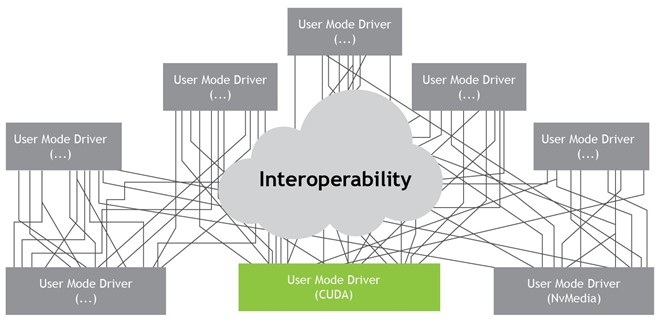
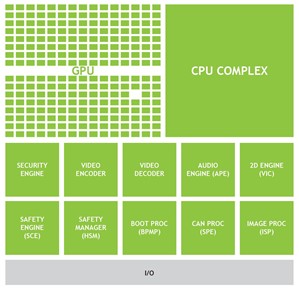
**通过与NvSciBuf和NvSciSync的互操作性共享CUDA资源**



嵌入式和 HPC 应用程序越来越需要共享资源并控制跨多个硬件引擎和软件应用程序的流水线工作流的执行。下图深入了解了 NVIDIA 嵌入式平台上可以支持的引擎数量。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/NVSciSync.jpg)**

*图1.NVIDIA 嵌入式平台上的各种硬件引擎。*

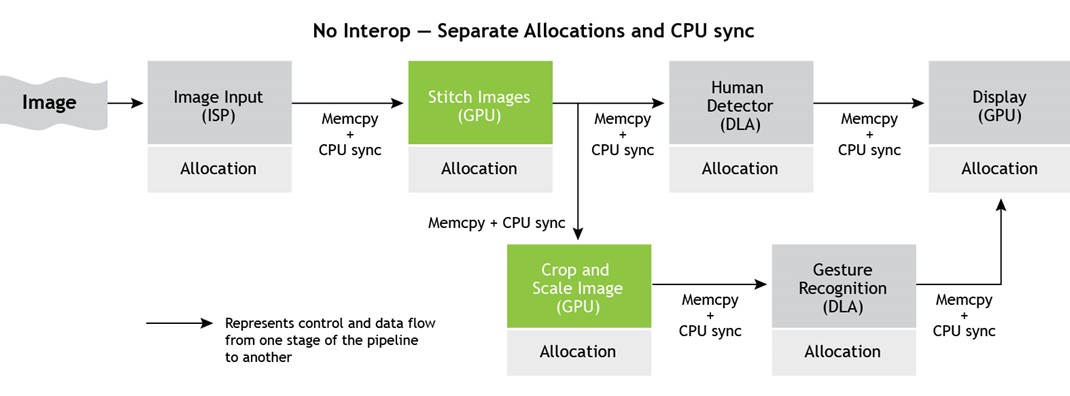
除了 GPU 之外，还包括图像信号处理器 （ISP） 等图像处理单元和深度学习加速器 （DLA） 或可编程视觉加速器 （PVA） 等加速器。其中每个引擎又具有一个或多个用户模式驱动程序 （UMD） 来公开功能，包括：

* **GPU**： CUDA ，OpenGL， Vulkan
* **ISP：** NvMedia， Argus
* **DLA**： NvMedia， TensorRT

这篇文章介绍了CUDA 10.2中引入的新功能：与NvSciSync和NvSciBuf的互操作性，旨在提供统一的互操作性解决方案。这些解决方案安全、可靠且坚固，足以用于安全关键型应用，例如 [**NVIDIA DRIVE AGX 平台**](https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/)。

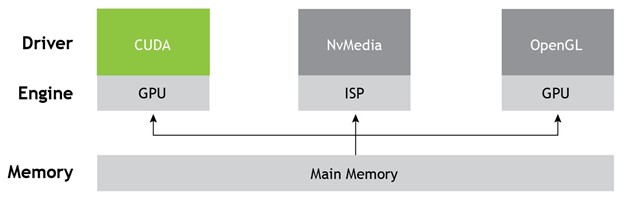
**什么是互操作性？**

下面是一个示例：手势识别等典型机器学习算法可以使用 ISP 进行图像输入，使用 DLA 进行推理，并使用 GPU 执行拼接、裁剪、缩放和显示等图像处理算法。图 2 显示，此类工作流通常以流水线方式执行，一个工作流的输出充当下一个工作流的输入。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Pipelines-execution-umd-memory.jpg)**

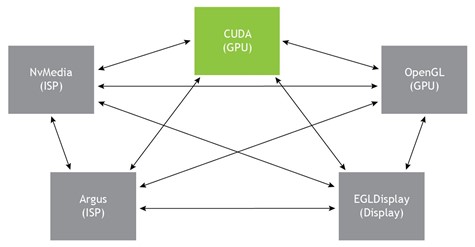
*图2.没有互操作性的流水线执行。*

跨引擎和驱动程序跨进程和系统共享资源的机制避免了特定于引擎的 UMD 内存分配器以及使用主机或 CPU 作为中介的引擎之间的数据迁移和同步。此机制提供了共享内存和同步基元的互操作性。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/CUDA-sharing-main-memory-across-engines.jpg)**

*图3.跨引擎共享主内存。*

**CUDA 互操作性的演变**

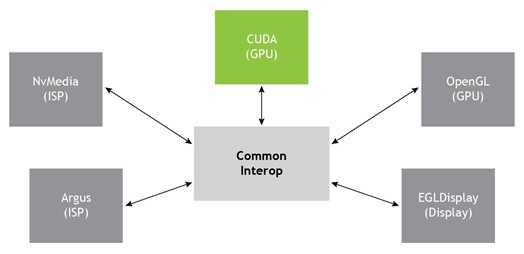
**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/cuda-interoperability.jpg)**

*图4.一对一互操作性。*

传统上，CUDA 支持与其他 NVIDIA 支持的 GPU 驱动程序（如 OpenGL 或 DirectX（如 CUDA-OpenGL）进行一对一互操作，以便 CUDA 可以访问 OpenGL 分配的内存。

这种一对一互操作无法针对嵌入式系统进行扩展，因为维持如此多的互操作的可维护性成本很高，更不用说开发人员理解和对每组 UMD 使用新互操作的开销了。

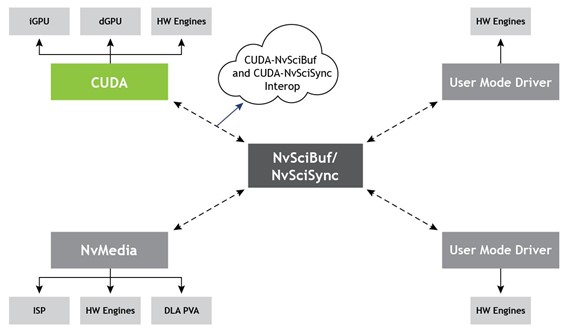
当务之急是适应可以插入以支持多个驱动程序和硬件单元的常见互操作。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/cuda-Common-interoperability.jpg)**

*图5.通用互操作性。*

EGLStream 和 EGLImage 是 CUDA 支持的一些常见互操作，广泛用于 NVIDIA 平台。

**NvSciBuf 和 NvSciSync 的动机**

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/CUDA-NvSciBuf-and-CUDA-NvSciSync-architecture.jpg)**

*图6.CUDA-NvSciBuf 和 CUDA-NvSciSync 架构。*

NvSciBuf和NvSciSync的开发是为了满足以下要求：

* 允许在引擎和 UMD (用户模式驱动程序)之间共享内存和同步基元。
* 允许跨线程、进程和 VM 分区共享。
* 为存储在引擎本地缓存中的数据启用缓存一致性。
* 同步以确保有序访问，并在流水线应用程序中提供控制流。
* 允许使用特定于驱动程序的数据类型。

您可能会问，为什么不能只使用现有的 [**EGL 互操作性**](https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/index.html#egl-interoperability)？虽然 NVIDIA 继续支持 EGL，但使用不同的互操作有几个原因：

* EGLStream 生产者公开的分配 API 未考虑 EGLStream 使用者的引擎限制，并且经常导致使用者端的映射失败。
* EGL接口在设计时没有考虑到汽车安全，并且具有无法在汽车级安全认证系统中使用的故障模式。
* EGLStream 在应用程序上强制生产者-使用者语义，如果您只想设置控件依赖项，则这些语义并不灵活。
* EGL只能识别二维图像数据。它无法处理 [**DLA**](https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#dla_topic) 等干扰引擎所需的张量或其他非图像传感器数据。

**NvSci简介**

与 NVIDIA SDK 捆绑在一起的 NvSci（NVIDIA 软件通信接口）库有两个主要用途：

* 它们允许预先分配资源，访问限制由应用程序定义，并且充分了解其总体需求的详细信息。
* 它们允许在彼此不了解的库之间交换资源。

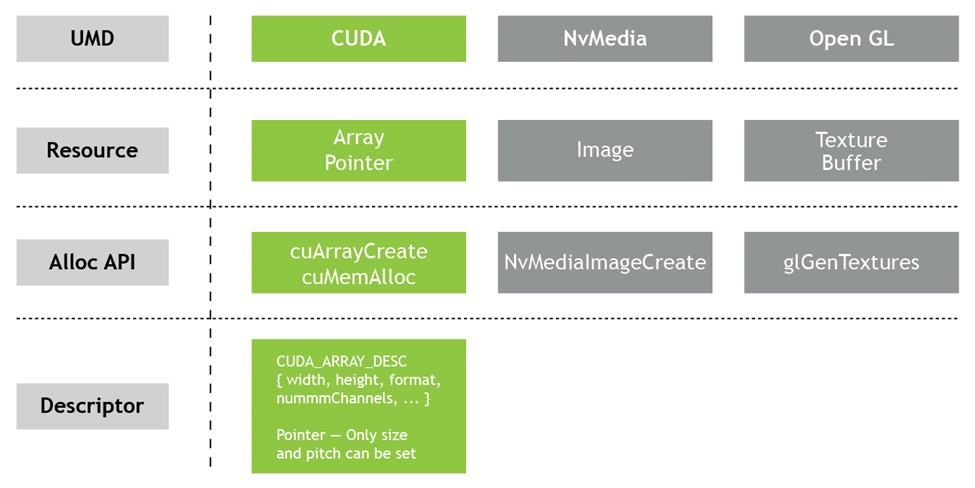
NvSci 公开了多个库以满足单独的功能：

* 用于缓冲液管理的 NvSciBuf
* 用于同步的 NvSciSync
* 用于进程间通信的 NVIPC
* 用于流媒体的 NvStream

其中，CUDA公开接口以导入NvSciBuf和NvSciSync分别作为CUDA外部存储器和信号量。有关详细信息，请参阅 [**NV流**](https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Graphics/nvsci.html)。

**CUDA-NvSciBuf 互操作概述**

使用当前特定于 UMD 的内存管理 API，可以进行改进以允许一个 UMD 的分配由另一个 UMD 使用，可能作用于不同的引擎，特别是考虑到特定于 UMD 的分配器和数据类型之间的兼容性。图 7 显示了 CUDA 数据类型与其他数据类型的不同之处，其中一些数据类型公开了图像或纹理等更高级别的结构，而不是 CUDA 指针和数组，这使得使用另一个更难重建一个。

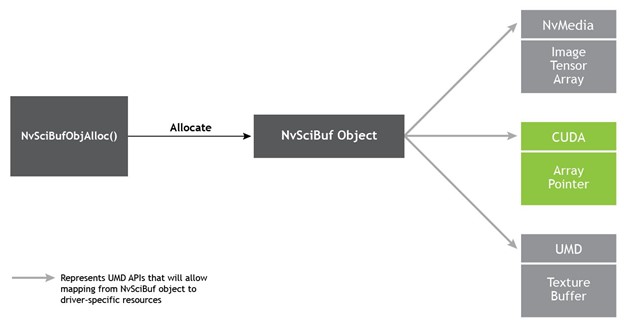
**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Data-types-supported-across-various-UMDs.jpg)**

*图7.各种 UMD 支持的数据类型。*

**什么是 NvSciBuf**

NvSciBuf 是一个缓冲区分配模块，使应用程序能够分配可在硬件和软件单元之间共享的缓冲区。NvSciBuf具有双重目的：

* 充当了解所有引擎约束的谈判者。
* 分配可由各个 UMD 使用并根据其现有数据类型访问的内存。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Overview-of-NvSciBuf-allocation-and-its-mapping.jpg)**

*图8.NvSciBuf分配及其映射概述。*

图 8 显示 NvSciBuf 对象根据应用程序规范进行分配，然后可以将其作为特定于 UMD 的数据类型导入。

**NvSciBuf工作流程**

NvSciBuf的缓冲区分配模型扩展了多个阶段。有关更多信息，请参见[**缓冲区分配**](https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Graphics/nvsci_nvscibuf.html)。

* **指定属性** – 如果两个或多个硬件引擎必须访问公共缓冲区，则应用程序以列表的形式为每个访问器指定属性。
* **协调** - 协调各个列表中的所有属性，并检查可行的分配。
* **分配** – 如果对帐通过，则会分配缓冲区。
* **通信** – 与所有访问器共享缓冲区。
* **导入** – 每个访问器 UMD 都会根据其内存模型公开接口以导入 NvSciBuf。例如，在CUDA UMD的情况下，[**CUDA外部存储器接口**](https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#nvidia-softwarcommunication-interface-interoperability-nvsci)提供了导入NvSciBuf对象的机制。

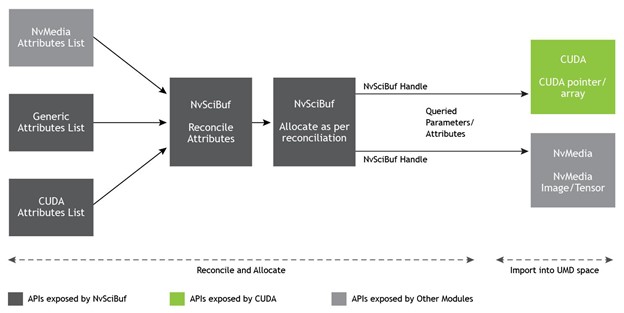
**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/NvSciBuf-workflow.jpg)**

图9.NvSciBuf 工作流程。

**将 NvSciBuf 与 CUDA 结合使用**

CUDA 支持使用 cudaImportExternalMemory 函数将 NvSciBufObj 对象作为 NvSciBuf 类型的 [**CUDA 外部存储器**](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EXTRES__INTEROP.html#group__CUDART__EXTRES__INTEROP)导入。导入后，使用 cudaExternalMemoryGetMappedBuffer 或 cudaExternalMemoryGetMappedMipmappedArray 将导入的 NvSciBuf 映射为 CUDA 指针或 CUDA 数组。[**API 规范**](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EXTRES__INTEROP.html#group__CUDART__EXTRES__INTEROP)提供了有关函数签名的详细信息，而[**编程指南**](https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#importing-memory-objects-nvsci)提供了有关如何在 [**CUDA 应用程序和**](https://www.google.com/url?q=https://github.com/NVIDIA/cuda-samples/tree/master/Samples/cudaNvSci&sa=D&ust=1603263761779000&usg=AOvVaw0jc4D1Q8XJT69qGZXbEOUN)最佳实践中使用它们的指导。

作为导入的前体，应用程序应查询分配的 NvSciBufObj 以获取填充 CUDA 描述符 cudaExternalMemoryHandleDesc 或 cudaExternalMemoryBufferDesc 所需的属性，这些属性作为参数传递给导入映射。如果导入到 CUDA 中的 NvSciBuf 对象也被其他驱动程序映射，则应用程序必须使用下面描述的 CUDA 外部信号量 API 作为适当的屏障，以保持 CUDA 与其他驱动程序之间的一致性。创建 CUDA 指针或数组后，它可以与 CUDA 内核和 CUDA 函数（如 memcpy 或 memset）一起使用，并使用 CUDA 函数释放。对于NvSciBufObj上的操作，应用程序应引用NvSciBuf函数。

**安全适应**

为了用于安全关键平台，NvSciBuf互操作提供了一些附加功能。分配、导入和映射功能都可以在非安全关键或初始化阶段完成，从而消除安全关键任务模式阶段的非确定性故障。此外，这也符合安全合规所需的ISO标准。

使用 NVIDIA DRIVE OS 的应用程式必須遵守 NVIDIA DRIVE OS 安全手冊，以确保 API 可以在安全環境中使用。

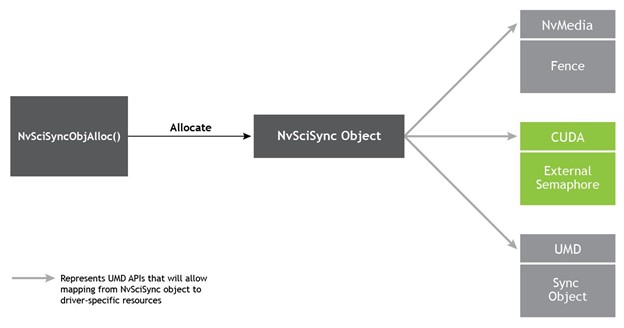
**CUDA-NvSciSync 互操作概述**

对于 [**NVIDIA DRIVE AGX 平台上**](https://www.nvidia.com/en-us/deep-learning-ai/products/agx-systems/)的应用程序，除了 NvSciBuf 满足的共享数据的能力外，还必须满足引擎之间的执行依赖关系。这种执行依赖项的一个例子是，当 CUDA 应该访问缓冲区以执行推理时，例如，只有在相机完成缓冲区初始化之后。执行控制的一个关键特征是每个 UMD 都有自己的编程模型。例如，CUDA 遵循控制流模型，而其他 UMD（如 NvMedia）遵循数据流模型。

构建用于共享同步对象的互操作机制以进行协调必须确保同时遵守现有的编程模型。

**什么是 NvSciSync**

NvSciSync 是一个通用同步框架，它允许 UMD 以它认为合适的方式解释 NvSciSync 对象，并使描述复杂的应用程序工作流变得更加简单。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Introduction-to-NvSciSync.jpg)**

*图 10.介绍 NvSciSync.*

**NvSciSync 工作流程**

NvSciSync的同步模型扩展了多个阶段。基本设置过程类似于用于分配缓冲区的过程。

* **指定信号器的属性** - 向同步对象发出信号的硬件组件施加的限制。
* **指定等待者的属性** - 等待同步对象的硬件组件施加的限制。
* **协调** - 协调并分配同步对象。
* **共享** - 与访问者共享分配的同步对象。
* **映射** - 将同步对象映射到特定于 UMD 的接口。

有关详细信息，请参阅[**同步**](https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Graphics/nvsci_nvscisync.html)。

**将 NvSciSync 与 CUDA 结合使用**

CUDA 支持将 NvSciSync 对象作为 NvSciSync 类型的 [**CUDA 外部**](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EXTRES__INTEROP.html#group__CUDART__EXTRES__INTEROP)信号量导入。互操作使传统的 CUDA 流能够等待 CUDA 域之外的任务，并使其他 UMD 能够本机等待在 CUDA 流中排队的任何任务。

如果使用 CUDA-NvSciSync 互操作来确保对导入到 CUDA 中的共享 NvSciBuf 对象的互斥访问，则应用程序必须设置正确的标志。有关更多信息，请参阅 [**NVIDIA 软件通信接口互操作性 （NVSCI）。**](https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#nvidia-softwarcommunication-interface-interoperability-nvsci)

**安全适应**

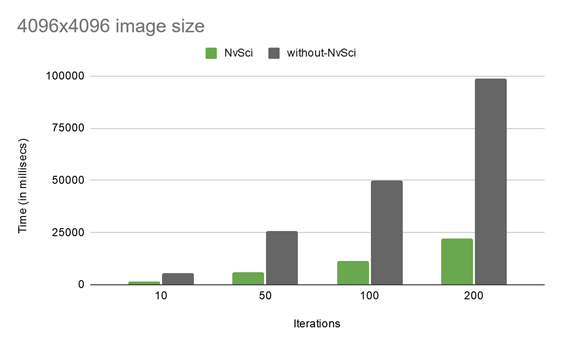
与NvSciBuf一样，NvSciSync互操作提供了符合安全要求的功能。应在 init 阶段调用分配和导入函数，以完成所有分配并避免在运行时阶段出现非确定性故障。运行时只允许等待和信号调用。此外，这也符合安全合规所需的ISO标准。

应用程序必须遵守 NVIDIA DriveOS 安全手册，以确保 API 可以在安全环境中使用。

**示例应用程序和性能**

示例应用程序演示了使用和不使用 NvSciBuf 和 NvSciSync API 的 CUDA-NvMedia 互操作，以评估性能提升。它将RGBA图像作为输入，并在NvMedia2DblitEx的帮助下将其转换为YUV 420。然后，YUV输出由CUDA内核转换为灰度图像。此操作执行两次，一次使用 NvSci 分配器和同步，另一次不使用 NvSci，并记录计时。有关带和不带 NvSci 的示例工作流的更多信息，请参阅 cudaNvSciNvMedia Workflow （PDF） 和 [**NVIDIA/cuda-samples/Samples/cudaNvSciNvMedia**](https://github.com/NVIDIA/cuda-samples/tree/master/Samples/cudaNvSciNvMedia) GitHub 存储库。

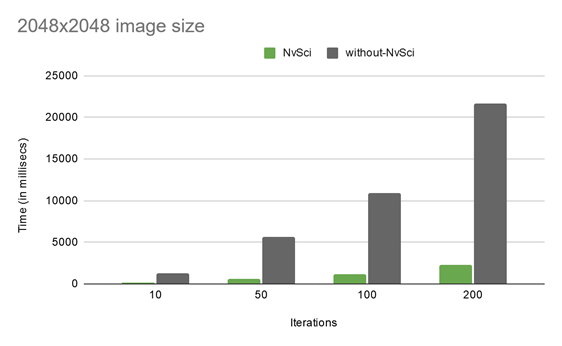
图 11 显示了使用此示例应用程序（使用 NvSci 代码路径和不使用 NvSci 代码路径）针对三种不同图像大小进行的性能测量：

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Comparision-with-and-without-NvSci.png)**

*图 11.与NvSci和不使用NvSci的比较。*

*NvMedia + CUDA 在 NVIDIA Xavier iGPU 上*

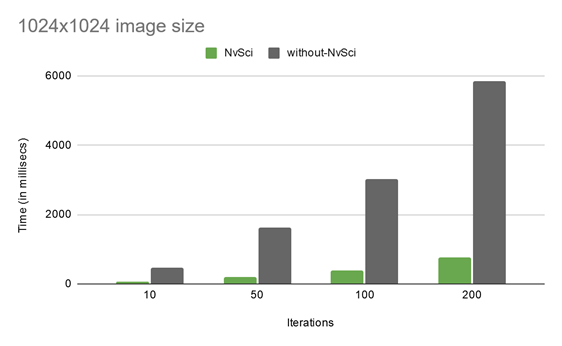
*（（n\*操作）+ 分配 + 清理）4096×4096 图像所花费的时间，其中 n = 迭代次数。*

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Comparision-with-and-without-NvSci-2048.png)**

*图 12.与NvSci和不使用NvSci的比较。*

*NvMedia + CUDA 在 NVIDIA Xavier iGPU 上*

*（（n\*操作）+ 分配 + 清理）2048×2048 图像所花费的时间，其中 n = 迭代次数。*

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Comparision-with-and-without-NvSci-1024.png)**

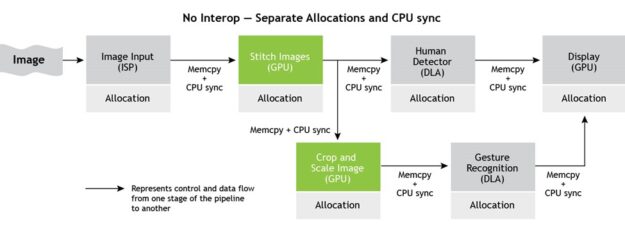
*图 13.与NvSci和不使用NvSci的比较。*

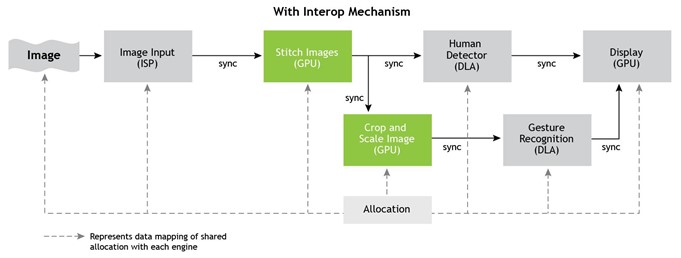
*NvMedia + CUDA 在 NVIDIA Xavier iGPU 上*

*（（n\*操作）+ 分配 + 清理）1024×1024 图像所花费的时间，其中 n = 迭代次数。*

从这些性能图表中可以明显看出，随着迭代操作数量的增加，无论图像大小如何，使用基于 NvSi 的互操作的性能优势都是巨大的。例如，对于 4096×4096 图像大小和 200 次迭代，与不使用 NvSci 相比，使用 NvSci 的性能提高了 4.45 倍。

图 14 显示了前面描述的手势识别算法如何利用 NvSci 互操作，允许在所有参与的硬件单元中使用单个分配。它使用 NvSciSync 定义控制流，以消除 CPU 端同步。

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Comparison-of-execution-with-and-without-interoperability-2.jpg)**

**[](https://developer-blogs.nvidia.com/wp-content/uploads/2020/11/Comparison-of-execution-with-and-without-interoperability-2.jpg)**

*图 14.比较具有和不具有互操作性的执行。*

**NvSciBuf 和 NvSciSync 的优势**

* **可移植性**：根据硬件和操作系统的不同，AGX 平台本机支持一个或多个同步基元，例如同步点、信号量、同步 FD 等。CUDA NvSciSync 互操作的用户从内部平台特定的详细信息中抽象出来。
* 可重用性：分配和预留后，互操作对象**可重用**，以减少资源占用。
* **多重广播**：多个 UMD 或实体可以等待单个互操作对象。此功能有助于避免创建多个对象来跟踪与原始互操作对象相同的任务。
* **高级软件抽象**：NvSciBuf和NvSciSync非常适合更高级别的软件抽象和框架，如NvStream，基于图形的执行框架，用户空间调度器，分析器等。
* **安全和安保**：NvSciBuf和NvSciSync旨在满足安全关键系统的严格要求：
  + 资源需求预测和预留。
  + 清除初始化和运行时 API 之间的分区。
  + 软件和硬件保护机制。
* 熟悉 x86 **和桌面：在 x86** 平台上构建的应用程序可移植到 AGX 平台，只需对应用程序进行最少的更改。

**结论**

CUDA 与 NvSciBuf 和 NvSciSync 的互操作提供了一种高性能、符合安全标准且可扩展的解决方案，可部署在自动驾驶汽车等安全关键领域。这些互操作允许应用程序共享资源并获得对有效描述跨越硬件边界的依赖关系的细粒度控制，包括整个芯片，例如 NVIDIA DRIVE AGX 平台上的 Tegra SoC A 和 Tegra SoC B，以及软件边界，例如线程、进程或虚拟机。

CUDA 与 NvSciBuf 或 NvSciSync 的互操作在任何 NVIDIA 平台上都受支持，其中 CUDA 外部资源互操作性可用，CUDA 10.2 及更高版本。鉴于当前版本的NvSciBuf和NvSciSync是为安全关键型汽车用例而设计的，因此它们在有限的平台上受支持。如果您使用的是以下任一平台，则可以立即试用此新互操作：

* x86/桌面（在 CUDA 10.2 中的 Ubuntu 18.04 上受支持）
* 嵌入式 Linux 和 QNX（VOLTA+ 架构）

随着即将发布的适用于 DRIVE AGX 的 NVIDIA DRIVE OS 5.2.0 版本，您可以探索这项新功能，因为它开箱即用，并直接受益于使用基于 NvSi 的互操作的性能改进。