CUDA 11.4 有什么新功能？ - 知乎

https://www.zhihu.com/question/477004197/answer/2037002910

NVIDIA英伟达中国已认证的官方帐号 Sat Aug 07 01:05

CUDA 11.4此版本包括 GPU – 加速库、调试和优化工具、编程语言增强功能，以及一个运行库，用于跨 CPU 主要体系结构（ x86 、 Arm 和 POWER ）在 GPU 上构建和部署应用程序。

CUDA 11 . 4 专注于增强 CUDA 应用程序的编程模型和性能。 CUDA 继续推动 GPU 加速的边界，并为 HPC 、图形、 CAE 应用、 AI 和深度学习、汽车、医疗和数据科学中的新应用奠定基础。

此版本引入了关键增强功能，以提高 CUDA 图形的性能，而无需对应用程序进行任何修改或任何其他用户干预。它还提高了多进程服务（ MPS ）的易用性。我们在 CUDA 编程指南中正式定义了 异步编程模型 。

CUDA 图

减少图形启动延迟是开发人员社区的常见要求，特别是在具有实时约束的应用程序中，如 5G 电信工作负载或 AI 推理工作负载。 CUDA 11 . 4 在减少 CUDA 图形启动时间方面提供了性能改进。此外，我们还集成了 CUDA 11 . 2 中引入的流顺序内存分配功能。

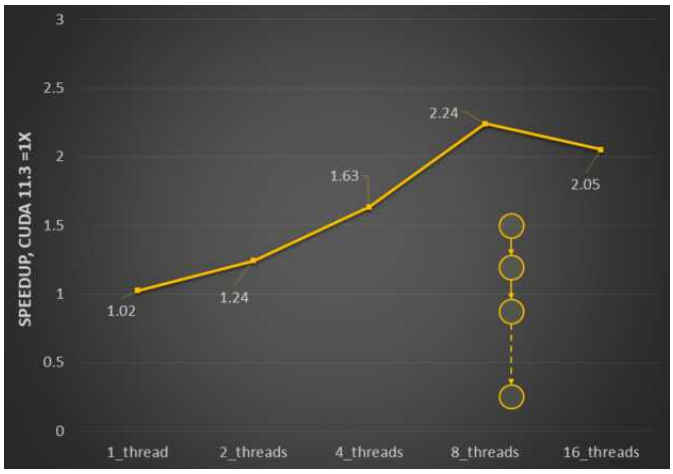
有关更多信息，请参阅 CUDA 工具包编程指南中的 CUDA 图 和 CUDA 图形入门 。

性能改进

CUDA 图形非常适合于多次执行的工作负载，因此，在为工作负载选择图形时，一个关键的折衷办法是将创建图形的成本分摊到重复启动上。重复次数或迭代次数越多，性能改进越大。

在 CUDA 11 . 4 中，我们对 CUDA 图形内部进行了两项关键更改，进一步提高了启动性能。 CUDA 图形已经避开了流，以实现更低的延迟运行时执行。我们对此进行了扩展，甚至在启动阶段也绕过了流，将图形作为单个工作块直接提交给硬件。对于单线程和多线程应用程序，我们已经看到了这些主机改进带来的良好性能提升。

多线程启动性能尤其受到并行启动多个图形时发生的资源争用的影响。我们已经优化了线程间锁定以减少争用，因此多线程启动现在效率显著提高。图 2 显示了 CUDA 11 . 4 中为缓解资源争用而进行的更改的相对性能优势，以及它如何随线程数扩展。



流顺序内存分配器支持

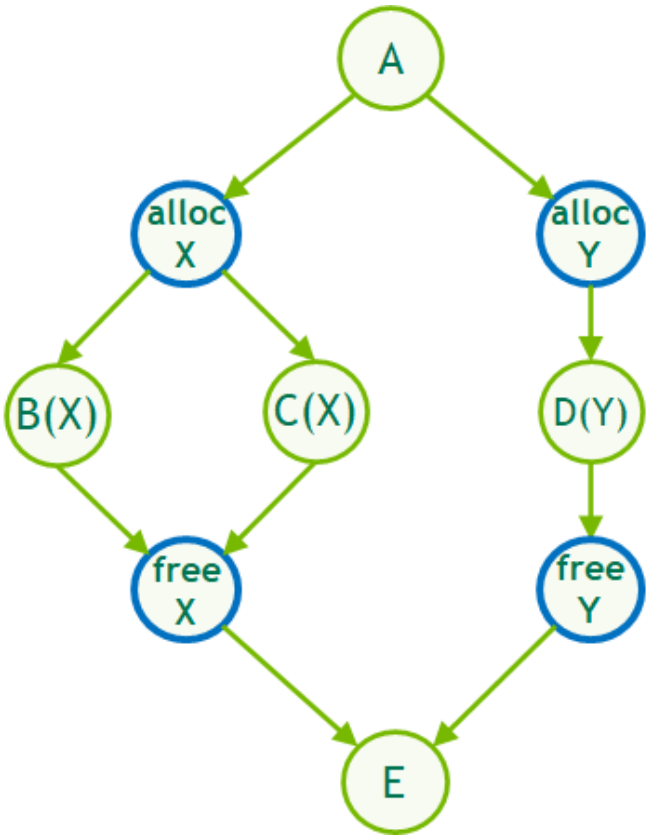
流排序内存分配器使应用程序能够针对 CUDA 流中启动的其他工作排序内存分配和释放。这还支持分配重用，这可以显著提高应用程序性能。有关该特性和功能的更多信息，请参阅 使用新的 NVIDIA CUDA 11 . 2 功能增强内存分配 。

在 CUDA 11 . 4 中， CUDA 图形现在通过流捕获或通过新的 `allocate` 和 `free node` 类型在本机图形构造中支持流顺序内存分配，从而在图形中实现相同的高效、延迟内存重用逻辑。

这些节点类型统称为内存节点。它们可以通过多种方式创建:

- 使用显式 API
  - 分别使用 `cudaGraphAddMemAllocNode` 和 `cudaGraphAddMemFreeNode`
- 使用流捕获
  - 分别使用 `cudaMallocAsync/cudaMallocFromPoolAsync` 和 `cudaFreeAsync`

与流顺序分配使用隐式流顺序和事件依赖来重用内存的方式相同，图顺序分配使用由图的边定义的依赖信息来执行相同的操作。



增强 MPS

多进程服务（MPS）是 CUDA API 的二进制兼容客户机 - 服务器运行时实现，旨在透明地支持协作多进程 CUDA 应用程序。

它由一个控制守护进程、客户端运行时和服务器进程组成。在单个进程不使用所有计算和内存带宽的情况下，MPS 可以实现更好的 GPU 利用率。MPS 还减少了 on- GPU 上下文存储和上下文切换。有关更多信息，请参阅 GPU 管理和部署指南中的 多进程服务 。

在这个版本中，我们做了几个关键的增强，以提高 MPS 的易用性。

SM 分区的编程配置

某些用例具有以下特点:

- 它们由很少或没有交互的内核组成，从而支持并发执行。
- 这些工作负载所需的 SMs 比率可能会发生变化，并且需要灵活分配正确数量的 SMs 。

MPS 活动线程百分比 设置允许您将执行限制到 SMs 的一部分。在 CUDA 11 . 4 之前，这是一个固定值，为流程中的所有客户机平等设置。在 CUDA 11 . 4 中，这已被扩展，以提供一种机制，通过编程接口在每个客户端级别对 SMs 进行分区。这使您能够在同一应用程序进程中创建具有不同 SM 分区的上下文。

名为 `CU_EXEC_AFFINITY_TYPE_SM_COUNT` 的新资源类型使您能够指定上下文所需的最小数量 *N* 。系统保证至少分配这么多的 SMs ，但可能会保留更多的 SMs 。 CUDA 11 . 4 还引入了相关的关联 API `cuCtxGetExecAffinity` ，该 API 查询为上下文分配的资源（如 SM 计数）的确切数量。有关更多信息，请参阅 API 文档中的 cuCtxGetExecAffinity 部分。

错误报告

为了改进错误报告和易于诊断 MPS 问题的根本原因，我们引入了新的详细的驱动程序和运行时错误代码。这些错误代码提供了有关错误类型的更多具体信息。它们用附加信息补充常见的 MPS 错误代码，以帮助您追踪故障原因。将应用程序中的这些错误代码与服务器日志中的错误消息一起使用，作为根本原因分析的一部分。

新的错误代码：

```
CUDA_ERROR_MPS_CONNECTION_FAILED
CUDA_ERROR_MPS_SERVER_NOT_READY
CUDA_ERROR_MPS_RPC_FAILURE
CUDA_ERROR_MPS_MAX_CLIENTS_REACHED
CUDA_ERROR_MPS_MAX_CONNECTIONS_REACHED
```

形式化异步数据移动

为了支持 CUDA 11 . 4 中 NVIDIA A100 C ++ 20 障碍 微体系结构启用的异步内存传输操作，我们对 异步 SIMT 编程模型 进行了形式化定义。异步编程模型定义了 GPU 上 C ++ 20 障碍 和 GPU 的行为和 API 。

有关如何使用异步 API 将全局内存中的内存操作与流式多处理器（ SMs ）中的计算重叠的更多信息，请参阅 异步 SIMT 编程模型 。

其他增强功能

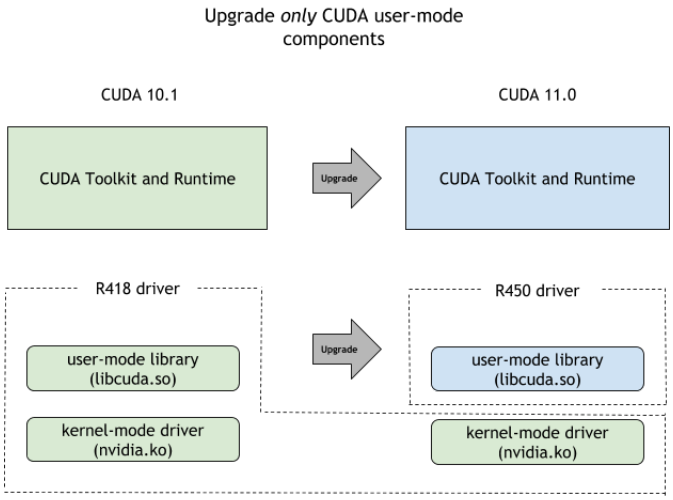
除了前面列出的关键功能外， CUDA 11 . 4 中还有一些增强功能，旨在提高多线程提交吞吐量，并将 CUDA 前向兼容性支持扩展到 NVIDIA RTX GPU s 。

多线程提交吞吐量

在 11 . 4 中，我们减少了 CPU 线程之间 CUDA API 的序列化开销。默认情况下，将启用这些更改。但是，为了帮助对潜在更改导致的可能问题进行分类，我们提供了一个环境变量 `CUDA_REDUCE_API_SERIALIZATION` ，以关闭这些更改。这是前面讨论的基础更改之一，有助于 CUDA 图的性能改进。

CUDA 前向兼容性

要启用希望更新 CUDA 工具包但仍保留当前驱动程序版本的用例，例如，为了降低转移到新驱动程序所需的额外验证的风险或开销， CUDA 提供了 CUDA 前向兼容路径 。这是在 CUDA 10 .0 中引入的，但最初仅限于数据中心 GPU s 。 CUDA 11 . 4 简化了这些限制，现在您也可以利用 NVIDIA RTX GPU 的前向兼容性路径。



C ++语言对 CUDA 的支持

以下是一些在 CUDA 11 . 4 中使用 C ++语言支持的关键增强。

- 主要版本:
  - NVIDIA C ++标准库（ LIbCu +++） 1 . 5 . 0 被 CUDA 11 . 4 发布。
  - 推力 1 . 12 . 0 具有新的 `thrust::universal_vector` API ，使您能够将 CUDA 统一内存与推力一起使用。

- Bug 修复版本： CUDA 11 . 4 工具包版本包括 CUB 1 . 12 . 0 。
- 添加了新的异步 `thrust::async:exclusive_scan` 和 `inclusive_scan` 算法，这些算法的同步版本已更新为直接使用 `cub::DeviceScan` 。

CUDA 编译器增强功能

CUDA 11 . 4 NVCC C ++编译器在预览中有 JIT LTO 支持，提供更多的 L1 和 L2 缓存控制，并公开了一个 C ++符号分散静态库以及 NVIDIA nVIEW 调试器支持 `alloca` 。

JIT 链路时间优化

JIT 链接时间优化（LTO）是一项预览功能，仅在 CUDA Toolkit 11 . 4 上可用，在嵌入式平台上不可用。此功能允许在运行时执行 LTO 。使用 NVRTC 生成 NVVM IR ，然后使用 `cuLink` 驱动程序 API 链接 NVVM IR 并执行 LTO 。

下面的代码示例显示如何在程序中使用运行时 JIT LTO 。
使用带有 `-dlto` 选项的 `nvrtcCompileProgram` 生成 NVVM IR ，并使用新引入的 `nvrtcGetNVVM` 检索生成的 NVVM IR 。现有的 `cuLink` API 被扩充，以采用新引入的 JIT LTO 选项，以接受 NVVM IR 作为输入并执行 JIT LTO 。将 `CU_JIT_LTO` 选项传递给 `cuLinkCreate` API 以实例化链接器，然后将 `CU_JIT_INPUT_NVVM` 用作 `cuLinkAddFile` 或 `cuLinkAddData` API 的选项以进一步链接 NVVM IR 。

```
nvrtcProgram prog1, prog2;
CUlinkState linkState;
int err;
void* cubin;
size_t cubinSize;
char *nvvmIR1, *nvvmIR2;


NVRTC_SAFE_CALL(
     nvrtcCompileProgram(&prog1, ...);
NVRTC_SAFE_CALL(
     nvrtcCompileProgram(&prog2, ...);

const char* opts = (“--gpu-architecture=compute_80”, “--dlto”);

nvrtcGetNVVM(prog1, &nvvmIR1);
nvrtcGetNVVM(prog1, &nvvmIR2);


options[0] = CU_JIT_LTO;
values[0] = (void*)&walltime;
...
cuLinkCreate(..., options, values, &linkState);
err = cuLinkAddData(linkState, CU_JIT_INPUT_NVVM,
                    (void*)nvvmIR1, strlen(nvvmIR1) + 1, ...);
...
err = cuLinkAddData(linkState, CU_JIT_INPUT_NVVM,
                    (void*)nvvmIR2, strlen(nvvmIR2) + 1, ...);
...
cuLinkComplete(linkState, &cubin, &cubinSize);
...
```

Libcu ++ flt 库支持

CUDA SDK 现在使用 LIbCu +++ FILT 来传输，这是一个静态的库，它将编译器损坏的 C ++符号转换成用户可读的名称。 `nv_decode.h` 头文件中的以下 API 是该库的入口点：

```
char* __cu_demangle(const char* id, char *output_buffer, size_t *length, int *status)
```

下面的 C ++示例代码显示用法：

```
#include <iostream>
#include '/usr/local/cuda-14.0/bin/nv_decode.h'

using namespace std;
int main(int argc, char **argv)
{
  const char* mangled_name = '_ZN6Scope15Func1Enez';
  int status = 1;
  char* w = __cu_demangle(mangled_name,0,0,&status);
  if(status != 0)
    cout<<'Demangling failed for: '<<mangled_name<<endl<<'Status: '<<status<<endl;
  else
    cout<<'Demangling Succeeded: '<<w<<endl;
}
```

此代码示例输出如下：

```
Demangling Succeeded: Scope1::Func1(__int128, long double, ...)
Demangling Succeeded: Scope1::Func1(__int128, long double, ...)
```

有关更多信息，请参阅 CUDA 二进制实用程序文档中的 Library 可用性 。

在 PTX 中配置缓存行为

PTX ISA 7 . 4 使您能够更好地控制一级缓存和二级缓存的缓存行为。本 PTX ISA 版本中引入了以下功能：

- **增强的数据预取：** 新的 `.level::prefetch_size` 限定符可用于预取附加数据以及内存加载或存储操作。这使得能够利用数据的空间局部性。
- **驱逐优先权控制：** PTX ISA 7 . 4 引入了四种缓存逐出优先级。可以在内存加载或存储操作（适用于一级缓存）和 `prefetch` 指令（适用于二级缓存）上使用 `.level::eviction_priority` 限定符指定这些逐出优先级。
  - `evict_normal` （默认值）
  - `evict_last` （数据应在缓存中保留更长时间时有用）
  - `evict_first` （用于流式传输数据）

- `no_allocate`（完全避免缓存数据）

- **增强的二级缓存控制：** 这有两种口味：
  - **特定地址上的缓存控制：** 新的 discard 指令允许从缓存中丢弃数据，而无需将其写回内存。仅当不再需要数据时才应使用。新的 `applypriority` 指令将特定数据的逐出优先级设置为 `evict_normal` 。当数据不再需要在缓存中持久化时，这对于从 [articularly useful in downgrading the eviction priority from `evict_last` 降级逐出优先级非常有用。
  - **内存操作的缓存提示：** 新的 `createpolicy` 指令允许创建缓存策略描述符，该描述符为不同的数据区域编码一个或多个缓存收回优先级。使用 `.level::cache_hint` 限定符时，一些内存操作（包括加载、存储、异步复制、 atom 、 red 等）可以接受缓存策略描述符作为操作数。

这些扩展仅被视为性能提示。缓存系统不保证使用这些扩展指定的缓存行为。有关用法的更多信息，请参阅 PTX ISA 规范 。

调用堆栈 11 . 4 中的其他编译器增强功能包括支持新的主机编译器： ICC 2021 。 CUDA 前端编译器发出的诊断现在是 ANSI 颜色的， Nsight debugger 现在可以在 CUDA 视图中通过 `alloca` 调用正确展开 CUDA 应用程序。

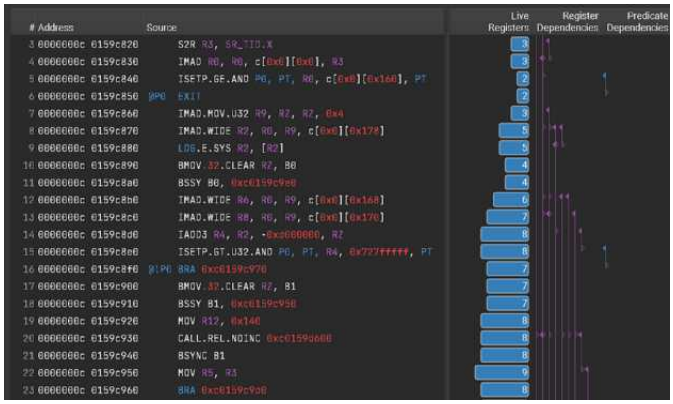
Nsight 开发工具

NVIDIA Nsight Visual Studio 代码版（ VSCE ）和 Nsight Compute 2021 . 2 现在提供了新版本，为 CUDA 编程的开发人员体验增加了增强功能。

NVIDIA Nsight VSCE 是一种针对异构平台的应用程序开发环境，将 CUDA 对 GPU 的开发纳入 Microsoft Visual Studio 代码中。 NVIDIA Nsight VSCE 使您能够在同一会话中构建和调试 GPU 内核和本机 CPU 代码，并检查 GPU 和内存的状态。

它包括 CUDA 应用程序的 IntelliSense 代码突出显示和 IDE 的集成 GPU 调试体验，支持单步执行代码、设置断点以及检查 CUDA 内核中的内存状态和系统信息。现在，直接从 VisualStudio 代码开发和调试 CUDA 应用程序很容易。

Nsight Compute 2021 . 2 添加了新功能，有助于检测更多性能问题，并使其更易于理解和修复。新的寄存器依赖关系可视化（图 6 ）有助于识别可能限制性能的长依赖链和低效的寄存器使用。此版本还添加了一个经常请求的功能，使您能够在源代码视图中查看 CUDA 内核的并行程序集和相关源代码，而无需收集概要文件。此独立源代码查看器功能允许您直接从 GUI 中的磁盘打开. cubin 文件，以查看代码相关性。



一些功能，包括突出显示的焦点指标、报告交叉链接、增强的规则可见性和文档参考，都添加到了 Nsight Compute 提供的内置概要文件和优化指导分析中，以帮助您了解和修复性能瓶颈。

此版本还包括对 OptiX 7 资源跟踪的支持，用于读取报告数据的新 Python 接口，以及对基线报告、字体设置和 CLI 过滤器的管理的改进。