

基于 CPU/GPU 集群的编程的研究

刘钢锋

(西北工业大学 计算机学院, 陕西 西安 710072)

摘 要: 随着微处理器技术的发展, GPU/CPU 的混合计算已经成为是科学计算的主流趋势. 本文从编程的层面, 介绍了如何利用已有的并行编程语言来, 调度 GPU 的计算功能, 主要以 MPI(一种消息传递编程模型)与基于 GPU 的 CUDA(统一计算设备架构)编程模型相结合的方式进行 GPU 集群程序的测试, 并分析了 CPU/GPU 集群并行环境下的运行特点. 从分析的特点中总结出 GPU 集群较优策略, 从而为提高 CPU/GPU 并行程序性能提供科学依据.

关键词: GPU/CPU 的混合计算; 结合 MPI 与 CUDA

中图分类号: TP312

文献标识码: A

文章编号: 1000-7180(2013)02-0128-04

Research on Programming Based on CPU/GPU Heterogeneous Computing Cluster

LIU Gang-feng

(College of Computer, Northwestern Polytechnical University, Xi'an 710054, China)

Abstract: With the fast development in computer and microprocessor, Scientific Computing using CPU/GPU hybrid computing cluster has become a tendency. In this paper, from programming point of view, we propose the method of GPU scheduling to improve calculation efficiency. The main methods are through the combination of MPI (Message Passing Interface) and CUDA (Compute Unified Device Architecture) based on GPU to program. According to running condition of the parallel program, the characteristic of CPU/GPU hybrid computing cluster is analyzed. From the characteristic, the optimization strategy of parallel programs is found. So, the strategy will provide basis for improving the CPU/GPU parallel program.

Key words: CPU/GPU hybrid computing cluster; combination of MPI and CUDA

1 引言

随着微处理器技术的发展, 由主要用于图像处理的 GPU(Graphic Processing Unit)已经发展成为为用于科学计算的重要处理器, GPU 具有极高的吞吐量的, 适合进行计算密集度高、逻辑分支简单的大规模数据计算, 同时 GPU 具有功耗低, 性价比高的特点, 所以 GPU 在科学计算中的作用越来越重要. 为此同时, CPU 具有复杂的控制逻辑和大容量的缓存, 擅长复杂的逻辑运算和事务管理等功能. 为了使 CPU 和 GPU 的优势互补, CPU 与 GPU 混合的网络计算环境已经成为科学计算不可缺少的部分. 据

2011 年 11 月 TOP500^[1] 超级计算机的统计. 排名前五的超级计算机, 其中有 3 个超级计算机采用了 GPU. NVIDIA 发明了 CUDA^[2] (Compute Unified Device Architecture, 统一计算设备架构) 编程模型, 主要为了在单个主机上的应用程序中充分利用 CPU 和 GPU 各自的优点. MPI(Message Passing Interface)^[3] 是一种消息传递编程模型, 是这种编程模型的代表和实时上的标准, 是一个库, 是一种标准或者规范的代表. MPICH2^[4] 是对 MPI(包括 MPI-1 和 MPI-2) 高性能和普遍便携式的实现. 本文基于 MPICH2 对 MPI 实现的基础上, 通过信息传递方式, 把 CUDA 作用于 GPU 计算功能进行

调用.从实验数据中,对GPU/CPU的混合计算和CPU集群,单GPU,单CPU的性能进行了更加细致的分析,实践的基础上,对什么问题,使用什么的运算环境提供了最有价值的参考.

2 GPU/CPU 集群编程背景

2.1 MPICH2 的背景

MPI是对信息传输的一个库规范,是由用户,实现者和供应商共同建议的一种标准.主要用于大规模的并行计算机上,和工作集群上的高性能计算.他把每个任务看作一个进程,是适用于分布式存储结构的高性能计算机. MPI提供对C、C++和FORTRAN的绑定,而且提供可移植,标准化,运行性能,调用函数等功能.这些功能包括点到点的通信和集合通信操作. MPI提供了用于高性能分布式测试程序的编写方法和调试函数库.我们实验环境所使用的是MPICH2. MPICH2是对MPI(包括MPI-1和MPI-2)高性能和普遍便携式的实现. MPICH2提供了包括不同的计算和通信平台,包括商业集群(桌面系统,共享存储系统,多核结构),高速网络(万兆以太网, InfiniBand, Myrinet, Quadrics)和高性能终端计算系统(Blue Gene, Cray, SiCortex).通过兼容其它可扩充的实现的来扩充的模块框架,从而进行高端的科学研究.

在MPI的编程模式中,计算过程包括一个或者多个进程,这些进程通过调用库函数来实现数据的发送与接收.在大多数情况下, MPI实现中的进程是在初始化时创建的,每个处理器创建一个进程.但是这些进程可能执行不同的程序,因此MPI也涉及到多数据流程序与单程序多数据流相区别.

2.2 CUDA 编程背景

CUDA是一个并行计算平台,是由NVIDIA开发的并行编程模型,他可以利用图像处理单元的计算能力动态增加计算性能.当GPU在普遍使用以后,软件开发商,科学家,和研究者发现使用CUDA的GPU具有很强的计算前景.使用CUDA时,可以直接把C、C++和Fortran代码直接发送给GPU,而不需要有汇编语言的参与.在使用高级语言编程的时候, GPU的加速应用程序在CPU上的加载大部分是串行的,如果想优化,必须在GPU上的单个线程上的优化.这其实就叫做“GPU的计算”.

CUDA编程包括一个或者多个阶段,这些阶段要不工作在主机端(CPU)或者工作在设备端比如GPU,在主机端的代码有很少的或者没有数据的并

行性,在设备的代码中有很高的数据并行性.各个端程序的代码是程序是单一的,包括主机段和设备端代码. NVIDIA C的编译器(NVCC)分为两部分,主机端的代码是直接的ANCIC代码,主机的标准编译器可以直接编译主机端的代码,运行这些代码时和普通程序一样.设备端的代码是用扩充了叫做核(kernels)的标记作为数据并行的关键字和相关的结构体的ANCIC代码.设备段的代码,用NVCC编译,然后执行在GPU设备上.核函数产生了大量的线程来执行数据的并行行.典型的CUDA程序执行是在主机端启动,当“kernel”函数被激活时,那么“kernel”函数将移交给设备端(GPU)进行执行,这时将会产生大量的数据并行性来充分利用数据的并行性.

3 算法设计

在数学中,一个矩阵一般就是一个二维数组.一个 n 行 m 列的矩阵 A 乘以一个 m 行 p 列的矩阵 B ,得到的结果是一个 n 行 p 列的矩阵 C ,其中 C 的第 i 行第 j 列位置上的数等于 A 个矩阵第 i 行上的 m 个数与 B 矩阵第 j 列上的 m 个数对应相乘后所有 m 个乘积的和.

CUDA程序的设计:我们将会设计CUDA的内核函数,以及通过内核函数的调用来进行线程的组织.在CUDA中一个内核函数,将会被并行阶段的所有线程执行.也就是所有的线程都执行相同的代码,这就说明了CUDA编程时一个单程序多数据流的并行程序类型.

CUDA中对于内核函数的设计:

对于每一个块,迭代的把 A 的子矩阵和 B 矩阵相乘得到 C 的子矩阵.迭代的每一步如下:

(1) 同步地把 A 的子矩阵和 B 的子矩阵加载到共享存储器.

(2) 同步的让在一个块上的所有线程被加载

(3) 计算出子矩阵的乘积,然后和上次迭代的结果相加.

(4) 同步的让每一步都子矩阵乘积结束,开始下一轮的迭代.

MPI+CUDA的算法设计如下(假设集群有 P 个结点):

(1) 把 A 矩阵按行平均的分成 P 部分(对于进程 i 控制的 A 的行数是 $i * \lfloor n/p \rfloor$ 到 $(i+1) * \lfloor n/p \rfloor - 1$ 行的数据.也就是对于特定元素 j ,控制它的进程是 $\lfloor (p * (j+1) - 1) / n \rfloor$.所以划分数据时,主要

依据以上思路进行划分)^[5].

(2) 然后把收到的数据复制到 GPU 的设备存储器,调用 GPU 的内核函数,计算结束后把数据复制到各自到主机内存.

(3) 各结点计算完后将数据发送给主进程,将相应的数据汇总并存储响应数据.

4 实验结果

本实验借用了深圳超算中心的“星云”进行的测试,计算网络采用了单向 40Gbps QDR Infiniband 技术,核心存储采用了自主涉及的 Parastor 高速 IO 系统,并使用了 IntelXeon5650 + Fermi 架构的 NVIDIA TeslaC2050 通用处理器,运行在 Linux 操作系统之上.

从图 1 可以看出 GPU 对于矩阵乘法当矩阵小于 256×256 时数据 GPU 的开销明显大于 CPU 的开销,这是由于使用 GPU 计算时有额外开销(数据拷贝、GPU 的启动等)产生了大于 CPU 的开销,抵消了计算的高效性.当矩阵大于等于 512×512 时,由于计算占了较大的比例,使 CPU 的运行时间远远大于 GPU 的开销.经过测试当矩阵大于 1024×1024 时,运算效率甚至提高是原来的几十倍.

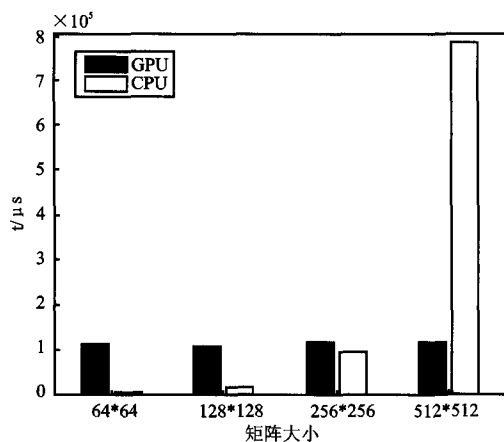


图 1 为单 CPU 和单 GPU 的计算对比图

从图 2 可以看出,当矩阵小于 512×512 时由于本身矩阵的计算量较小,用多个 CPU 运行没有明显优势,反而降低了性能.这主要由于基于 MPI 的并行程序,存在一定的通信开销导致的原因.当矩阵大于 2048×2048 时,CPU 集群的优势得到了进一步体现,这时相对于计算开销,通信开销就微乎其微.

图 3、图 4、图 5 是针对不同个数的 GPU 和 CPU 的运算时间对比.可以看出,矩阵乘法在 GPU 集群的情况下,矩阵小于 256×256 时,GPU 的集群

的运算性能明显高于 CPU 集群.由于当矩阵小于 256×256 时,由于 GPU 的额外开销(如内存拷贝,启动等原因)使得效率低于 CPU 集群.当矩阵大于 1024×1024 时,CPU 集群的对矩阵的花费时间要比 GPU 集群的花费不只是线性增长,这时在通信开销基本一致的情况下,GPU 集群的计算能力得到充分的体现.

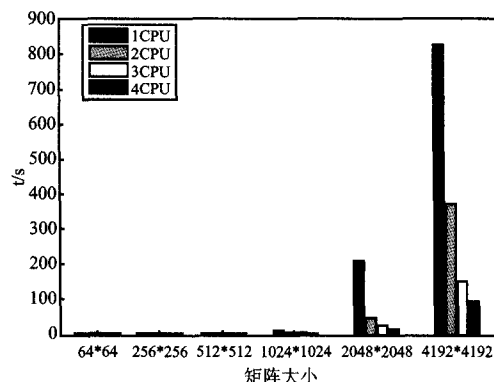


图 2 不同 CPU 的运算时间图

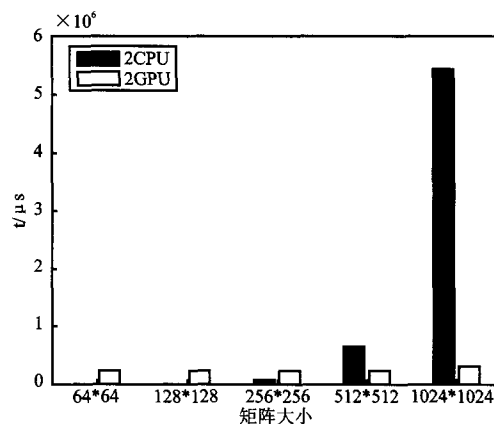


图 3 2 个处理器运算时间对比图

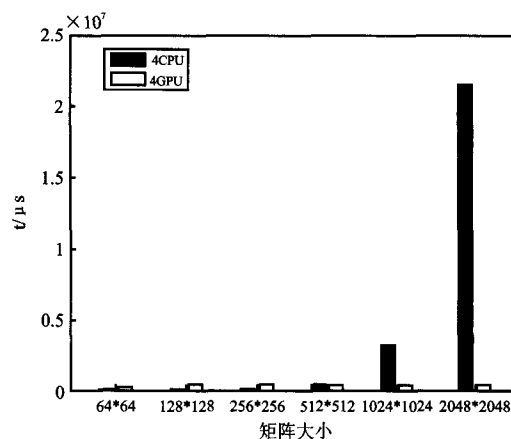


图 4 4 个处理器运算时间对比图

在单个 GPU 的与 2、4、8 的情况下测试结果如图 6.

从图 6 中可以看出当矩阵数目很小的情况下,

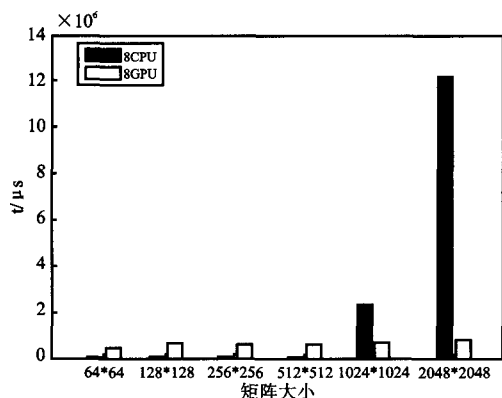


图5 8个处理器运算时间对比图

单个GPU的计算能力明显高于多个GPU的情况下,如在矩阵小于 2048×2048 的情况单个GPU的性能要高于多GPU的性能,而且随着GPU数目的增加性能出现下降的趋势.当矩阵大小大于 2048×2048 时随着GPU的数目增加效率有了明显提高,当矩阵为 8192×8192 时结点为4个GPU效率最高,可以说明GPU集群的调用过程中,不是GPU的数目越多那么效率就越高,而是依据问题的规模使用适当的GPU数目才能更好的计算效率.

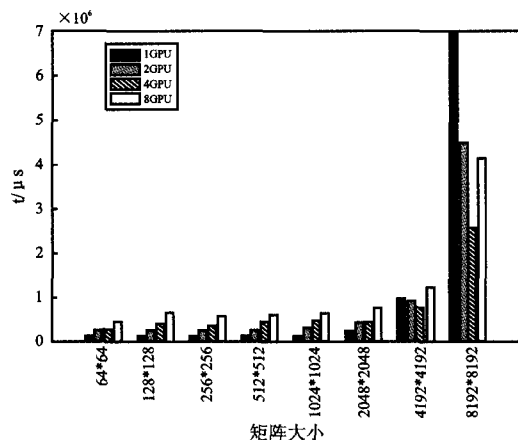


图6 不同数目GPU的运算对比图

5 结束语

本文是基于MPICH2对MPI实现的基础上,对CUDA在GPU程序的调用做了进一步的测试和研究,以及对GPU集群的性能进行了进一步的分析.从分析可以看出如何有效的根据问题规模进行合适结点的调用,是解决GPU集群效率的关键.我们将来的工作将是对GPU集群优化策略的进一步的分析,提出针对不同问题的优化策略.

参考文献:

- [1] 陈文光,武永卫, Michael. Quinn. MPI与OpenMP并行程序设计: C语言版[M]. 北京:清华大学出版社,2004.
- [2] NVIDIA CUDA Programming Guide[EB/OL]. [2012-05-18]. http://developer.nvidia.com/nvidia-gpu-computing-documentation/CUDA_C_Programming_Guide.pdf.
- [3] Top500 List-June 2011. [EB/OL]. [2012-05-18]. <http://www.top500.org/>.
- [4] NVIDIA CUDA Programming Guide. http://developer.nvidia.com/nvidia-gpu-computing-documentation/CUDA_C_Programming_Guide.pdf.
- [5] User's Guide[EB/OL]. [2012-05-18]. <http://www.mcs.anl.gov/research/projects/mpich2/documentation/index.php?s=docs/mpich2-1.4.1-userguide.pdf>.

作者简介:

刘钢锋 男,(1985-),硕士研究生,研究方向为高性能计算.