第58卷 第2期 天 文 学 报 Vol.58 No.2
2017年3月 ACTA ASTRONOMICA SINICA Mar., 2017

doi: 10.15940/j.cnki.0001-5245.2017.02.005

基于OpenCL的MUSER CLEAN算法研究与实现*

冯 勇[1] 陈 坤[1] 邓 辉[1†] 王 锋[1,2] 梅 盈[2] 卫守林[1,2]
戴 伟[1,2] 杨秋萍[1,2] 刘应波[1] 吴静平[3]

(1 昆明理工大学云南省计算机技术应用重点实验室 昆明 650504)
(2 中国科学院云南天文台 昆明 650216)
(3 云南省信息技术发展中心 昆明 650011)

摘要 天文软件开发中迫切需要在单机环境下进行高性能数据处理工作, 但由于机器配置不同, 采用传统的多线程、CUDA (Compute Unified Device Architecture)+GPU (Graphic Processing Unit)等方式都存在明显的局限, 不利于天文软件的快速移植和无缝运行. 对明安图频谱射电日像仪(MingantU SpEctral Radioheliograph, MUSER)数据处理系统开发中所采用的OpenCL (Open Computing Language)技术进行介绍, 并基于OpenCL实现Högbom CLEAN算法. 整体工作通过Python语言和PyOpenCL扩展包实现并行洁化处理. 实验结果表明: 基于OpenCL实现的CLEAN算法与基于CUDA实现的CLEAN算法具有大致相当的运行效率, 同时也可以无需修改代码直接实现纯CPU (Central Processing Unit)环境下的高性能数据处理, 解决了对CUDA+GPU环境依赖的问题, 在保证MUSER数据处理系统洁化过程性能的基础上, 提高了系统对硬件平台的适用性. 该工作验证了OpenCL在科学数据处理中的可用性, 可以预见: 由于OpenCL所具有的异构环境下高性能计算特性, OpenCL将是未来天文高性能软件开发的首选技术.

关键词 太阳: 射电辐射, 仪器: 干涉仪, 技术: 图像处理, 方法: CLEAN
中图分类号: P161; **文献标识码**: A

1 引言

新一代射电望远镜的观测对分辨率的要求日益提高. 由于单天线射电望远镜口径受限, 目前世界上最大的单天线望远镜FAST (Five-hundred-meter Aperture Spherical radio Telescope)或成为单天线望远镜的终结者. 同时, 随着世界上最大规模的天线阵列平方公里阵SKA (Square Kilometre Array)国际联合项目的提出, 射电天线阵的研究将成为未来研究的重点.

2016-08-15收到原稿, 2016-11-01收到修改稿
*中国科学院–国家自然科学基金委员会天文联合基金项目(U1231205、U1531132)资助
†dh@cnlab.net

中国新一代厘米分米波射电日像仪—明安图频谱射电日像仪(MingantU SpEctral Radioheliograph, MUSER)是利用综合孔径成像原理实现在频率范围0.4–15.0 GHz内对太阳进行高空间、高时间分辨率的多频段观测并成像的射电望远镜[1−3].

MUSER由天线(天线阵)、接收系统、数据处理系统3部分组成. 天线阵由低频阵(MUSER-I)和高频阵(MUSER-II)两部分组成, 其拓扑结构为螺旋阵. MUSER-I共40面天线, MUSER-II共60面天线[4]. MUSER-I中数据输出率为32 MB/s, 每日数据量约为1.2 TB[5].

面对天线阵产生的海量天文数据, 除了科学高效地进行实时任务调度以外, 如何从原始的观测数据高效、精确地获得观测图像是至关重要的工作. 以MUSER-I为例, 40面天线构成40×(40-1)/2架干涉仪, 即40×(40-1)/2条基线, 在UV平面采样得到780个数据点. MUSER-I根据频率不同, 成图大小有3种, 分别是256 pixels×256 pixels、512 pixels×512 pixels、1024 pixels×1024 pixels. 而780个采样数据点相对稀少, 天线阵对整个UV平面采样点覆盖不完全导致旁瓣干扰, 成像得到的图(称为脏图)包含大量虚假信息, 洁化用于对脏图进行处理, 以提高MUSER成像图像的质量, 使图像更加接近真实的太阳亮度分布. 此外, MUSER-I在高空间与高时间分辨率情况下, 每3 ms需要成16张图像. 显然, 这对MUSER成像在精确性与高效性方面提出很高要求.

在前期研究工作中, 研究人员已经利用分布式计算、并行计算中CUDA (Compute Unified Device Architecture)等方法开展了一系列高性能计算的研究工作, 取得了较好的效果[6−9]. 此外, MUSER数据处理系统已经成功部署到明安图观测站内的高性能计算环境中, 正逐步投入应用. 但在具体开发、部署和应用过程中, 这样的高性能计算模式仍存在一些不足. 具体表现为: 在开发过程中, 成图工作必须依赖GPU (Graphic Processing Unit)环境来完成; 在部署过程中, 硬件环境受限于NVIDIA的GPU, 选择过于单一, 不能适用于其他类型处理器; 在应用过程中, 科学家即使只处理一个小的数据也必须在这样的高性能平台上进行, 没有充分考虑实际需求, 也没有充分利用计算机资源.

是否能够实现只需一次编码完成, 就可以部署在不同平台、满足不同计算需求成为一个值得研究与探讨的问题. 本文正是基于这样的背景, 系统地研究了并行计算新标准OpenCL (Open Computing Language), 并采用OpenCL实现了面向MUSER成像中的CLEAN算法.

2 OpenCL

2.1 OpenCL简介

开放运算语言OpenCL[10]是一个面向异构系统的并行计算编程架构, 它用于在配备异构计算设备的系统中开发可移植的并行应用程序, 具有通用、开放、免费和跨平台的特点. 基于OpenCL编写的并行程序能在多核CPU (Central Processing Unit)、GPU、DSP (Digital Signal Processor)、FPGA (Field-Programmable Gate Array)以及其他处理器中运行[11]. OpenCL最早由Apple公司提出, 并由Khronos Group制定OpenCL规范[12], Khronos Group通过协调各个厂商, 从而使OpenCL具备跨平台性.

OpenCL定义了一个类C语言进行编程, 以及一系列的应用编程接口API (Application Program Interface). OpenCL异构系统包括计算设备(CPU、GPU等, 统称

为OpenCL设备)和与之相连的主机(CPU), 运行在OpenCL设备上的函数称为内核. OpenCL描述了平台模型、执行模型、内存模型以及编程模型[10−13]. 平台模型描述了平台内部单元之间的关系, 一台主机连接一个或多个OpenCL设备, 每个OpenCL设备包括多个计算单元(Compute Unit, CU), 每个计算单元包括多个处理单元(Process Element, PE), 由处理单元PE完成底层的计算过程. 执行模型描述了内核如何执行、内核与主机之间的交互以及内核之间的交互. 内存模型定义了设备上的4种不同内存空间, 分别是全局内存、局部内存、常量内存、私有内存. OpenCL提供了两种编程模型: 任务并行和数据并行.

2.2 OpenCL与CUDA比较

CUDA是一种由NVIDIA推出的并行计算架构[14]. 在高性能计算中采用CUDA进行并行计算已经相对成熟, 其利用GPU强大的并行计算能力, 使适合并行计算的程序执行效率大大提升. 但CUDA只针对单一的供应商NVIDIA, 不能适用于其他并行设备. 实际中, 并不是每台计算机或服务器上都配有NVIDIA的GPU, 由此限制了高性能计算中并行计算的发展. 而采用OpenCL充分利用了设备的并行特性的同时, 还为程序员提供了平台独立性. 毕竟绝大多数设备都配有CPU, 有些设备可能会配置其他厂商的GPU以及其他可进行并行计算的处理器, 在充分利用计算资源的同时也将极大扩展并行计算的应用领域. OpenCL与CUDA对比见表1.

表 1 OpenCL与CUDA比较
Table 1 Comparison between OpenCL and CUDA

Item	CUDA	OpenCL
Objective	Parallel computing	Parallel computing
Supporter	NVIDIA	Khronos group (Intel, AMD, NVIDIA, etc.)
Computation device	CUDA-enabled GPU	Multi-core CPUs, GPU, DSP, etc.
For developers	CUDA Development Library	OpenCL API
Programing language	C, C++, Java, Python, etc.	C, C++, Python, etc.
Interoperability	Not having access to OpenCL memory	Having access to CUDA memory

3 基于OpenCL的Högbom CLEAN算法实现

3.1 Högbom CLEAN算法

CLEAN算法[15]由Högbom于1974年提出, 用于综合孔径成像中消除由于UV平面采样点覆盖不完全对图像造成的影响, 以提高综合孔径成像图像的质量. 该算法使用已知的脏束来区分脏图上的真实结构(真实信息)和旁瓣干扰(虚假信息), 对点源有较好的处理效果, 在天文图像重建中被广泛应用. 洁化算法步骤如下:

(1)假设射电源可以用点源表示, 由天线阵拓扑结构得到采样函数, 对其进行傅里叶

逆变换得到脏束, 对可见度函数采样值进行傅里叶逆变换得到脏图;

(2)在脏图上找到最大强度(绝对值)的位置, 记录最大强度值及其位置;

(3)将脏束的中心移到这个最大强度的位置上, 并从脏图上减去乘上最大强度绝对值和循环增益因子γ后的脏束, 得到剩余图;

(4)在剩余图上重复步骤(2)和步骤(3), 直到剩余图中最大值小于指定阈值, 迭代终止, 得到一张包含被除去点的强度和位置的表, 此称为一个δ函数;

(5)用事先得到的洁束和这个δ函数作卷积, 并把剩余图加回去. 加剩余的图不是必须的, 不过这样可以保留原图的噪声水平和未找出的弱源信息.

3.2 洁化算法并行化

根据OpenCL规范, OpenCL程序主要由两部分组成: 一部分是在并行设备上运行的程序, 主要是利用OpenCL语言编写的内核函数; 另一部分是在主机上运行的程序, 主要作用是利用OpenCL的API管理在并行设备上运行的程序.

由于洁化算法中存在迭代过程, 每次迭代是对上一次迭代结果(剩余图)进行操作, 迭代过程不能并发执行, 不满足并行计算条件, 所以完全将洁化算法整个过程写成内核函数进行并行计算暂时是无法实现的. 但洁化算法中的某些操作或功能是满足并行计算条件的, 因此, 可以将这些操作或功能编写成内核函数在并行设备上进行并行计算, 由此来提高洁化算法中这些操作或功能的执行效率, 而其他无法进行并行计算的操作则在主机上维持串行执行, 相应地整个洁化算法的执行效率会被提高.

洁化算法步骤(2)的主要目的是从脏图中获得当前最大强度值及其位置. OpenCL提供得到矩阵中最大值的数学函数, 该函数是采用并行方式查找最大值, 但不返回其位置, 而矩阵中点的位置可由OpenCL提供的内置索引获得, 让脏图中所有点同时与这个最大强度值进行比较. 如果两个值相等, 说明这个值就是最大强度, 返回最大强度及其位置(执行OpenCL中的原子操作atomic_xchg()记录最大强度的位置), 初始化一个和脏图大小一样的矩阵model, 用来记录最大强度值及其位置. 查找脏图中点元素的最大值和最大值位置都是比较操作, 能同时进行, 即能通过并行计算提高查找的效率. 编写成内核函数find_max_kernel实现该功能. 该内核函数代码如下:

```
//Function to compute 1D array position
#define GRID(x,y,W) ((x)+((y)*W))
__kernel void find_max_kernel(__global float*dimg, __global int*maxid, float maxval, int W, int H, __global float*model){
    //Identify place on grid
    int idx=get_global_id(0);
    int idy=get_global_id(1);
    int id=GRID(idy,idx,H);
    //Ignore boundary pixels
    if(idx> -1 && idx<W && idy> -1 && idy<H){
        //Is this greater than the current max
        if(dimg[id]==maxval){
            int dummy=atomic_xchg(maxid,id);
```

```
        }
    }
    //Update the model
    barrier(CLK_LOCAL_MEM_FENCE);
    if(id==maxid[0]){
        model[id]+ =dimg[id];
    }
}
```

该代码中, __kernel是OpenCL内核函数的声明, get_global_id(0)和get_global_id(1)是OpenCL内置获取数组索引操作, atomic_xchg(maxid,id)是OpenCL原子交换操作, 将id值和maxid值交换, 返回最大强度位置, barrier(CLK_LOCAL_MEM_FENCE)是OpenCL中实现同步机制操作, 要求每个线程都执行完上述操作之后才能进行后续操作.

在洁化算法步骤(3)的处理过程中, 脏图和脏束上每个点是相互独立的, 可以对移位和减法操作进行并行处理, 从而提高步骤(3)的计算效率. 编写成内核函数sub_beam_kernel实现该功能.

洁化算法步骤(5)中参与计算的每个点的数据是相互独立的, 可以对其中的加法操作进行并行处理, 从而提高步骤(5)的计算效率. 编写成内核函数add_noise_kernel实现该功能.

根据上述分析, 给出洁化算法并行处理流程图(见图1).

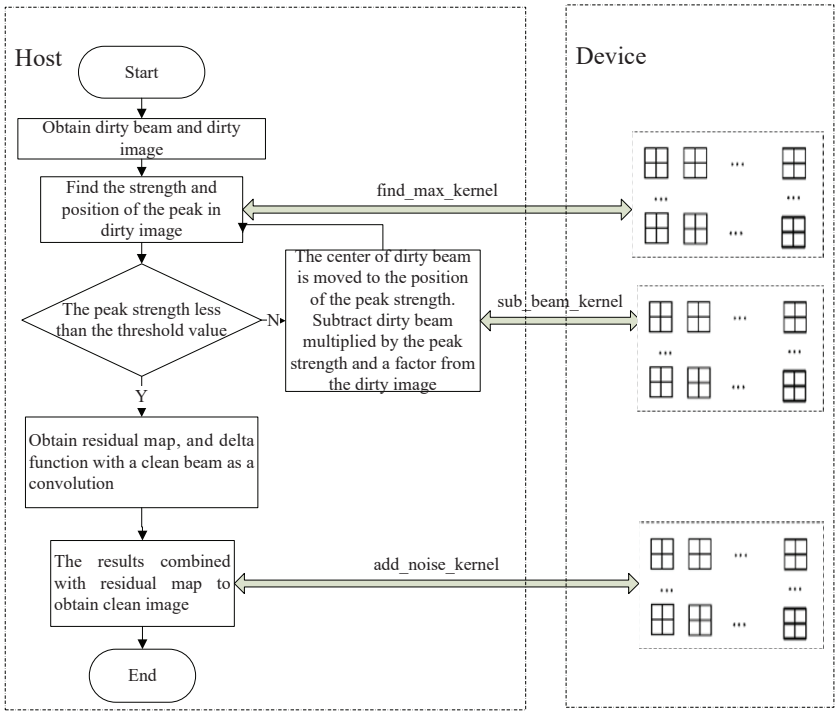

图 1　洁化算法并行处理流程图

Fig. 1　The flow diagram of CLEAN algorithm parallel processing

将编写好的内核函数以字符串方式放在kernel_source中, 导入Python中的OpenCL扩展包PyOpenCL[16], 遵循OpenCL编程规范, 对内核函数进行编译、调用等操作, 并给出基于OpenCL的并行洁化算法实现图(见图2).

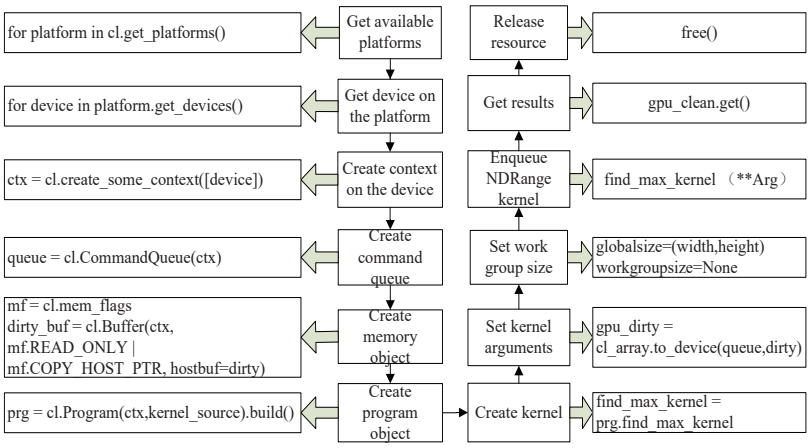

图 2 基于OpenCL的并行洁化算法实现图

Fig. 2 The implementation diagram of OpenCL-based CLEAN algorithm

图2给出了基于OpenCL的并行洁化算法的实现过程, 但由于洁化算法不能整体编写成内核函数, 所以主机部分程序中还包括洁化算法中必须串行执行的部分.

3.3 并行洁化算法主机部分

编写并行洁化算法在主机上运行的程序, 包括洁化算法中必须串行执行部分以及主机调用内核函数执行部分. 并行洁化算法在主机上运行程序伪代码如下:

Input gpu_dirty, gpu_pmodel, gpu_clean, gpu_dpsf, gpu_cpsf, thresh=0.2, gain=0.05, add_flag=1, add_back=1

Initialize

height, width = np.shape(gpu_dirty) //Set height and width as gpu_dirty's size

globalsize = (width,height) //Set globalsize as thread total

//Set gpu_max_id as the id of max value

gpu_max_id = cl_array.to_device(self.queue,np.zeros(1, dtype = 'int32'))

imax=self.gpu_getmax(gpu_dirty) //Set imax as max value

thresh_val=thresh //Set thresh_val as threshold value

i=0 //Set i as the number of iterations

Do While abs(imax) > thresh_val **AND** i < 200 **Then**

//Call the kernel function to find max value position

self.find_max_kernel(self.queue, globalsize, None, gpu_dirty.data, gpu_max_id.data, imax, np.int32(width), np.int32(height), gpu_pmodel.data)

//Call the kernel function to minus the dirty beam from dirty map

self.sub_beam_kernel(self.queue, globalsize, None, gpu_dirty.data, gpu_dpsf.data, gpu_max_id.data, gpu_clean.data, gpu_cpsf.data, np.float32(gain ∗ imax), np.int32(width),

np.int32(height), np.int32(add_flag))
    i += 1
    imax = self.gpu_getmax(gpu_dirty) //Obtain max from residual map
End While
If add_back == 1 **Then**
    //Call the kernel function to add back residual map
    self.add_noise_kernel(gpu_dirty, gpu_clean, np.float32(width + height))
End If
Output gpu_dirty, gpu_pmodel, gpu_clean

4 实验结果分析与讨论

4.1 实验环境

OpenCL定义不同的公司或厂商为不同的平台(Platform), 每个公司或厂商提供不同的OpenCL设备(Device). 本文选用的OpenCL设备有NVIDIA的GPU (Tesla k20m和Tesla k80)和Intel的CPU (Intel Xeon E5-2620 V2), 平台及设备参数见表2.

表 2 平台及设备参数
Table 2 The parameters of platform and device

Parameter	Intel CPU	Tesla k20m GPU	Tesla k80 GPU
Platform name	Intel(R) OpenCL	NVIDIA CUDA	NVIDIA CUDA
Platform vendor	Intel(R) Corporation	NVIDIA Corporation	NVIDIA Corporation
Platform version	OpenCL 1.2 LINUX	OpenCL 1.2 CUDA 8.0.44	OpenCL 1.2 CUDA 8.0.44
Device name	Intel(R) Xeon(R) CPU E5-2620 V2 @ 2.10 GHz	Tesla k20m	Tesla k80
Device max clock speed	2100 MHz	705 MHz	823 MHz
Device compute units	24	13	13
Device max work group size	8192	1024	1024

由于GPU核的数量较多, 所以GPU在并行计算中具有很大优势. 在OpenCL程序中, work group分配的大小会影响程序的执行效率, 本文采用默认值(None).

4.2 实验结果

实验数据来源于MUSER 2015年11月1日12时8分49秒354毫秒的观测数据, 原始数据经过一系列处理后保存为标准天文数据格式(FITS文件), 数据文件被命名为20151101-120849_354161240.uvfits. 通过MUSER数据处理系统执行相关操作, 分别生成脏图、脏束以及洁束, 再调用改进后的洁化算法对脏图进行洁化处理, 生成洁图. 脏图与洁图见图3.

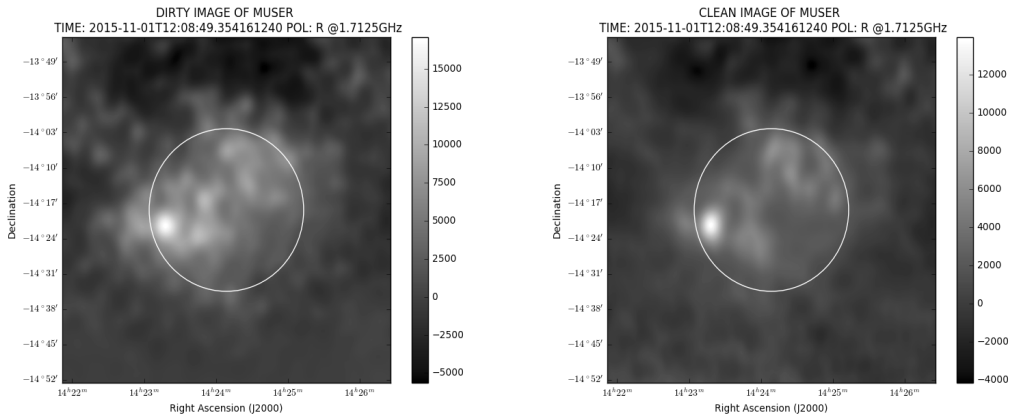

图 3 MUSER成像的脏图(左)与洁图(右). 观测时间是2015年11月1日12:08:49:354, 观测波频率是1.7125 GHz, 极化是右旋. 颜色棒表示未归一化的亮度数值.

Fig. 3 Dirty (left) and clean (right) images of MUSER. The figure shows dirty image of MUSER at 12:08:49:354 on November 11th, 2015 (Frequency: 1.7125 GHz, Polarization: right) and its corresponding clean image. The color bar shows the unnormalized numerical value of brightness.

脏图和洁图中的圆圈所围区域是根据亮度值及太阳半径计算出的近似太阳圆盘位置, 圆盘外围区域被认为是天空背景, 图中亮度较高的区域可能是太阳活动区域(更为精确的太阳圆盘与天空背景模型将在下一步工作中进一步研究). 对比脏图与洁图, 可以看出, 脏图中大量的虚假信息被消除, 洁图更加真实地反映出太阳亮度分布.

在同一台服务器上, 分别在CPU (Intel Xeon E5-2620 V2)、GPU (Tesla k20m和Tesla k80)上运行基于OpenCL实现的CLEAN算法和在GPU (Tesla k20m和Tesla k80)上运行基于CUDA实现的CLEAN算法, 生成大小为1024 pixels×1024 pixels的洁图. 每种情况下运行10次, 记录每次执行的时间, 实验结果见图4.

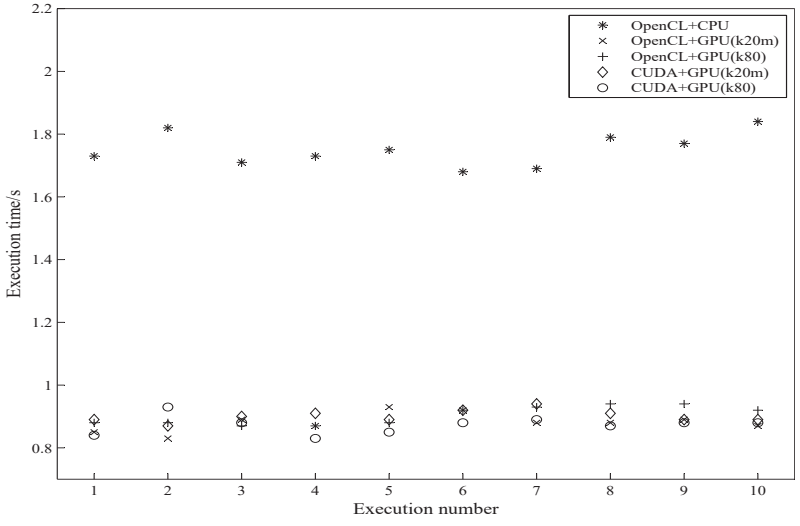

图 4 基于OpenCL和基于CUDA实现的CLEAN算法执行时间比较

Fig. 4 Comparison of time consume between OpenCL-based CLEAN and CUDA-based CLEAN

从图中可以看出, 基于OpenCL和基于CUDA实现的CLEAN算法在GPU环境下所消耗的时间基本相当, 同时, 基于OpenCL实现的CLEAN算法也能在CPU环境下运行, 并且取得较好的性能(平均每执行一次的时间约为1.7 s).

4.3 讨论

本文采用OpenCL实现了CLEAN算法的并行计算, 基于OpenCL实现的CLEAN算法具有以下优点:

(1)在GPU环境下, 基于OpenCL实现的CLEAN算法与基于CUDA实现的CLEAN算法执行时间大致相当, 保证了MUSER数据处理过程的整体效率;

(2)基于CUDA实现的CLEAN算法只能在NVIDIA的GPU上运行, 而基于OpenCL实现的CLEAN算法, 既可以在NVIDIA的GPU上运行, 也可以在其他厂商的GPU上运行, 这为MUSER数据处理系统提供了更多的硬件平台选择;

(3)由于OpenCL程序既可以运行于GPU环境, 也可以运行于CPU环境, 因此, 在实际工作中这提高了MUSER数据处理系统的容错能力, 即当没有GPU或者在GPU发生故障的情况下, 系统还可以在CPU环境下运行.

综上所述, 基于OpenCL实现的CLEAN算法在保证MUSER数据处理系统洁化过程性能的基础上, 增加了MUSER数据处理系统在硬件平台选择方面的灵活性. 同时, 本工作也验证了OpenCL在科学数据处理中的可用性, 可以预见, 由于OpenCL所具有的异构环境下高性能计算特性, OpenCL将是未来异构环境下天文高性能软件开发的首选技术.

参 考 文 献

[1] Yan Y, Zhang J, Wang W, et al. EM&P, 2009, 104: 97
[2] 颜毅华, 张坚, 陈志军, 等. 天文研究与技术, 2006, 3: 91
[3] 姬国枢, 王威, 何俊波. 天文研究与技术, 2010, 7: 95
[4] 耿立红, 颜毅华, 宋庆辉, 等. 天文研究与技术, 2016, 13: 160
[5] 陈泰燃, 王威, 王锋, 等. 天文研究与技术, 2016, 13: 184
[6] Wang F, Mei Y, Deng H, et al. PASP, 2015, 127: 383
[7] 代慧梅, 梅盈, 王威, 等. 天文学报, 2016, 57: 19
[8] 周鑫磊, 王威, 王锋, 等. 天文研究与技术, 2015, 12: 503
[9] 卫守林, 石聪明, 高姣姣, 等. 天文研究与技术, 2016, 13: 117
[10] Munshi A, Gaster B, Mattson T G, et al. OpenCL Programming Guide. 北京: 科学出版社, 2012: 7-22
[11] Stone J E, Gohara D, Shi G. CSE, 2010, 12: 66
[12] Khronos OpenCL Working Group. The OpenCL Specification. 2015: 25-90
[13] Du P, Weber R, Luszczek P, et al. ParC, 2012, 38: 391
[14] Nicholas W. CUDA专家手册—GPU编程权威指南. 北京: 机械工业出版社, 2014: 1-20
[15] Högbom J A. A&AS, 1974, 15: 417
[16] 刘颖, 吕方, 王蕾, 等. 软件学报, 2014, 7: 1459

The Research and Implementation of MUSER CLEAN Algorithm Based on OpenCL

FENG Yong[1] CHEN Kun[1] DENG Hui[1] WANG Feng[1,2] MEI Ying[2]
WEI Shou-lin[1,2] DAI Wei[1,2] YANG Qiu-ping[1,2] LIU Ying-bo[1]
WU Jing-ping[3]

(1 Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650504)
(2 Yunnan Astronomical Observatories, Chinese Academy of Sciences, Kunming 650216)
(3 Yunnan Information Technology Development Center, Kunming 650011)

Abstract It's urgent to carry out high-performance data processing with a single machine in the development of astronomical software. However, due to the different configuration of the machine, traditional programming techniques such as multi-threading, and CUDA (Compute Unified Device Architecture)+GPU (Graphic Processing Unit) have obvious limitations in portability and seamlessness between different operation systems. The OpenCL (Open Computing Language) used in the development of MUSER (MingantU SpEctral Radioheliograph) data processing system is introduced. And the Högbom CLEAN algorithm is re-implemented into parallel CLEAN algorithm by the Python language and PyOpenCL extended package. The experimental results show that the CLEAN algorithm based on OpenCL has approximately equally operating efficiency compared with the former CLEAN algorithm based on CUDA. More important, the data processing in merely CPU (Central Processing Unit) environment of this system can also achieve high performance, which has solved the problem of environmental dependence of CUDA+GPU. Overall, the research improves the adaptability of the system with emphasis on performance of MUSER image clean computing. In the meanwhile, the realization of OpenCL in MUSER proves its availability in scientific data processing. In view of the high-performance computing features of OpenCL in heterogeneous environment, it will probably become the preferred technology in the future high-performance astronomical software development.

Key words sun: radio radiation, instrumentation: interferometers, techniques: image processing, methods: CLEAN