我是如何进行异构并行计算算法设计的

作者 风辰

当 2011 年我从中科院毕业的时候，虽然当时已经成为了 GPU 并行计算领域小有名气的人物，但是我当时的心境非常不安，因为无论是在 51job，还是在智联招聘上都搜索不到 GPU 并行计算相关的职位，甚至连并行计算相关的职位也查找不到，“毕业即失业”成为隐忧。而在写作本文时，我在 51job 和智联招聘上搜索“GPU”或 CUDA 时，弹出来的职位数量和内容已经非常丰富了，一些企业也开始招聘 X86 或 ARM 代码优化人员。而招聘的公司不但有百度、腾讯等一线公司，也有一些民营企业，而更多的是一些初创企业在寻找相关方向的人才。

我想我的经历是异构并行计算在中国乃至世界发展的一个缩影：异构并行计算已经从实验室里教授们的研究对象变成了真正提高企业生产力的重要工具。

作为一个异构并行计算这一浪潮的重度参与者，我想我有必要和大家分享我的异构并行计算算法设计方法：希望更多的读者从我的方法中借鉴与掌握应该如何设计异构并行计算算法，更希望我的方法能够促进异构并行计算在中国的进一步发展。

1. 异构并行计算平台的差异

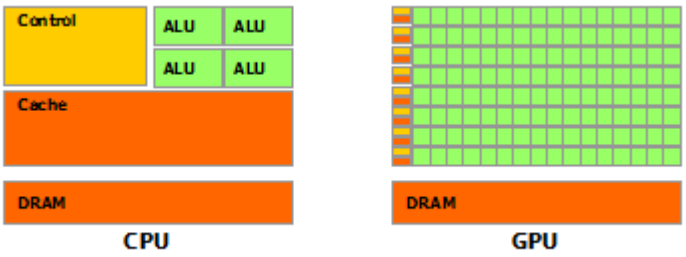
在不同的应用场景下，需要满足不同要求的处理器，才能够获得最高的计算性能。在 2008 年以前，Intel 希望通过其 X86 处理器架构一统天下，把其在桌面和服务器领域的成功复制到移动和嵌入式，以及图形处理。事实证明其策略是失败的：在移动处理器上，Intel 出过几款处理器，也被联想等企业应用到手机中，但是市场并不买账，而今天移动处理器市场上基本上没有 Intel 的足迹；在嵌入式上面，Intel 最近推出了爱迪生芯片，虽然爱迪生基于 Intel X86 架构，但是依据嵌入式应用的需求做了许多改进，但是市场依旧在观望；在图形处理器方面，Intel 分别针对图形图像处理和通用计算推出了 GEN 架构和 MIC 架构，而无论是 GEN 架构和 MIC 架构都和传统的 X86 架构有本质区别，更明确地说：GEN 和 MIC 已经不是 X86 处理器了，而是纯正的 GPU。Intel 的这些行为充分证明了 Intel 一直不愿意承认的异构并行计算的核心理念：不同的架构设计的处理器具有不同的特点，而不同的应用也具有不同的特点，应当为不同特点的应用使用不同的处理器，使用一种处理器架构满足各个不同市场的需求是痴心妄想。

为了提高系统的性能，则必须要把应用的特点和处理器的特性相互配合，这就是协同设计。从应用的特点来看，不同的应用具有不同的需求：有的应用需要大量的访问数据；有的应用局部性很好，而有的应用局部性又很差。从不同的处理器的特点来看，不同的处理器适合做不同的事情：如 X86 处理器为进行延迟优化，以减少指令的执行延迟为主要设计考量（当然今天的 X86 处理器设计中也有许多以吞吐量作为设计考量的影子），编程相对简单，使用了大量的缓存，因此应用易于达到硬件的峰值性能；如 NVIDIA GPU 和 AMD GPU 则以以提高整个硬件的吞吐量为主要设计目标，在编程理念和方法上与 X86 体系具有许多不同。

异构并行计算的理论和实践涉及到许多不同的内容，本节只是简介了其中比较本质内容。笔者所著的《并行算法设计与性能优化》关注并行优化和并行计算相关的基础理论、算法设计与高层次的实践经验。这本书是笔者多年经验的积累，填补了国内代码性能优化和并行计算的空白，希望对广大想要了解代码性能优化和并行化背后的理论基础的读者有所帮助。

2. 异构并行算法的设计方面

同一算法在不同的处理器上具有不同的性能瓶颈，这主要是因为不同的处理器的设计理念不同，应用场景也有区别，因此将不同比例的晶体管用在了不同的单元上。比如下图展示了 Intel X86 处理器和 NVIDIA GPU 的设计理念差异。



Intel X86 处理器将大量的晶体管用于缓存和逻辑控制，因此真正用于计算的晶体管数量反而偏少。而 NVIDIA GPU 只将很少的晶体管用于逻辑控制，用于缓存的比例则更少，因此能够用于计算的比例就比较高。这两种设计使得在同样数量的晶体管规模下，NVIDIA GPU 所能够获得的峰值计算能力远大于 X86。由于大量的晶体管用于缓存和控制逻辑，在 X86 上编程相对容易，简单的代码往往也能获得比较好的性能，降低了为了获得更高性能所需要的编程难度。由于不同处理器擅长做不同的事情，因此对于某个固定的算法也需要选择合适的硬件和合适的编程模型以将算法很好地映射到硬件上执行。

对于某个具体算法的异构并行化过程，可以分成以下几个环节：

- 为算法选择合适的硬件和编程模型，如前面所述：硬件具有不同的特性，而应用的各个部分也有不同的特点，如何选择合适的异构并行平台和编程模型以将算法的各个部分和硬件平台相匹配。
- 将算法的不同部分划分给异构并行平台上多个处理器计算，异构平台上通常具有多种不同的处理器，每个处理器擅长的任务通常也不相同，如 CPU+GPU 的异构平台上，CPU 适合处理逻辑复杂的计算，而 GPU 适合处理数据并行的计算密集型运算，如何将算法的各个部分分配给不同的处理器以获得更好的性能使我们需要考虑的一个问题。
- 将分配给某个处理器的计算高效地映射到处理器硬件上，对于异构平台上每个处理器来说，要发挥其峰值性能需要的编程方法可能并不相同，要尽量将算法和硬件的架构细节相协调，以充分发挥硬件的性能。
- 对算法的实现进行性能分析，如果性能不如预期，分析性能瓶颈产生在什么地方，然后依据性能瓶颈优化算法或实现，返回 1,2,3 修改算法设计细节，进一步提升性能，周而复始，直到算法性能达到极致。

（1）如何选择合适的硬件和编程模型

目前市场上的主流处理器厂商有 Intel、AMD、NVIDIA 和 ARM 等，其中 Intel 和 AMD 生产服务器和桌面使用的 CPU，而 AMD 和 NVIDIA 生产桌面和服务器使用的 GPU，至于 ARM 则主要是授权移动 CPU 和 GPU 知识产权。从硬件架构上说，Intel 和 AMD 的 X86 处理器以及 ARM 的 CPU 具有许多共同点，一些性能优化思想和方法也基本类似，但是如果为了发挥硬件的峰值性能，在这几种处理器上编程还是有非常巨大的区别，比如 Intel X86 处理器在寄存器重命名机制和乱序执行能力上就比 AMD X86 处理器和 ARM CPU 要强很多。对 NVIDIA GPU 和 AMD GCN（Generation Core Next）GPU 来说，大多数优化方式都通用，在 NVIDIA GPU 上的优化在 AMD GPU 上也能够获得高性能。

目前可以用于异构并行计算的编程环境主要是 CUDA、OpenCL 和 OpenMP，其中 CUDA 由 NVIDIA 开发，经过多年发展已经比较稳定，是在 NVIDIA 硬件平台上进行异构计算的首选。OpenCL 是一个开放的标准，已经获得了包括 NVIDIA、AMD、Intel 和 ARM 在内的许多硬件巨头的支持。为了支持更广泛的处理器，OpenCL 编程相比 CUDA 要繁琐一些。OpenCL 和 CUDA 的许多概念非常类似，一些完全可以互换使用，因此在两者之间转换并不难，一些公司也开

发了源到源的编译器以支持将 CUDA 代码转化成 OpenCL 代码。最新的 OpenMP 4 标准已经加入了异构并行计算的支持，一些编译器厂商正在试图使得其编译器满足 OpenMP 4 标准，相信不久之后，GCC 等开源的编译器都会加入 OpenMP 4 的支持，这样一份异构并行计算代码就可以同时在 Intel、AMD 和 NVIDIA 的处理器上编译运行，虽然 OpenMP 可能不能在某些异构处理器获得高性能，但是 OpenCL 和 OpenMP 混合使用能够解决这个问题。

为了商业利益，不同的硬件厂商的库和编程工具之间并不兼容，市场上也很少有相关的参考著作。笔者所著的《并行编程方法与优化实践》关注于 C 程序设计语言的向量化和并行化扩展及算法到硬件的映射。此书介绍了常见的 SSE/AVX/NEON SIMD 指令集编程，用于 GPU 的异构并行编程语言 CUDA、OpenCL 和 OpenACC，以及常见的用于多核编程的 OpenMP 标准。并且以稠密矩阵运算和图像处理为例，介绍了如何使用这些工具优化程序性能。想要学习这些工具的朋友，或因为项目需要学习这些工具尽快上手的同学可以参考此书。

（2）如何在异构平台上进行任务分配

对于一个算法的不同部分，需要选择使用最优的处理器来处理。如果算法的一部分包含有许多逻辑处理，那么最好将这部分运算交给 X86 处理；如果一部分包含许多数据并行处理，那么最好将这部分运算交给 GPU 处理；如果一部分代码对实时性要求非常高，使用 FPGA 处理可能是更好的选择。

在异构平台上进行任务分配的主要难度在于：除了要考虑各个处理器的计算性能、特点外，还需要考虑多个不同的处理器之间的交互对性能的影响。在 CPU+GPU 的平台上，通常 GPU 是通过 PCI-e 总线连接到主板上的，因此 CPU 和 GPU 之间的通信需要通过 PCI-e 总线进行，这就引入了一项额外的消耗，由于 PCI-e 总线的带宽远小于内存带宽（PCI-e 3 的实际带宽大约 12 GB/s，而内存带宽以达近 100 GB/s），这个消耗有的时候会成为算法性能的瓶颈。在一些情况下，将一些不太适合 GPU 的运算转移到 GPU 上可能会获得更好的性能，因为这样潜在地降低了在 CPU 和 GPU 之间进行数据传输的需求。

在异构平台上进行任务分配是一个比较复杂的问题，这更多的是一个知识、技术和经验相互结合的难题。

（3）任务映射

如何将算法中的计算映射到硬件上，以便获得更好的性能，这有许多方面需要考虑，在不同的硬件中，这些考虑有相同点也有不同点，笔者以在大 GPU 集群中计算某个变量（这个变量可以是某个运算，比如计算矩阵乘法、从数据流中过滤一些数据等）为例，展示这些考虑。

① 在节点间，不同的节点需要通过网络来传输数据，通常网络带宽要比计算慢很多，故减少网络传输的损耗非常重要。常见的方法有：

- 把算法映射到节点上，以减少数据传输的次数和总传输的数据量；
- 数据传输的本地化：利用节点内的数据传输，减少节点间的数据传输；
- 合并小的数据传输为大的数据传输，以减少总的数据传输次数；
- 在数据传输的同时进行计算以掩盖节点间数据传输的消耗；
- 提高节点间的数据传输带宽，比如使用性能更高的 Infiniband 网卡和交换机。

② 在节点内，多个 GPU 计算时可能需要相互传输数据，这需要通过 PCI-e 总线进行数据传输，NVIDIA GPU 支持 GPUDirect P2P 技术，通过这种技术在两个 GPU 之间传输数据时，CPU 无须参与，GPU 之间的数据交换也无须通过内存中转。GPUDirect P2P 技术需要主板硬件的支持，因此各个 GPU 如何连接到主板上也非常重要。在多个 GPU 间进行计算的同时，通过 PCI-e 总线进行数据传输，如果数据传输时间小于计算时间，数据传输时间就可以被掩盖。

③ 单个 GPU 内，如何合理地安排计算和存储器访问，以最大程度地发挥计算单元和存储器访问单元的效率。比如如何满足全局存储器的合并访问要求；如何使用共享存储器访问减少全局存储器访问次数；如何合理安排指令顺序，以减少寄存器延迟的影响。在单个 GPU 内，也需要在计算前通过 PCI-e 总线将数据传到 GPU 上，在计算完成后将数据传回 CPU，一些 GPU 具备多个数据拷贝引擎，能够同时进行从 CPU 到 GPU 的数据传输和从 GPU 到 CPU 的数据传输。

在 GPU 内的单个流多处理器内，如何使得流多处理器上具有足够的线程同时在执行，重叠访存和计算，以计算掩盖访存的延迟，这更考验软件开发人员的智慧、经验和知识。

从流多处理器到集群中的多个 GPU，这是一个从小到大的层次，在实际工作中，笔者通常进行从小到大的设计，在小的层次上解决问题，再将算法扩展到大的，解决问题后再向更大的层次扩展。

（4）性能分析

在某个具体的硬件上实现算法后，需要评估其性能，找到其性能瓶颈，找到其性能瓶颈后，再重新审视算法的设计，以确定是否需要更改算法以更好地映射到硬件上，以去除性能瓶颈，周而复始以使得算法的性能达到最优。在一些情况下，评估性能后可能发现算法并不适合在当前的异构平台上执行，此时可能需要换一种异构平台配置。

无论是 Intel、AMD，还是 NVIDIA，它们都提供了丰富的软件来分析运行在其硬件上的程序性能，比如 Intel 的 Vtune 工具，AMD 的 CodeXL 工具和 NVIDIA 的 Nsight 工具，关于这些工具笔者就不介绍了，感兴趣的读者可以参考官方介绍或手册。

3. 异构并行计算的应用场景

异构并行已经走出实验室，正在产业界得到越来越广泛的应用，除了超算中心的传统科学计算之外，异构并行计算还在油气勘探、图像处理、大数据处理和深度学习领域获得重视，并且还在越来越多的领域生根发芽。

图像处理是 GPU 的传统优势领域，由于大多数图像处理算法具有数据并行的特征，因此非常适合向量处理器进行向量处理。目前 X86、ARM 和 GPU 都已经广泛应用在图像处理领域。

在油气勘探领域，超级计算机被大量用来模拟地下物理结构，以推测是否有石油或天然气。油气勘探领域的主要计算是偏微分方程求解和傅立叶变换。对于偏微分方程求解，通过差分方法可转化成两类问题：一类是递推迭代；一类是线性方程求解。由于需要模拟大区域的情况和差分方法的运用，通常这两类问题都具有非常好的并行性，非常适合 CPU+GPU 的异构平台。

在深度学习应用领域，现在开源的深度学习平台（如 caffe, theano）都支持 GPU 的训练，如 Google、Facebook、百度、腾讯、阿里都在开发各自的深度学习平台。可以毫不讳言地说：GPU 硬件已经是深度学习训练平台的标准配置。

大数据处理中需要从海量数据中寻找特殊的模式，这也是异构并行计算的用武之地。许多大数据创业公司使用 GPU 来分析数据，从中找出有用的信息。

笔者所著的《科学计算与企业级应用的并行优化》介绍了如何将线性代数、偏微分方程求解、分子动力学和机器学习领域的常见算法良好地在 X86 和 GPU 平台上实现出来，希望

能够帮助从事这些领域的朋友提高职业技能。