Supporting Multiple Accelerators in High-Level Programming Models

杜昆
2015310608
网络与信息系统安全

Contents

一、摘要

- **Publication: PMAM '15 Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores**
- **Conference: PPoPP (Principles and Practice of Parallel Programming)**
- **下载地址：http://dl.acm.org/citation.cfm?id=2712405**

一、摘要

- 计算加速器（Computational accelerators），如多核NVIDIA GPU、Intel Xeon Phi和FPGA，正变得越来越普及。
- 充分挖掘这些加速器的大规模并行计算能力，需要在程序的编程模型的设计和实现阶段都对并行计算做有针对性的设计。
- 作者设计了在高层就能够支持多加速器的编程模型，通过实现对OpenMP的扩展，支持从主机对多个加速器设备的计算管理，提高工作效率。

一、摘要

作者认为，文章的主要贡献包括：

1.在OpenMP中实现功能扩展，这些扩展包括：支持多个加速器数据加载、将多维数组以一定规则分布到多个加速器中、支持多个加速器间的数据共享、支持将循环迭代分布到多个加速器设备中。

2.实现了支持1中的各种需求的编译器和运行时，包括内存管理、不连续内存映射和多个加速器内存共享数据的打包/解包、CUDA内核代码生成、多个加速器协调管理。

3.实现了一个或多个用户线程同步/异步操作加速器接口，并对这些操作接口做性能分析。

二、问题的提出

- 异构计算架构能够将通用CPU和多种类型的计算加速器结合，如GPU、MIC和FPGA。但目前的编程模型都是面向主机的，在计算过程中以主机为中心。

- 高层编程模型如OpenACC和OpenMP都提供了并行计算接口，通过如pragma这样的变成接口实现。

二、问题的提出

- 这些模型有一个很大的缺点，就是默认情况下一次只能对一个加速器进行数据加载和计算。

- 如果要对多个加速器进行同时操作，如OpenMP 4.0，编程接口需要对每一个加速器手工设置数据环境，比如数据打包/解包、多加速器间的数据共享/传输/同步、数据内存边界。

三、问题的解决

1.设备抽象和虚拟拓扑

作者使用device关键字来声明多个设备，将多个设备抽象成二维数组，同时参考MPI和HPF对多个设备生成虚拟拓扑，如指定device(0:10)，表示选中设备编号从0到9的10个设备。

2.数据映射

通过map关键字实现向设备的数据映射，使用dist_data子句声明对数组每个维度的分配策略，如device(0:4) map(to:x[0:n]) dist_data(DUPLICATE)，其中DUPLICATE表示每个设备映射的内容相同，其它选项还包括BLOCK(n)（将连续数据分片映射到不同设备）、CYCLIC(n)（将数组的第i个元素块映射到第i个设备）

三、问题的解决

3.Halo区域和更新

Halo区域是多个设备间的数据共享区，仅用在BLOCK分布策略中使用。作者使用halo子句标识halo区域开始，同时使用halo_update语句来控制halo区域更新操作。如halo_update left(A[*][])，表示从左边设备中更新halo区域的一维部分。

4.循环迭代器平均分布

当在多个加速器中分配并行循环的数据时，理想的状态是将循环平均分布在每个加速器中。作者使用schedule语句表示循环如何在设备间分割，dist_iteration和dist_data语句来表示循环维度的分配。

四、系统实现

- 文中的实现针对NVIDIA GPU，生成CUDA代码，使用异构OpenMP编译器作为基础进行实现，主要分为两步实现：

（1）将使用本文提供的语句和指令写成的代码翻译成标准OpenMP 4.0代码，使用单设备编程模型；

（2）使用异构OpenMP编译器从标准单设备OpenMP代码生成C/CUDA代码，同时增加对多设备并行计算的支持。

四、系统实现

1.CUDA内核代码生成

CUDA内核使用循环更新（round-robin）算法分配程序中的循环迭代器，在CUDA内核中映射完整的数据内存，同时使用collapse子句将数据内存的一部分映射到单独的循环中。

四、系统实现

2.多设备管理

对于多设备管理有四种方案：

（1）通过同步操作使用单用户线程管理多个设备。这种方式简单方便，缺点是用户线程需要等待每个设备线程结束后才能进行下一个动作，限制了并行能力。

（2）通过多个用户线程或OpenMP线程管理多个设备，每个线程使用同步方式与设备通信。这种方式能够使设备具有良好的并行性，但用户线程可能会因为等待设备完成而阻塞，浪费计算时间。

四、系统实现

（3）使用单用户线程管理所有设备，但使用异步操作与设备交互。异步操作能够减少等待并使设备效能最大化，但是用户线程在某些时间点仍需要等待设备操作完成，因此这个线程在等待期间也无法做很多有用的操作；

（4）使用一个或多个后台线程管理一个、多个或所有设备。理论上这种方式不会产生用户线程的阻塞，但实际情况是用户线程往往需要等待设备操作完成才能继续进行程序操作，因此这种优势并不是十分明显。

文中作者使用了第二种设备管理方式。

四、系统实现

3.数组和循环迭代的分配

传统C和Fortran将多维数组分配在连续的内存空间，然而每个加速器往往只计算数组的一部分元素，因此运行时需要对数组元素划片（需要单独分配空间、计算边界等）和数据传输。

四、系统实现

4.Halo区域的处理

由于Halo区域存在于内存的不连续空间，将会在GPU内核的共享内存空间创建一个缓冲区，每个CUDA线程都能够向这个缓冲区中拷贝各自线程Halo区域的数据。如果硬件支持，Halo区域的数据交换可以通过异步CUDA操作完成。如果硬件不支持，运行时将在两个设备间完成完成数据交换。运行时会检查两个对象的硬件可达性，然后决定使用何种交换方式。

五、系统评估

1. 实验环境

一台实验主机包含两个4核Intel Xeon 5530 2.4GHz处理器（共8核）、24GB DDR3 ECC内存。通过使用超线程，每个核能够支持最多16个OpenMP线程。同时试验中使用4块NVIDIA Tesla C2050 Fermi GPU，每个GPU配有448个GPU核和3GB GDDR5内存。配套软件环境使用CUDA v5.5。

五、系统评估

2.评估内容

2.1 两种管理多个加速器的机制比较

（1）一个OpenMP线程使用异步操作管理所有GPU；
（2）多个OpenMP线程管理使用同步方式管理，每个线程对应一个GPU。

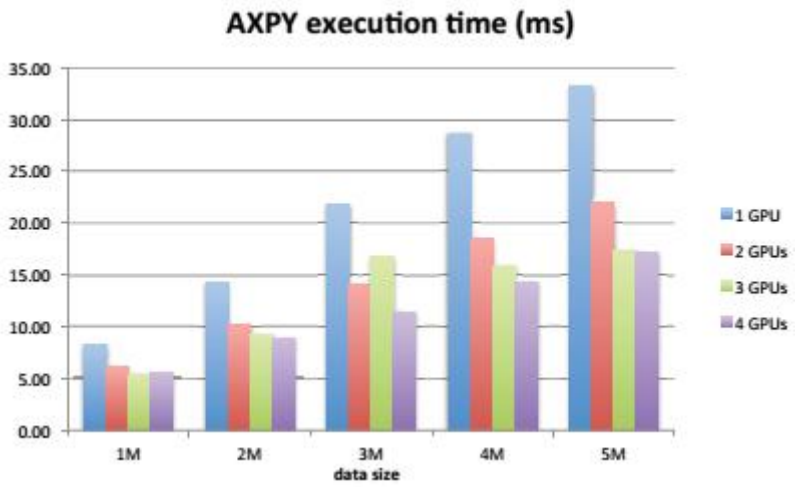
实验结果证明，随着GPU数量的增大，性能并没有线性增长。根本的原因在于硬件的限制，因为当有大量数据需要在内存移动时，共享的PCI数据总线将很快成为瓶颈；GPU数量的增加将导致串行内存操作增加，同时增大了竞争，导致性能下降。

五、系统评估

2.2 使用多GPU的性能

❖测试1. AXPY（向量乘法和加法 Y = a*X+Y）

这个计算不需要依赖其他数据，因此具有良好的并行性。



大量数据拷贝耗费巨大的计算时间，仅通过增加GPU数量并不能够有效改进计算效率。

五、系统评估

2.2 使用多GPU的性能

- 测试2. 矩阵乘法 $C_{i,j}=\sum_{k=0}^{n}A_{i,k}*B_{k,j}$ 的运算测试，算法复杂度是O(3)。

  这里可以采取三种策略：

（1）以行作为数据块分配A、C计算；

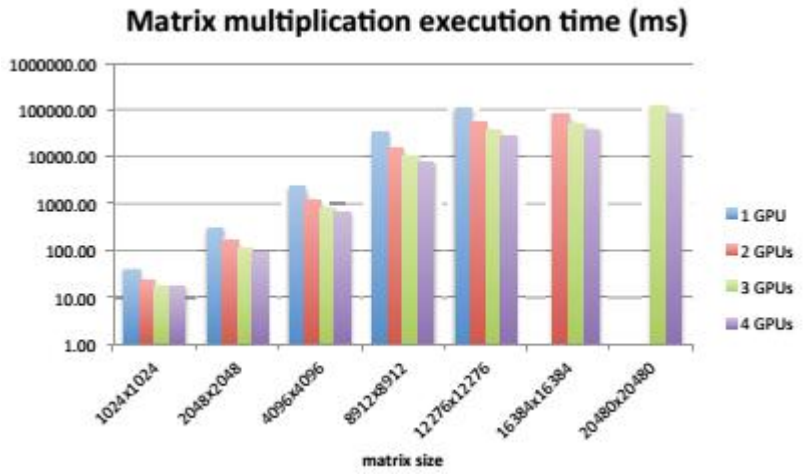
（2）以列为数据块分配A、C计算；

（3）同时以行和列为数据块分配C计算。

注意：后面两种计算方式的内存并不是连续分布的，因此需要数据打包/解包操作。

五、系统评估

2.2 使用多GPU的性能



- 说明：大数组没有在单GPU中测试
- 实验表明，由于硬件结构的限制，多个GPU的数据拷贝是串行的，因此增加GPU并不能带来效率的提高，通过增加硬件带来优化和线性增长效果并不理想。

五、系统评估

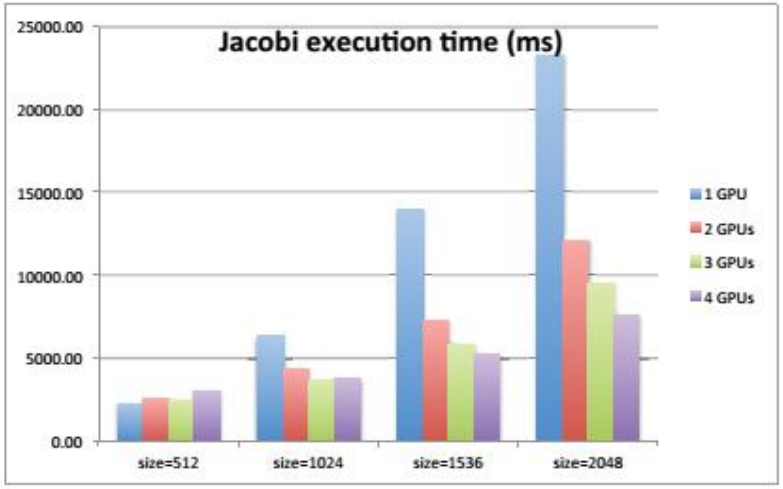
2.2 使用多GPU的性能

❖测试3. Jecobi算法

Jecobi算法用来计算2D视图更新，将所有元素映射成一个二维数组进行运算，每个元素计算的结果都依赖于四个临近方向元素的值。这个计算可以映射成Halo共享区的性能测试。作者采用以行作为数据块的分配策略在多个GPU中进行数据分配。

五、系统评估

2.2 使用多GPU的性能



❖这里有三个阶段的内核操作：

1.从内存向GPU的数据移动；

2.Halo区的内存互操作；

3.Jacobi算法内核计算。

试验中内核操作占全部时间比分别是24%、34%、42%。从这里可以看出硬件加速对性能的影响（Halo操作不占用CPU和内存资源）。

Thank You!