轻工学报® 2019 年 3 月 第 34 卷 第 2 期
JOURNAL OF LIGHT INDUSTRY Vol. 34 No. 2 Mar. 2019

引用格式:吴怀广,刘琳琳,石永生,等. ARL 中 Gridding 算法的并行化实现[J]. 轻工学报,2019,34(2):82 – 87.
中图分类号:F222.3 **文献标识码**:A
DOI:10.3969/j.issn.2096 – 1553.2019.02.011
文章编号:2096 – 1553(2019)02 – 0082 – 06

ARL 中 Gridding 算法的并行化实现

Research on parallelization of Gridding algorithm in ARL

吴怀广,刘琳琳,石永生,李代祎,谢鹏杰
WU Huaiguang, LIU Linlin, SHI Yongsheng, LI Daiyi, XIE Pengjie

郑州轻工业大学 计算机与通信工程学院,河南 郑州 450001
College of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou 450001, *China*

关键词:
ARL;并行化算法;Gridding 算法;CUDA

Key words:
ARL; parallelization algorithm; Gridding algorithm; CUDA

摘要:针对海量天文数据实时性处理效率低的问题,通过对 SKA 图像采集及成像 ARL 算法库中耗时较长的 Gridding 算法进行耗时分析,找出了该算法中调用频率高且运行时间长的两个函数 convolutional-grid 和 convolutional-degrid,利用 GPU 的多线程并行化处理降低两个函数的循环迭代,实现了 Gridding 算法在 GPU 和 CPU 上的协同运行. 验证实验结果表明,在相同的数据量下,改进后的 Gridding 算法运行时间大大缩短,特别是在处理海量数据时,有效提高了 ARL 的整体运行效率.

收稿日期:2018 – 12 – 13
基金项目:国家重点研发计划政府间科技合作项目(2016YFE0100600;2016YFE0100300)
作者简介:吴怀广(1976—),男,山东省聊城市人,郑州轻工业大学副教授,博士,主要研究方向为形式化方法、数据质量、大数据分析技术.

Abstract: Aiming at the low real-time processing efficiency of massive astronomical data, through time-consuming analysis of gridding algorithm in SKA image acquisition and imaging ARL library, two functions of convolutional-grid and convolutional-degrid with high frequency and long running time were found out in this algorithm. Then, two functions were parallelized on GPU by multi-threading to realize the cooperative operation of gridding algorithm on GPU and CPU. The experimental results showed that under the same amount of data, the running time of the improved gridding algorithm was greatly shortened, especially when dealing with massive data, the overall running efficiency of ARL was effectively improved.

0 引言

在过去的几十年里,射电望远镜的灵敏度和图像分辨率均有很大的提升. SKA[1-2] 作为世界上最大综合孔径的射电望远镜,采集数据的速率非常快,数据采集量非常大,其设计目标是要大于 12 TB/s. 海量数据的产生和天文成像本身对实时处理的严格要求,给计算机的计算能力带来了巨大的挑战. 海量数据的科学处理一般需要百亿亿次量级的超级计算机来完成. 目前,通过提高主频来提高 CPU 处理能力的传统方式已受到集成电路集成度的制约,因此利用多核 CPU 和多核加速器(如 GPU, Cell/BE 等)处理大量时效性数据成为发展趋势. 在天文成像过程中,网格化和去网格化是最耗时的两个操作. 如果处理的可见度数据是 EB 量级或以上,则消耗的时间不能通过调整计算机性能来缓和. 传统的网格化一般都是在 CPU 上运行,加速器在每一个浮点上的带宽很小,应用的时效性也相应较低. 所以,并行化算法作为提高计算速度的一个重要途径,在射电天文学数据处理[3]方面越来越受到关注.

网格化(Gridding)算法的作用是将落在笛卡尔坐标外的数据点插值到网格上. W. N. Brouw[4] 最早提出了 Gridding 方法,用于实现极坐标网格采样的离散傅里叶变换. 针对极坐标网格采样,J. D. O'Sullivan[5] 提出了一种快速的 sinc 函数网格算法,并将其应用在 CT 图像重建上,该算法利用有限范围的卷积函数得到 sinc 函数插值以减小计算时间,但是计算量太大. C. H. Meyer 等[6] 将 Gridding 算法应用于磁共振成像(MRI)中的螺旋采样,采用的卷积核是 Kaiser-Bessel 窗口函数,但是 Kaiser-Bessel 窗口函数不是最优的卷积函数. 劳保强等[7] 使用卷积函数网格化不规则的微波全息数据,分析了不同卷积函数的抗混叠性能,验证了球体函数是最具抗混淆性能的卷积函数. A. L. Varbanescu 等[8] 以增加附加计算为代价,通过对数据的排序和搜索来改善空间局部性,进而改善内存的性能,但其结果尚未达到 GPU 运算峰值的 14%.

以上研究主要是在 CPU 环境下对 Gridding 算法的改进,鲜见在 ARL(SKA 的算法参考库,用以实现整体天文数据成像流程)中针对 Gridding 算法在 GPU 实现时的运行效率的研究. 鉴于此,本文拟分析 CPU 环境下 ARL 中耗时最长的 Gridding 算法,利用性能分析工具找出 Gridding 算法中耗时最长的函数模块,然后对函数模块使用 CUDA 进行 GPU 并行执行,以期利用 GPU 在并行运算方面的优势,实现 Gridding 算法的 CPU 和 GPU 协同处理,从而提高 ARL 的整体运行效率.

1 Gridding 算法在 ARL 网格化中的应用

在射电天文学中,所采集到的信号通常受到各类因素影响,数据呈现出不规律性. 在利用射电望远镜采集到的数据重建天空图像时,需要将不规则采样的数据映射到标准的二维网格中,这称为网格化. 只有经历这一步,网格才可

以通过快速傅里叶变换来重建天空图像. 在射电望远镜数据处理中,网格化是计算量最大、耗费时间最长的步骤.

在天文成像中,对于天空亮度 I,初始光束模式 A 和能见度值 V,存在以下傅里叶变换关系:

$$A(l,m)I(l,m) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} V(u,v)e^{2\pi i(ul+vm)}\mathrm{d}u\mathrm{d}v \qquad ①$$

用 $I^D(l,m)$ 表示修正天空亮度 $A(l,m)I(l,m)$,利用线性操作对式 ① 的右边作估计可得

$$I^D(l,m) \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S(u,v)V'(u,v)e^{2\pi i(ul+vm)}\mathrm{d}u\mathrm{d}v \qquad ②$$

其中,S 表示 (u,v) 采样函数,V' 表示观测值.

对式 ② 作数值估计的方法有直接傅里叶变换 DFT(direct Fourier transform) 和快速傅里叶变换 FFT(fast Fourier transform) 两种方法. 其中,DFT 对 $N \times N$ 网格的每个点都做计算,通过蛮力计算求和来估计 $I^D(l,m)$,其表达式为

$$I^D(l,m) = \frac{1}{M}\sum_{k=1}^{M} V'(u_k,v_k)e^{2\pi i(u_k l+v_k m)}$$

FFT 则通过利用 DFT 运算中的对称性和周期性,减少 DFT 的运算量,从而更快地估计 $I^D(l,m)$. 但是利用 FFT 需要先将数据插值到矩形网格点上,然后才可以应用 FFT. 插值的过程就称为网格化. ARL 中 Gridding 算法的作用是将落在笛卡尔坐标外的数据点插值到网格上,以便对网格化的数据进行 FFT 计算.

天文学中的 Gridding 算法步骤如下.

1) 密度补偿函数与采样数据相乘. Gridding 算法首先要用密度补偿函数与采样数据相乘来弥补采样数据的不均匀. 网格算法中的采样密度补偿函数和插值函数(卷积函数)的选择对图像重建呈现的最后效果影响较大,也是网格化算法研究的核心问题.

采样函数可表示为

$$S(u,v) = \sum_{k=1}^{M}\delta(u-u_k,v-v_k)$$

其中,S 是一个广义函数,或者称之为广义分布,可以表示为二维狄拉克 δ 函数或 δ 分布.

采样补偿函数,又称为采样补偿分布,可表示为

$$W(u,v) = \sum_{k=1}^{M} R_k T_k D_k \delta(u-u_k,v-v_k)$$

其中,R_k,T_k,D_k 均为数据点 (u_k,v_k) 的对应补偿系数.

2) 在网格点 (u_c,v_c) 处计算出卷积,即

$$\sum_{k=1}^{M} C(u_c-u_k,v_c-v_k)V^W(u_k,v_k)$$

其中,$C(u_c-u_k,v_c-v_k)$ 表示卷积函数,(u_c,v_c) 表示网格点,$V^W(u_k,v_k)$ 是经过采样和处理后的数据.

3) 对数据进行重采样,使数据落到网格点上. 在所有网格点上对 $C \times V^W$ 作采样处理,即

$$V^R = R(C \times V^W) = R(C \times (WV'))$$

其中,R 是重采样函数,可表示为

$$R(u,v) = III(u/\Delta u,v/\Delta v) = \sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty}\delta(j-u/\Delta u,k-v/\Delta v)$$

此处,Δu 和 Δv 定义了网格点间距的大小,III 为 Shah 函数.

4) 应用 FFT 对 V^R 作傅里叶变换,得出经过简单修正的初步图像. V^R 是规则间隔的 δ 函数的线性组合,可以由离散傅里叶变换计算出其傅里叶变换 $\mathcal{F}V^R$ 的采样矩阵,其中 $\mathcal{F}$ 表示傅里叶变换. $\mathcal{F}V^R$ 是经过正规化和一个简单修正的初步图像,用 $\tilde{I}^D$ 来表示.

$$\tilde{I}^D = \mathcal{F}R \times [(\mathcal{F}C)(\mathcal{F}V^W)] = \mathcal{F}R \times [(\mathcal{F}C)(\mathcal{F}W\,\mathcal{F}V')]$$

至此,图像 $\tilde{I}^D$ 得到初步重建.

5) 为了消除卷积函数的影响,需要对重建后的图像作进一步的修正,即除以卷积函数的傅里叶变换.

2 Gridding 算法的耗时分析

Gridding 中一共含有 14 个函数,对这些函数进行分析,选出调用频率较高、运行时间最长的 4 个函数:1)grdsf 函数,其功能是获取网格函数及对网格进行修正;2)anti_aliasing_calculate 函数,其功能是计算 prolate 球体反混叠函数;3)convolutional_degrid 函数,其功能是利用被采样的 gcf 部分 *uv* 坐标值,进行卷积解网格;4)convolutional_grid 函数,其功能是由频率和偏振独立的 gcf 进行卷积网格. Gridding 算法中其他函数因其调用次数和单个函数耗时都很小,在整个过程中可以忽略不计.

选出的 4 个函数被调用次数如图 1 所示,执行时间如图 2 所示. 分析中可见性数据量为 10^4 条.

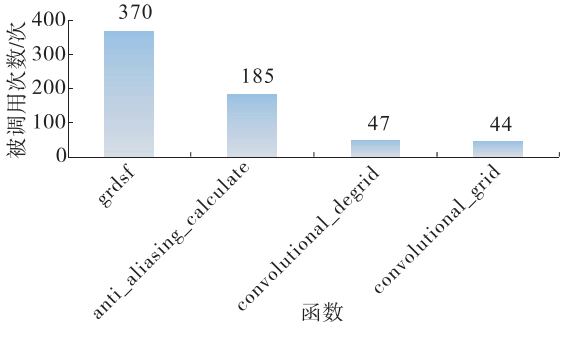

图1 Gridding 算法中 4 种主要函数被调用次数

Fig. 1 The call number of four main functions in Griddina algorithm

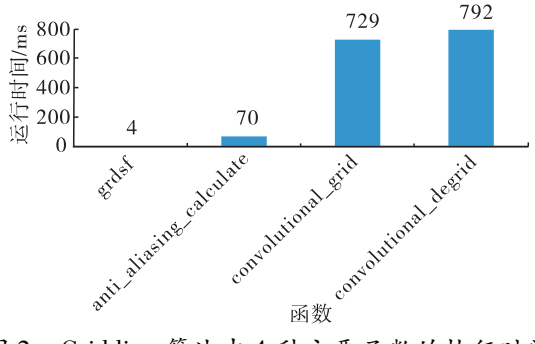

图 2 Gridding 算法中 4 种主要函数的执行时间

Fig. 2 The execution time of four main functions in the Gridding algorithm

由图 1 可以看出,grdsf 函数的被调用次数最高,达到 370 次,anti_aliasing_calculate 函数被调用 185 次,convolutional_degrid 函数被调用 47 次,convolutional_grid 函数被调用 44 次.

由图 2 可以看出,convolutional_grid 和 convolutional_degrid 两个函数运行时间最长,而 grdsf 和 anti_aliasing_calculate 两个函数运行时间较少.

综合来看,需要对 convolutional_grid 和 convolutional_degrid 两个函数进行并行化处理.

3 Gridding 算法的并行化实现

在整个 SKA 图像采集和最终成像的过程中,Gridding 算法是计算量大、耗时长的算法之一. ARL 算法库是基于 CPU 运行的,利用 PyCUDA 进行并行化加速可以使整个 ARL 的运行时间得到有效缩短.

通常,GPU 并行加速方式两种:一种是在串行代码的基础上通过降低循环进行加速;另一种是重新设计算法,将总模块划分为多个独立的子模块,通过分配线程来并行处理子任务. 本文采用第一种优化措施. 这是因为 Gridding 算法在处理数据的时候迭代次数很多,通过对 Gridding 算法的调用次数和耗时的分析可以看出,计算量最多的是 convolutional_grid 和 convolutional_degrid 函数,而这两个函数运算时 for 循环较多,且数据之间不存在依赖关系,所以可以直接利用多线程实现其在 GPU 下的并行计算,从而达到降低 for 循环次数,加快运算速度的目的.

由于 GPU 与 CPU 之间不能直接通信,所以进行并行化处理之前需将数据从主机端(CPU)拷贝到设备端(GPU),然后才能执行相关的核函数. 同样的,对于 GPU 并行化处理后的结果也需要从 GPU 拷贝到 CPU.

本文利用全局存储器进行并行化处理. 实

验所用显卡的静态存储器的大小是11 440 MB，因此所有的线程模块中的线程不会产生访问冲突. 利用 CUDA[9] 中的 cudaMemcpy( )将数据载入到 GPU 的静态存储器中，以便所有的线程共享. 根据线程索引获取可见性数据进行求和. 等到线程都完成计算，最后将结果载入到内存. 进行 GPU 并行处理的流程图如图 3 所示.

4　验证实验结果与分析

实验环境：CPU 为 Intel Xeon E5 - 2620 V3，内存为 816GB DDR4，内存最大支持为 768 GB，操作系统是 CentOS 7.0，python 版本为 3.5；GPU 采用 Tesla K80，为双核 GK210 架构，主板型号为 MG50 - G20.

实验中选用的 GPU 一共含有 4992 个 CUDA Cores，每个块最多分配 1024 个线程，对于不同的数据量可以选择不同的线程和线程块. 本实验使用的数据量分别为 10^3 条，10^4 条，10^5 条，10^6 条，10^7 条，10^8 条，针对不同的数据量，线程块统一采用(32,32,1)的形式，采用的网格块的分配分别是(1,1,1)，(10,1,1)，(10,10,1)，(100,10,1)，(100,100,1)，(1000,100,1).

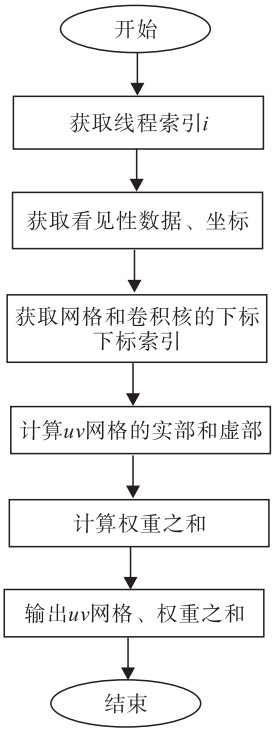

图 3　Gridding 算法并行流程

Fig. 3　The parallel flow of Gridding algorithm

ARL 中 convolutional_grid 函数和 convolutional_degrid 函数在 CPU 上运行的时间与在 GPU 加速后的运行时间见表 1，其中加速比指在 CPU 上运行时间和 GPU 上运行时间的比值.

从表 1 可以看出，随着数据量的增加，两个函数的加速比不断增加，但是加速比增加的速度逐渐减小. 为了更直观地表示加速比和数据量之间的关系，采用直方图进行表示，结果如图 4 和图 5 所示.

由图 4 和图 5 可以看出，当数据量较少(10^3 条)时，两个函数的加速比分别是 0.22 和 0.38，利用 GPU 并行处理的运行时间反而增加，这是因为数据量相对较少时，GPU 器件的启动时间、内存和显存之间数据交互等都需要时间. 因此在数据量比较小时，GPU 因其内部调度相对来说耗时较长，其并行加速并不占优势. 随着数据量的增加，GPU 加速的优势逐渐

表 1　两个函数在 CPU 和 GPU 上的运行时间

Table 1　Running time of two functions on CPU and GPU

函数	数据量/条	CPU 下运行时间/s	GPU 下运行时间/s	加速比
convolutional_grid	10^3	0.040	0.180	0.22
	10^4	0.290	0.179	1.62
	10^5	3.110	0.396	7.85
	10^6	27.714	2.197	12.61
	10^7	277.703	19.539	14.21
	10^8	2 886.311	182.794	15.79
convolutional_degrid	10^3	0.058	0.152	0.38
	10^4	0.462	0.181	2.55
	10^5	3.292	0.351	9.38
	10^6	30.525	2.134	14.30
	10^7	272.345	17.935	15.19
	10^8	3 057.845	179.233	17.06

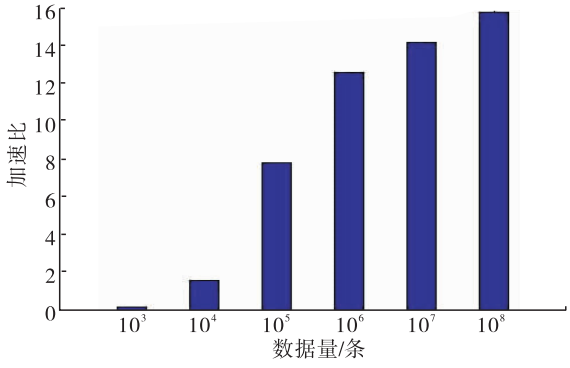

图 4 convolutional_grid 函数在不同数据量下的加速比

Fig. 4 Acceleration ratio of convolutional_grid function under different data volumes

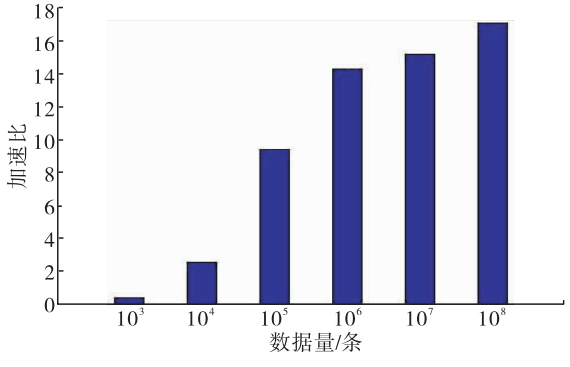

图 5 convolutional_degrid 函数在不同数据量下的加速比

Fig. 5 The acceleration ratio of the convolutional_degrid function under different data volumes

显现. 当数据量大于 10^4 条时,加速比不断增加. 因此,GPU 并行加速适用于大量数据的运算情况,可在一定程度上提高 ARL 的整体运行效率.

4 结语

本文针对 SKA 海量天文数据实时性处理效率较低的问题,对 ARL 算法库中 Gridding 算法进行了耗时分析与优化:通过分析 Gridding 算法中每个函数的调用次数和运行时间,找出了需要优化的两个函数,即 convolutional_grid 和 convolutional_degrid 函数;然后利用 GPU 的多线程并行化处理降低两个函数的循环迭代,使得 ARL 中 Gridding 算法在 GPU 和 CPU 上实现了协同运行. 验证实验结果显示,在数据量不影响输入输出数据格式和大小的情况下,对 Gridding 算法的并行化处理缩短了 ARL 的整体运算时间,提高了运行的效率,且数据量越大,加速优势越明显.

本文所做的并行化研究对于天文图像的成像过程具有一定的参考意义,对 Gridding 算法在多 GPU 下实现算法加速将是下一步的研究方向.

参考文献:

[1] RAZAVI-GHODS N, ACEDO E D L, EL-MAKADEMA A, et al. Analysis of sky contributions to system temperature for low frequency SKA aperture array geometries[J]. Experimental Astronomy, 2012, 33(1): 141.

[2] ZHANG Y, BROWN A K. Bunny ear combline antennas for compact wide-band dual-polarized aperture array [J]. IEEE Transactions on Antennas and Propagation, 2011, 59(8): 3071.

[3] 彭晓明,郭浩然,庞建民. 多核处理器——技术、趋势和挑战[J]. 计算机科学, 2012, 39(S3): 320.

[4] BROUW W N. Aperture synthesis[J]. Methods in Computational Physics, 1975, 14: 131.

[5] O'SULLIVAN J D. A fast sinc function gridding algorithm for fourier inversion in computer tomography[J]. IEEE Transactions on Medical Imaging, 1985, 4(4): 200.

[6] MEYER C H, HU B S, NISHIMURA D G, et al. Fast spiral coronary artery imaging[J]. Magnetic Resonance in Medicine, 1992, 28(2): 202.

[7] 劳保强,王俊义,王锦清,等. 基于卷积核网格化二维近程微波全息[J]. 微波学报, 2014, 30(5): 82.

[8] VARBANESCU A L. On the effective parallel programming of multi-core processors [D]. Romania: Universitatea Politehnica Bucuresti, 2010.

[9] CHENG J. CUDA by example: an introduction to general-purpose GPU programming [M]. Boston: Addison-Wesley Professional, 2010.