为什么国内的AI芯片公司不去支持CUDA，是技术问题还是版权问题？ - 知乎

https://www.zhihu.com/question/461354739/answer/1964488472

cloudcore研究算法，软硬件协同优化啥的。 Sun Jun, 27 15:47

首先，实现真正意义上的完全兼容是极不可能的。最多是常用API和功能类似，减少用户移植成本。CUDA本身涵盖的功能非常广泛，硬件功能上就几乎体现了NV自家GPGPU的所有可能性，再加上驱动和软件上层封装（各种库，比如cuBLAS，cuFFT，cuDNN之类），以及完备的开发工具套件（编译器、调试器、profiler等等）。这些东西就算是全部开源，让各家移植支持自己的硬件，多数公司恐怕也是有心无力。更别说CUDA很多功能与硬件深度耦合，硬件设计不一致，靠软件封装来保持一致性，工作量真不是一般公司消化得了的。更何况其中很多东西并不公开，各家无从下手，功能就更难做得一致了。这还只是功能的一致性问题，性能上就更难保证了。李逵和李鬼，靠长的像是不行的，抡起板斧来就露相了……

其实就算是NV自己，各代硬件之间发生较大的功能改动时，更新工作量也不小。NV在硬件微架构和指令集上迭代很快，几乎每隔一两代就会有较大的功能变化。这样底层很多东西都要跟着调整。NV的PTX是一个很好的隔离机制，底层指令集之类的改动多数可以在PTX这层兼容，这样上层就不用动了。不过，底层仍然有PTX覆盖不到的改动（主要是运行逻辑的改变，比如Independent Thread Scheduling这种，同样的PTX代码在不同架构上行为不一致）。或者是有些程序没有内嵌PTX文件，那也没法兼容。而驱动或上层编程接口之类的改动跟PTX就没啥关系，当然也没法靠PTX覆盖。

老黄曾说NV是软件公司，也不是随便说说。至少CUDA的各种功能，并不都是跟着硬件版本走，很多都是软件层的封装。新硬件出来，CUDA一般会发个大版本用以提供相应支持。但CUDA自身软件层封装的功能也会不断新增和改进，也会有相应的版本（比如11.1，11.2之类）。只不过软件层的向后兼容可以做的比较好，用户通常不太关注而已。

这些硬件或软件上不同版本的差异，多多少少都会影响到用户的使用。所以很多软件包括AI框架甚至都要安装对应的CUDA版本，否则就可能出错。你说NV自家都不能完全兼容，外人还想兼容，那就太难了。这些对NV也是沉重的负担，感觉NV应该是在有意识的压缩产品支持周期，比如SM50（Maxwell架构）是14年首发，15、16年甚至再往后都有很多卡还在卖的。可是去年的CUDA 11.2版本已经把它deprecated了，这周期也就四五年而已……

其次，API这个东西，应该是没有版权的。之前Google与Oracle在Java的API版权上打过旷日持久的官司，去年最终裁决结果认为API本身是没有版权的。当然这应该说的是API的命名和总体结构设计没有版权，具体API的内部实现可能还是有的。实际上各家API的“借鉴”其实非常普遍，普遍到大家都习以为常，见怪不怪了。

就说NV自己，首先CUDA的编程模型很大程度复用了C语言，所以有cudaMalloc、cudaFree、cudaMemset这种定制版的“C API”。CUDA里的数学函数也多数沿用了cmath里的形式，比如exp()是double函数，expf()是float版的exp。cuBLAS就不用说了，多数API都源自LAPACK（虽然BLAS的这套API现在都快成标准接口了）。cuFFT的API与FFTW虽然有不小的差别，但两者之间的传承关系也很明显。这些API基本都来自曾经非常流行的开源库，应该说还算是常规操作。

但这还没完。熟悉Intel开发工具的人应该知道Intel有MKL（Math Kernel Library）和IPP（Intel Integrated Performance Primitives）。MKL相当于Intel的BLAS+数学函数库，IPP主要是做图像和信号处理的，有ippi（image processing）和ipps（signal processing）。NVIDIA有个库叫NPP，也有nppi和npps，你说巧不巧…… Intel还有个很有名的并行库叫TBB，可以基于模板做并行transform、reduce、scan之类的泛型操作，还能做并行任务拆分和调度。CUDA有个库叫Thrust，也是基于模板的泛型编程，也可以做并行transform、reduce、scan…… 当然，TBB和Thrust在接口上差别还是很大的，而且TBB从功能和可编程性上讲比Thrust要完整得多。但你要说Thrust没借鉴过TBB，我反正是不信……

我这里也不是针对NV。其实绝大多数功能相似的库或软件产品，具有类似的API是再正常不过的事情。比如各种计算机代数系统（以matlab为代表），各种AI框架等等。编译器还会造接口去接受其他编译器的参数输入格式呢！只要你不是直接抄代码，接口类似，但内部实现有差别，其实也不是什么见不得人的事情。

最后，API只是个入口，里面也会有非常多的坑。你把自家API跟用户常用的主流API做得像，可以大大减少用户的学习成本和移植工作量，甚至一个文本替换脚本就能搞定大部分。但前面也说了，功能的差异是不可避免的。99%相同，1%不同，看起来好像还行。但实际用起来，99%相同的部分占用时间1%，那1%不同的部分埋下的坑可能会耗费你99%的时间…… 更狠的是有时候你根本不知道到底是哪1%不一样……

所以我觉得API的复用，还是要格外慎重。有很大把握做到一致的可以复用，有差别的还是尽量区分开，否则真的是遗患无穷。有些公司意识不太好，老想着开始先蒙混过关，假装一样，将来有机会再慢慢修补。其实这是非常短视的做法，用户谁没事老关注你各个版本什么变化，一不小心踩坑真是心累又心碎。这种交付就非常不靠谱，严重不推荐。

其实用户最常用的CUDA核心功能也没有太多，能把这些支持好就不错。至少可以覆盖多数用户的多数需求。当前大部分的AI硬件公司应该都没啥2C业务，客户支持的压力会小很多。毕竟如果是2B的话，用户水准一般都还可以，做一些定制和差异化，能有自己的一些优势，也能保证先活下去。而且现在上层框架为了兼容性，接口一般不会对CUDA做那么深的定制。有这些基础功能和通用接口支撑，多数功能移植起来应该也没那么难。这些问题很多都不是技术难度问题，更多的是工作量的问题。CUDA里有些复杂功能，实在不支持就放弃算了。有些市场，吃不下就不要硬啃，真的划不来……

生态也不都是靠用的人多堆出来的，技术先进性和技术发展方向的话语权也很重要。花这么多功夫去做兼容和移植，还不如多研究研究怎么把自己的技术优势发挥出来。只要你有足够的竞争力，你在哪，哪里就有生态~