Tensor 所在设备

注解

- 通常用图形处理单元（GPU）代替中央处理单元（CPU）作为训练时的主要计算设备。 （ 解释 ）
- 默认情况下，MegEngine 会自动使用当前可用的最快设备（xpux）， **无需额外进行人为的指定。**

```
>>> megengine.get_default_device()
'xpux'
```

其中 xpu 表示 gpu 或者 cpu, 后面的 x 表示编号（如果有多个设备），默认从 0 开始。

在未检测到 GPU 设备的机器上，MegEngine 首次生成 Tensor 时，将进行一次提醒，如下所示:

```
>>> import megengine
>>> a = megengine.Tensor([1., 2., 3.])
info: ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
info: + Failed to load CUDA driver library, MegEngine works under CPU mode now.      +
info: + To use CUDA mode, please make sure NVIDIA GPU driver was installed properly. +
info: + Refer to https://discuss.megengine.org.cn/t/topic/1264 for more information. +
info: ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
debug: failed to load cuda func: cuDeviceGetCount
debug: failed to load cuda func: cuDeviceGetCount
debug: failed to load cuda func: cuGetErrorString
>>> a.device
CompNode("cpu0:0" from "xpux:0")
```

对于日常的 MegEngine 使用情景，我们不需要关注冒号 : 后面编号的含义。'

设备相关接口

以下是比较常用的几个接口:

- 我们可以通过 **get_default_device** 获得默认的计算节点;
- 我们可以通过 **set_default_device** 设置默认的计算节点;
- 如果想要将 Tensor 拷贝到指定的计算设备，可以使用 **copy** .

借助这些接口，我们可以有选择地在 CPU 上或 GPU 上进行 Tensor 计算。比如指定为 CPU:

```
>>> import megengine
>>> megengine.set_default_device("cpux")
>>> megengine.get_default_device()
'cpux'
>>> a = megengine.Tensor([1., 2., 3.])
>>> a.device
CompNode("cpu0:0" from "cpux:0")
```

由于指定了设备为 CPU, 则不会出现加载 CUDA 驱动失败的提醒。

参见

- 可在 Device 页面找到所有可调用的 API;
- 与设备相关的概念还有： 分布式训练（Distributed Training） 。

支持 GPU 设备和软件平台

想要在 MegEngine 计算时利用 GPU 设备，用户无需进行额外的编码。框架将在底层替用户进行主流 GPU 软件平台接口的调用。 因此用户可以专注于神经网络结构的设计，选择相信由框架在背后完成的性能优化工作。 但如果一名用户想要成为 MegEngine 核心开发者或拓展 MegEngine 功能，则需要了解相关背景知识。

警告

MegEngine 默认支持当前主流的 Nvidia GPU 设备 （Compute Capability 5.2~8.0），如果你的 Nvidia GPU 设备不在支持的 Compute Capability 范围内， 或需要使用支持 AMD GPU 的 MegEngine, 则需要 通过源码编译安装 ，否则会触发即时编译，或直接报错。

参见

感兴趣的用户可阅读下面的解释，略过这些部分的阅读不会影响 MegEngine 的基础使用。

Nvidia GPU 和 CUDA

Nvidia 是一家设计 GPU 的技术公司，他们创建了 CUDA 软件平台与自家的 GPU 硬件配对， 使开发人员可以更轻松地构建使用 Nvidia GPU 的并行处理能力加速计算的软件。 即 Nvidia GPU 是支持并行计算的硬件，而 CUDA 是为开发人员提供 API 的软件层。 开发人员通过下载 CUDA 工具包来使用它，该工具包附带了专门的库，如 cuDNN, 即 CUDA 深度神经网络库。

> **参见**
>
> - 可以通过 NVIDIA System Management Interface (nvidia-smi) 帮助管理和监控 NVIDIA GPU 设备；
> - 可以使用环境变量 `CUDA_VISIBLE_DEVICES` 来限制 CUDA 应用程序看到的设备。 （官方博客）

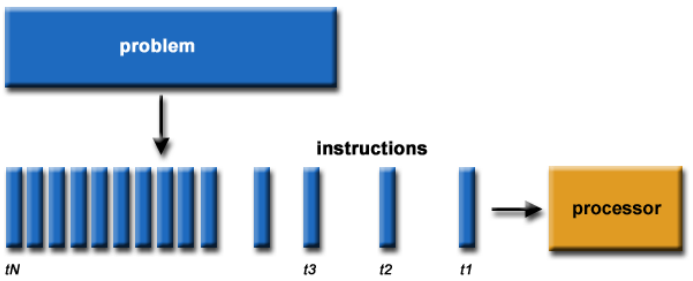
AMD GPU 和 ROCm

Advanced Micro Devices （AMD） 是一家半导体公司，主要产品包括微处理器、主板芯片组、嵌入式处理器和图形处理器。 他们提供了 ROCm 软件平台与自家的 GPU 硬件配对，其 API 设计与 CUDA 十分类似。

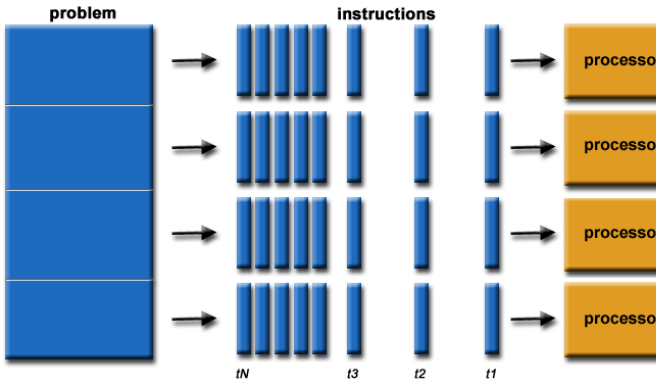
为何需要使用 GPU 训练?

在回答这个问题前，我们需要了解什么是 并行计算 （Parallel computing）—— 并行计算是一种计算类型，可将其中的计算分解成能够同时进行的较小独立计算，然后将计算结果进行重新组合或同步，得到原始计算的结果。

串行计算



并行计算



图形计算单元 （Graphics processing unit, GPU） 是一种擅长处理特定（Specialized）类型计算的装置，而 中央处理单元 （Central processing unit , CPU） 被设计用来处理一般（General）的计算。虽然 CPU 能够胜任各种复杂的计算操作情景， 但 GPU 高度并行的结构设计使它们在处理并行计算时比 CPU 更加高效。

一个更大的任务可以分解成的任务数量也取决于特定硬件上包含的核心（Kernel）数量。 核心是在给定处理器内实际执行计算的单元，CPU 通常有四个、八个或十六个内核，而 GPU 可能有数千个。

因此我们可以得出结论：

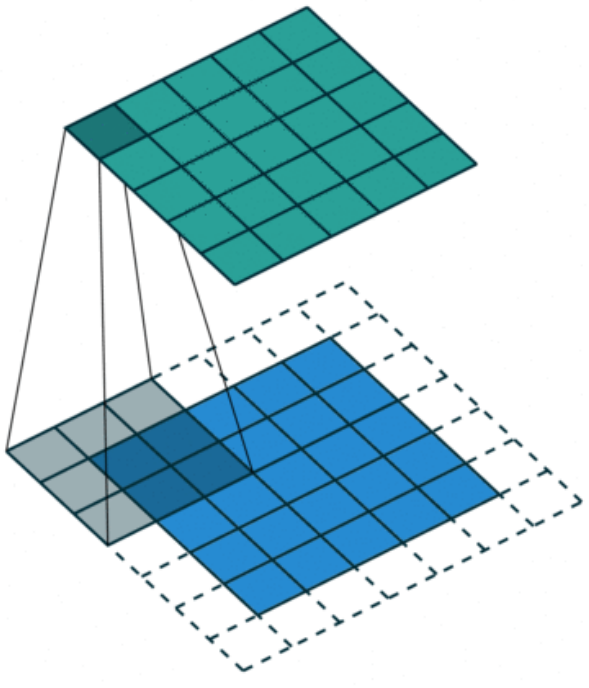
- 最适合使用 GPU 解决的任务是可以并行完成的任务。
- 如果计算可以并行完成，我们就可以使用并行编程方法和 GPU 来加速我们的计算。

> ⚠ **使用 GPU 不一定会更快!**
>
> GPU 可以很好地处理能够分解为许多更小的任务的任务，但如果计算任务已经很小，那么将任务移到 GPU 上可能不会有太多收益。 因此将相对较小的计算任务转移到 GPU 不一定能获得显著的提速，甚至有可能变慢。
>
> 另外，将数据从 CPU 移动到 GPU 的成本很高，如果计算任务很简单，整体速度反而可能变慢。

神经网络计算中的并行性

在神经网络中存在着大量的可并行计算任务，其中一些类型属于 Embarrassingly parallel , 即各个独立的线程之间都表现得很难为情，不愿意和其它线程进行交流。 实际上它描述的是各个线程在不进行交流的情况下，也能够独立地完成并行计算任务。 从语义上看，这样的并行计算是容易的、完美的、甚至令人愉悦的。

一个典型的例子是 —— 卷积（Convolution）运算。



以上图为例子，图中的蓝色部分（底部）表示输入通道，蓝色部分上的阴影表示 3×3 卷积核， 绿色部分（顶部）表示输出通道。对于蓝色输入通道上的每个位置，都会进行卷积运算， 即将蓝色输入通道的阴影部分映射到绿色输出通道的相应阴影部分。

- 这些计算一个接一个地依次发生，但每个计算都独立于其它计算，即不依赖于其它计算的结果；
- 因此所有这些独立的计算都可以在 GPU 上并行地进行，最终生成整个输出通道。

GPGPU 计算

GPU 一开始被用来加速计算机图形学中的特定计算，因此被命名为 “图形处理单元”。 但近年来，出现了更多种类的并行任务。正如我们所见，其中一项任务是深度学习。 深度学习以及许多其他使用并行编程技术的科学计算任务正在催生一种称为通用 GPU 计算 （general purpose GPU computing , GPGPU）的新型编程模型。

> **注解**
>
> GPGPU 计算更常被称为 GPU 计算或加速计算，因为在 GPU 上执行各种任务变得越来越普遍。

上一页
Tensor 数据类型

下一页
Tensor 具象化举例