基于多线程与并行计算的研究

计试 71 施膛骜 2150506098
计试 61 姜伟

项目引言：

相对于串行计算来说，并行计算可同时执行多个指令。在空间上，并行计算采用多个处理器并发的执行；在时间上，并行采用流水线技术。通过提高计算速度和扩大问题求解规模，并行计算得以解决大型而复杂的计算问题。并行计算的基本思想是用多个处理器来协同求解同一问题，即将被求解的问题分解成若干部分，每一部分由独立的处理单元进行运算。并行计算系统可以是专门设计的，含有超多核的超级计算机，也可以是互联的若干台计算机的集群。目常用的并行架构分为 SMP（多处理系统）、NUMA（非统一内存存储）、MPP（巨型并行处理）以及集群。

项目目的：

并行计算是实现高性能计算的主要技术手段，随着近几年云计算研究和大数据的崛起，越来越多的场景应用中需要并行化来提升性能。涉及并行计算的编程模型有 MPI、PVM、OpenMP、TBB 及 Cilk++等。其中许多都会调用系统多线程函数，需要用多种并行编程语言进行开发，利用这些并行技术可以充分发挥多核处理器的资源。在本项目中，我们首先将回顾并行计算在历史发展过程中出现的几种重要技术，并分析在当前比较常用的计算框架和技术，阐述它们在不同场景下的应用方向。并结合当前研究比较多的云计算和大数据，设计几个 benchmark 来比较分析这些技术在不同场景下的效率，探讨并行计算在未来的应用。

为了提高系统性能，我们将利用协同设计的原则，把不同场景应用的特点和处理器特性以及相应的并行技术相互配合。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，因此需要采用不同的架构进行处理。从处理器的特点来看，不同的处理器适合做不同的事情：如 X86 处理器为延迟优化，以减少指令的执行延迟为主要设计考量，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；而 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。

项目设计：

1）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，而各个应用也有不同的特点，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是我们要做的第一步。我们将比较常见的 SSE/AVX/NEON SIMD 指令集编程，以及用于多核编程的 OpenMP 标准；并且以稠密矩阵运算和图像处理为例，研究如何使用这些工具在不同场景下优化程序性能。

2）设计并行算法及任务分配

在设计并行算法中，需要充分考虑到当前应用场景的各种需求，选择合适并行架构和平台。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。面临比较复杂的应用场景，可以考虑异构平台上的并行：比如算法的一部分包含有许多逻辑处理，需要将这部分运算交给 X86 处理；而另一部分包含许多数据并行处理，又需要交给 GPU 处理。另外，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。这里提一下异构平台上进行任务分配的主要难点：即需要考虑多个不同的处理器之间的交互对性能的影响。例如在 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，这个消耗有的时候会成为算法性能的瓶颈。

3）任务映射

完成算法设计之后，如何将算法中的计算映射到硬件上，以便获得更好的性能，成为一个不得不考虑的问题。在不同的硬件中，这些考虑有相同点也有不同点。比如在集群中计算某个变量运算，在不同的节点之间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：

i. 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；
ii. 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；
iii. 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；

又或者在单个节点内，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。在多核心处理器上，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，这些设计都是需要大量智慧和经验的。

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，并分析性能瓶颈产生在什么地方，然后依据分析结果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架构下或平台上执行，此时可能需要更换平台或架构从头再来。