软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2011,22(Suppl.(2)):172−181

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# 基于 GPU 应用于大规模星模拟器的灰度弥散模型*

李 超[1,3+] 张云泉[2], 郑昌文[1], 胡晓慧[1]

[1](中国科学院 软件研究所 综合信息系统技术国家级重点实验室,北京 100190)

[2](中国科学院 软件研究所 并行软件和计算科学实验室,北京 100190)

[3](中国科学院 研究生院,北京 100049)

## Intensity Model with Blur Effect on GPUs Applied to Large-Scale Star Simulators

LI Chao[1,3+], ZHANG Yun-Quan[2], ZHENG Chang-Wen[1], HU Xiao-Hui[1]

[1](National Key Laboratory of Integrated Information System Technology, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Laboratoy of Parallel Software and Computational Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190 China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: happysuperlee@163.com

**Li C, Zhang YQ, Zheng CW, Hu XH. Intensity model with blur effect on GPUs applied to large-scale star simulators. *Journal of Software*, 2011,22(Suppl.(2)):172−181.** http://www.jos.org.cn/1000-9825/11038.htm

**Abstract**: Intensity model with blur effect is widely employed to accurately simulate the imaging process of star simulator used for attitude determination and guiding feedback. It imposes great demands of computing power for realistic domains, and modern Graphics Processing Units (GPUs) have demonstrated to be a powerful accelerator for this kind of computationally intensive simulations. This paper presents a parallel design and implementation of the intensity model applied to large-scale star simulators on GPUs using the compute unified device architecture (CUDA) programming model. The study analyzes the double parallel nature inherent in this model and use the CUDA framework to efficiently exploit the potential fine-grain data parallelism. Two versions of simulator are designed and studied: One is sequential simulator used as the baseline simulator, and another is parallel simulator using CUDA. In parallel strategy, model, and GPU implementation level, the study employs specific optimized strategies to efficiently improve the parallel performance. Finally, two benchmarks corresponding with the double parallelism are developed to fully evaluate the performance behavior of our simulators. The result analysis demonstrates the efficiency of the CUDA simulators and also illustrates the restriction and bottlenecks presented in this simulator.

**Key words**: GPU computing; CUDA; star simulator; blur effect; intensity model

**摘 要**: 灰度弥散模型被广泛应用于模拟星模拟器的成像过程.在实际问题域中,该模型需要巨大的计算能力以完成繁重的数值计算,而目前图形处理单元(GPU)已经发展成为一种有效的数值处理平台,对于计算密集型模拟具

* 基金项目: 国家自然科学基金(60303020); 国家高技术研究发展计划(863)(2009AA01Z303)
收稿时间: 2011-07-15; 定稿时间: 2011-12-02

有很好的加速能力.设计并实现了 GPU 平台下,基于统一计算架构(CUDA)的并行灰度模型,可应用于大规模星模拟器的快速灰度模拟.首先分析了该模型具有的双重并行特性,并采用 CUDA 模型模拟其良好的数据并行特征.为了便于对比研究,设计了两类模拟器:一类是串行模拟器作为基准模拟器;另一类是基于 CUDA 的并行模拟器.同时,在并行策略、模型以及 GPU 实现层面分别给出不同的优化方法以有效提高并行效率.最后,设计对应于双重并行粒度的两类测试基准,以评估并行模拟器的性能.数据分析表明,CUDA 并行模拟器取得良好的性能提升,同时也给出了该模拟器中存在的一些限制.

**关键词**: GPU 计算;CUDA;星模拟器;弥散效果;灰度模型

星模拟器是一种重要的航天模拟设备,主要用于模拟空间探测设备在任意时刻和任意视场指向下快速生成的星空图像.星空图像(简称星图)对于很多航天设备都有重要的应用,例如星敏感器[1]通过不断地拍摄星图进行实时姿态定位.另外,星模拟器还可以模拟太空背景星图,这在空间环境模拟系统有着广泛应用.灰度模拟是星模拟器进行精确星图模拟的关键步骤.为了能够获得真实感强的星图成像过程和较高精度的灰度值,实际模拟时广泛采用点扩散函数(PSF)来描述光学成像设备的弥散效应[2,3].当前,为了执行大规模星模拟器系统的灰度计算,基于弥散效应的星图灰度模拟需要数秒甚至分钟级别才能完成,达不到实时模拟的要求.这是因为,一个大规模真实感星图的灰度计算需要大量的弥散相关的数学计算,也即大规模星表中落入星图视场内每颗恒星都需要进行较多的数值计算.从 20 世纪 90 年代开始,针对大规模星图模拟需求,相继提出几个基于灰度模型的模拟器和软件系统来提高模拟速度[4−6].这些已有系统大多数基于串行架构设计并采用了几种设计语言,如 Pascal,C,GLSL 等.然而,所有的串行灰度模拟计算在执行上效率并不高,这是因为它们将模拟过程自身存在的并行度串行化,因此降低了系统的性能.

当前,处理器的发展促进了并行计算架构的不断进步.Intel 或者 AMD 最新的通用处理器片上已经包含 8 个处理核心,这些芯片通常用于处理集群,价格昂贵且能耗较高.然而,其他的并行架构正在逐渐作为替代的新计算环境.在这些并行环境中,可进行通用编程的 GPU 与传统的 CPU 计算集群相比,不但具有极高的浮点计算性能并且提供了大规模线程并行计算架构.对于大规模科学计算应用而言,这种架构特性能够很好地解决数据和计算密集处理的问题.通过 GPU 创建大规模并行环境,可支持从几百到几十万的活动并发线程.例如,当前 NVIDIA 公司的 GPU 可以支持片上 300 个标量处理单元,他们可以通过 C 和 CUDA 进行编程,提供了一种方便方式以进行 GPU 通用计算应用的开发.并且,基于 GPU 的并行计算与集群计算机相比,具有较低的花费[7,8].

为此,本文提出基于 GPU 的大规模星模拟器的灰度计算模型,通过对计算模型进行并行性分析,给出基于 CUDA 并行架构的并行算法,在并行策略、模型以及 GPU 实现层面分别给出不同的优化方法以有效提高并行效率.本文第 1 节介绍基于高斯弥散效应的真实感灰度计算模型.第 2 节详细给出基于 CPU 架构和 GPU 架构的模拟器的设计思路.第 3 节通过对比实验和测试用例对两种模拟器的性能进行对比分析.最后给出本文的结论和展望.

## 1 模型描述

星图灰度模型通过计算星表中落入成像面视场上的每颗恒星的亮度贡献值,最终得到一张星空灰度图像.在星图成像时,每颗恒星在模型中被看作距离成像面无限远的点光源.通过光学成像设备拍摄到的星图可以看作是成像系统对每颗映入成像面的星点的光学映射.在进行星图模拟时,计算灰度首先需要给出恒星作为点光源在像平面的光学亮度.在太空星表中,一颗恒星的亮度通过星等参数来表示,具体的亮度与星等的数学关系式如下:

$$g(m) = A \times 2.512^{-m} \tag{1}$$

式中 $A$ 是比例因子,$m$ 为恒星亮度星等(可观察范围通常为 0~15),$g$ 是恒星的亮度.

在实际模拟时,恒星的光亮度具有辐射效应,辐射能量分散在空间区域,并且分布规律符合弥散效应,即每颗星点在星图模拟时,其亮度辐射效应对星图上的每个像素灰度值都会贡献其亮度值.恒星能量弥散效应的数

学分析式称为点扩散函数(PSF).在空间相机光学系统,高斯点扩散函数[2,9](例如,高斯弥散效应)被用于准确地描述一个恒星的灰度分布率,它的数学表达式如下:

$$\mu(x,y)=\frac{1}{2\pi\delta^2}\exp\left[-\frac{(x-X)^2+(y-Y)^2}{2\delta^2}\right] \tag{2}$$

式中,$\delta$是反映辐射范围的光学参数,$(X,Y)$是辐射范围的中心也即恒星在像平面的坐标.点扩散分布在空间是对称的.$\mu(x,y)$是恒星$(X,Y)$在像素$(x,y)$的亮度贡献率.

通过高斯弥散函数可以看到,恒星的辐射分布随着距离的增加而快速衰减,也即距离星点较远的恒星获得该恒星的亮度贡献值微乎其微.因此,在计算像素灰度时,模拟系统通常采用控制辐射范围的策略:星点的灰度分布区域由整个像平面转变到感兴趣区域(ROI),它是以星点为中心的正方形区域(如图 1 所示).区域边长与相机光学参数有关,经验设定的范围在 2~20 像素之间.这样,只有在恒星感兴趣辐射区域内的像素才能接受到亮度辐射值.因此,恒星在像素$(x,y)$的亮度贡献值可以用恒星亮度值乘以亮度分布率得到,公式如下:

$$\varphi(m,x,y)=g(m)\times\mu(x,y) \tag{3}$$

式中,$(x,y)$限定在恒星感兴趣辐射区域的坐标范围内.

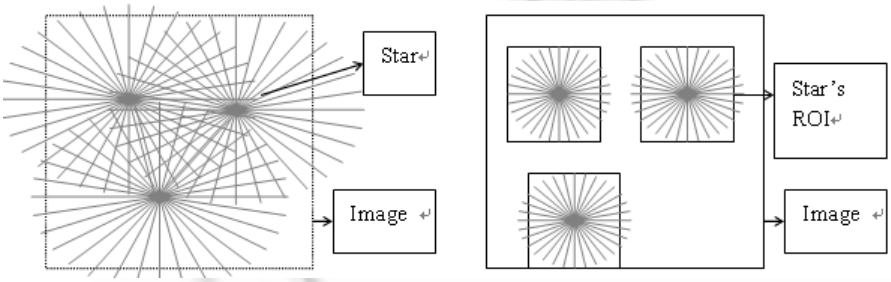


Fig.1 The effect of star brightness on image gray in different scopes: In the left part of the figure, every star scatters its brightness into the whole image, even out of the bound, which is very time-consuming in calculating the distribution; in the right-side figure, the distribution scope is restricted in a square, i.e. ROI. Most of the star's energy constrained in ROI area

图 1 恒星亮度辐射效应应用于不同的辐射区域:左图中每颗恒星的辐射区域都覆盖整个成像面区域,甚至超出图像边界,在计算其亮度贡献时耗时较长;右图恒星亮度辐射区域限定在方形区域(ROI)内,大多数恒星亮度的辐射能力集中在 ROI 区域内

## 2 模拟器设计

本节详细介绍基于高斯灰度弥散模型的星模拟器设计.首先,我们给出设计基于 GPU 的并行模拟器的先序工作-串行模拟器的设计;然后,给出 GPU 平台上的并行模拟器,设计基于 CUDA 的灰度并行计算模型,并且采用几种不同的策略实现高计算性能.

### 2.1 设计基准模拟器:串行模拟器

基于 CUDA 的编程模型以 C 语言为基础,因此,在设计 CUDA 并行应用时首先需要从一个基准 C 程序开始,然后将串行程序中的计算密集型部分转移到 GPU 上并行执行.串行模拟器的设计主要根据第 1 节的模型描述进行,思路较为直观.模拟过程可以分为 4 个阶段:恒星生成,星亮度计算,像素计算,像元输出.所有阶段都在模拟器上串行执行,也即在单 CPU 线程进行串行模拟.

首先,模拟器执行恒星生成步骤.在星图像平面视场上的恒星需要从星表中根据视场范围进行检索得到,每颗恒星包含两个参数:星等(探测范围通常为 0~15)和像平面上的恒星坐标.然后,星模拟器执行星亮度计算步骤,计算方法如模型描述中所述.灰度计算步骤执行星图每颗像素灰度值的计算,最后输出步骤将灰度值从 CPU 转移到 GPU 从而生成星图.

星模拟器的输入是一个恒星数据集,它主要通过星表中检索星图视场内的恒星组成.星表检索过程本文不再赘述,文献[4]中对检索算法有详细的分析.输出的像素灰度值写入到一种指定的图片类型,如 JPG,BMP 等,完

成一个星空图像的生成流程.

### 2.2 设计基于CUDA的GPU模拟器:并行模拟器

#### 2.2.1 并行策略

在灰度计算模型中,一个像素可能落入多个恒星的感兴趣辐射区域,其灰度值通过计算各个恒星在该像素的亮度贡献值得到.该模型的计算代价与恒星的数目和总的像素数是成比例关系.实际的星模拟器系统包含数万颗恒星和百万图像像素,这使得计算代价十分巨大.

在这个模型中,每颗像素灰度值计算与其他像素是独立的.每颗恒星的位置和亮度信息在计算感兴趣辐射区的灰度贡献值时可以重复使用.针对基于弥散效应的灰度计算问题,一个很好的分解策略是将灰度计算划分为子问题来计算每个像素的灰度值.所有的子问题计算相互独立,因此可以设计基于 CUDA 的大规模并发线程进行并行计算.

在并行算法中,并行粒度的组织需要分解整个计算任务到每个并行执行单元(CUDA 中的一个线程).在灰度计算模型中,有两种方法来组织并行执行方式:以恒星为中心和以像素为中心.对于以像素为中心的方式(如图 2(a)所示),每个线程对应一个像素,判断像素所在恒星的感兴趣辐射区域,然后计算恒星亮度辐射在像素的贡献值.这种方式不是最优的.因为每个线程要判断哪些恒星的亮度辐射区包含这个像素,这将产生很多的执行分支指令,在 CUDA 中,一个由 32 个线程组成的线程执行序列(Warp)中分支指令较多,会大大降低执行性能.以恒星为中心的模型(如图 2(b)所示)采用每个线程计算每颗恒星在其辐射区域内一个像素的亮度贡献值.这种方式是较优的,因为执行序列中避免了执行分支指令.另外,这种方式必须使用原子预防线程写冲突,也即防止多个线程对一个落入多个辐射区域的像素进行亮度贡献值的同时写操作.由于像平面上恒星的分布大体较为均匀,因此每个恒星的辐射区域的重复概率并不高,从而降低了原子操作的时间消耗.

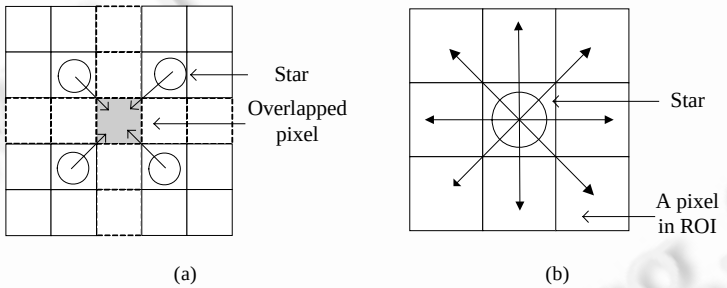


Fig.2 Two parallel computing modes: Pixel centric (a) and star centric (b); (a) shows four stars' ROI overlaid at one pixel. So each star contributes its brightness to this pixel, and there are many other star whose ROI excludes this pixel; (b) shows that one star contributes its brightness to each pixel in its ROI

图 2 两个并行计算模式:以像素为中心(a)和以恒星为中心(b);(a)中 4 个恒星的辐射区域 ROI 都包含了同一个像素点,因此每颗恒星对该像素都有亮度贡献值,而像平面其他恒星辐射区域都不包含该像素.(b)显示了一个恒星的亮度辐射区域以及它对每个像素的亮度贡献方式

#### 2.2.2 以恒星为中心的并行模型

在以恒星为中心的并行模型中,每个线程用来计算每颗恒星对其亮度辐射区域内的每个像素的亮度贡献值.在该模型中,计算恒星对每个像素的亮度贡献值是相互独立的,因此在每颗恒星的亮度辐射计算中存在一种并行粒度.这样,基于弥散的灰度计算模型本身固有两层并行性:恒星间并行(每个恒星的亮度计算是相互独立的)和每颗恒星亮度辐射区域内像素间并行(一个恒星对其亮度辐射区域内的各个像素亮度贡献值计算是相互独立的).两层并行粒度符合 CUDA 并行架构对应的并行模型(如图 3 所示).在基于 CUDA 的灰度计算模型中,每颗恒星对应一个线程块,块内每个线程对应恒星亮度辐射区域内的像素.每个线程块并行计算其所指代的恒星的亮度.每个线程块内的独立线程计算恒星对其辐射区域内它所指代的每个像素的亮度辐射值.在并行模型中,这种两层线程结构可以产生更多的执行线程,也即并行单元.

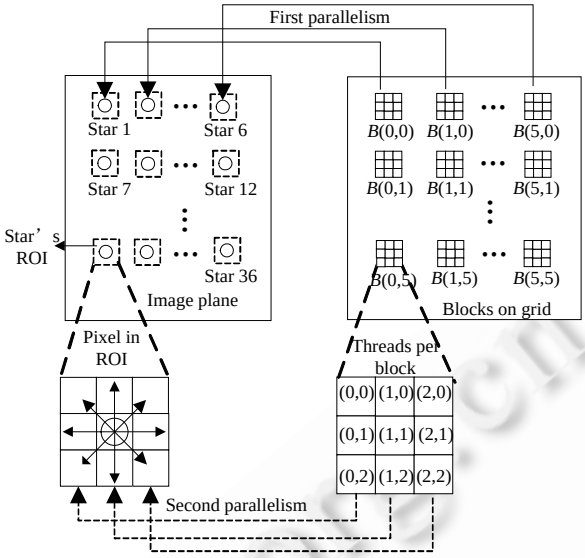


Fig.3 Parallel star-centric intensity model with Gauss blur effect. It configures a specific model: The stars num is 36 and ROI size is 3×3; First parallelism exists in block per grid and star per image, second parallelism exists in thread per block and pixel per star's ROI

图 3 基于高斯弥散的灰度计算模型.配置一个详细的模型参数:恒星数目为 36, 辐射感兴趣区域(ROI)大小为 3×3;第 1 层并行粒度存在于线程网格中的线程块与像平面上的恒星, 第 2 层并行粒度存在于线程块内的线程与恒星辐射区域内的每个像素

2.2.3 GPU 实现

并行模拟器的设计目标是在模拟星空图像时,尽可能地使灰度计算模型最大化的并行处理.为了达到这个目的,我们使用基准设计中灰度计算模型执行的 4 个阶段,并且第 1 步和最后一步由于有很少的计算要求而放在 CPU 平台执行,执行方式与基准模拟器相同;中间的两个计算密集型步骤放入 GPU 平台进行并行处理,实现基于 CUDA 的并行内核程序.

在设计内核程序时,我们应该按照 CUDA 执行模式部署以恒星为中心的并行灰度模型.灰度模型中的两层并行粒度通过定义两层线程结构来实现:*blocks*(每个线程网格内的并行线程块数)和 *threads*(每个线程块内的线程数).CUDA 中的 *blocks* 和 *threads* 可以定义为一维,二维和三维的.每一并行层次的维度要根据问题域的特点来调节.在灰度计算内核程序中,我们定义 *blocks* 为两维模式,这样能满足模型中对大规模恒星的模拟要求.每个线程块内线程结构(*threads*)也为两维模式.如图 4 所示,在串行基准程序中,采用两层循环计算一个恒星对其亮度辐射区域内的灰度贡献值(Step.7~Step.14);每个像素的 *x* 或者 *y* 坐标对应一个循环迭代器.并行模型中的两位线程结构(*threads*)能够完全匹配两层循环方式.图 3 给出了一个并行计算模型的详细二维线程架构实例.

在内核(kernel)执行之前,需要为内核程序提供输入和输出参数,并且提供指示参数防止并行线程在上下文执行环境的错误地址访问.具体来说,两个数据集作为参数确保内核的正确执行.首先给出图像相关的两个参数:图像像平面的大小(防止恒星亮度辐射区域边界越界)和指向像素输出数组的指针.其次,给出恒星相关的两个参数:恒星数目变量 *starcount*(防止线程块索引超过恒星数目)和指向装载像平面恒星信息的设备指针.

内核程序执行分为两个连续的步骤:恒星亮度计算和像素灰度计算.第 1 步计算每颗恒星的亮度,该亮度值可以计算第 2 步中每个恒星对像素的亮度贡献值.两个步骤结合起来即可得到每个像素的灰度值.图 5 给出了内核程序的伪代码.

```
1.  for ( i from 0 to starCount)
2.      mag ← starArray[i].mag;                                  /* read star magnitude*/
3.      starPosX ← starArray[i].posX;                            /* read star x-coordinate */
4.      starPoSY ← starArray[i].posY;                            /* read star y-coordinate */
5.      integer   pixelX;
6.      integer   pixelY;
7.      for (pixelY from starPosY-MARGIN to starPosY+MARGIN)     /* determine pixel y-coordinate*/
8.          for (pixelX from starPosX-MARGIN to starPosX+MARGIN) /* determine pixel x-coordinate*/
9.              if (pixelX & pixelY locate in the range of the image)
10.                 starBgt ←calculate star brightness;
11.                 imagePixelArray[pixelY*img_width+pixelX] +← calculate pixel gray contribution
12.         end for
13.     end for
14. end for
```

Fig.4 The loop inherent in CPU version: One-Loop way in identifying the star and two-loop way is used to iterate the pixel in each star's ROI

图 4 CPU 平台上程序的两层循环:一层循环对恒星亮度计算;第 2 次循环对应每颗恒星的亮度辐射区域内像素贡献值的计算

```
The kernel Pseudo-code of parallel simulator
Input:   Integer: image_width, image_height, starCount; star*starArray;
Output:  float* imagePixel
1.  __shared__ float  shareMem[3];
2.  threadX ← threadIdx.x, threadY ← threadIdx.y,        /* identify thread index in a thread block*/
    blockId ← blockIdx.x + blockIdx.y*gridDim.x          /* identify block index in thread grid */
3.  if ( blockId > = starCount) return;
4.  magnitude ← starArray[blockId].mag;
5.  if( threadX = =0 && threadY= =0)                     /* compute and store star's brightness */
        { shareMem[0] ← calculate the brightness of starArray[blockId];
          shareMem[1] ←  starArray[blockId].posX;
          shareMem[2] ←  starArray[blockId].posY; }
6.  __syncthreads();                                     /* synchronize all threads in this point*/
7.  starPosX ← shareMem[1];
    starPosY ← shareMem[2];
    pixelX ← starPosX – MARGIN +threadX                  /*compute each pixel position in each star's ROI */
    pixelY ← starPoxY – MARGIN +threadY                  /* MARGIN is the length of ROI */
8.  if ( pixelX & pixelY in the range of image)
        { grayDistribution ← compute the star's contribution on this pixel;   /* using PSF method */
          atomicAdd(& imagePixel[pixelY*image_width+pixelX], grayDistribution); }
9.  return imagePixel;
10. end kernel
```

Fig.5 The kernel pseudo-code of parallel simulator on CUDA: Each emphasized keyword in black bold type indicates a core technology or key step in designing the kernel

图 5 CUDA 上并行模拟器的内核伪代码:加黑的关键字代表内核设计中的核心步骤或者核心优化

执行第 1 步时,恒星的亮度和坐标信息通过线程块索引(*blockId*)访问得到,每个恒星的亮度在线程块间并行计算得到.每个线程块内的所有线程都会访问该线程块所指代的恒星的亮度值和像平面坐标,这是因为每颗恒星对其辐射区域内的每个像素都会贡献亮度值,而且辐射区域内的每个像素坐标通过恒星的坐标以及指代线程的索引得到.由于在进行像素灰度计算时恒星的星等和位置信息都不会修改,这意味着每个恒星的亮度值可以先行计算,然后与恒星的位置信息一起存入 GPU 片上的共享内存(shared memory).这样,所有同一线程块内的线程均可快速访问到恒星的亮度和位置信息.在实现时,通过每个线程块内的第 1 个线程计算出该线程块对应

的恒星的亮度值,然后存入到共享内存(图 5 中的步骤 5).此时,每个线程块内的所有线程需要同步以防止块内其他线程在共享内存未被写入数据时进行空读(图 5 中的步骤 6).通过部署线程块内的线程间共享内存,使得全局内存的访问频率从所有线程减少为 1 个线程的读取,并且使得亮度计算为每个线程块进行 1 次计算.该延迟和计算消减策略有效提升了模拟器的执行性能.

在第 2 步执行过程中,每个线程计算其指代的像素灰度值.首先,每个线程从片上共享内存读取恒星的亮度和像平面坐标.由于恒星位置需要进行两次读取(图 5 中的第 7 步和第 8 步),因此每个线程先从共享内存中将坐标信息读入片上寄存器,然后只需多次访问寄存器即获取坐标信息(图 5 中的步骤 7).这种方式可以缓解共享内存在不同线程同时访问时可能出现的 bank 冲突.从寄存器中读取恒星坐标,每个线程根据坐标信息和线程索引确定其指代的辐射区域内的像素坐标,然后通过增加每个恒星在该像素点的灰度贡献值修改每个像素的灰度值(图 5 中的步骤 8).需要指出的是,距离较近的恒星的亮度辐射区域可能重叠,因此在重叠区域内的像素操作可能因不同线程同时改变像素的灰度值而引起写冲突.为此,我们使用了原子操作在像素上加入一个动态内存锁,使得并行线程可以安全地进行共享数据的并发更改(图 5 中的步骤 8).原子操作可能因在等待内存读写时造成时间延迟,然而在执行灰度并行模型时,这种延迟被大规模并发线程的执行所覆盖.

在执行完内核程序后,我们需要将 GPU 上图像像素数组从全局内存输出到 CPU 内存.这会带来一定的通信代价.类似地,在执行内核程序之前,需要在主机和设备之间进行恒星和图像数据的通信.在异构并行系统中,数据传输虽然不可避免,但是应该通过采用一系列的 CUDA 通信优化策略以尽量减少通信延迟,文献[10]对此有详细介绍.

本文的 CPU 和 GPU 平台源代码可以从相关网页[11]中免费下载,并且使用说明也包含在网页内.

## 3 性能分析

本节分析了上述两种模拟器的性能:基于 C++的串行模拟器(称为模拟器 1)和基于 CUDA 的并行模拟器(称为模拟器 2).实验中所采用的 GPU 为 NVIDIA GPU GTX480,包含 480 个计算核心和 1.5GB 的设备内存,计算机服务器 CPU 芯片为 Intel core i7 @2.80GHz,内存为 3.5GB.虽然 CPU 片上有 8 个处理核心,为了能够准确地控制串行模拟器的执行,也便于进行清晰的比较,我们采用 CPU 片上的一个处理核心运行 CPU 上的串行模拟器.本文采用的 CUDA 编程环境为版本 3.2.

本文设计了两类测试(分别称为测试 1 和测试 2)以分析模拟器的性能:通过增加模拟星图上的恒星数目(相应地,线程网格上线程块数目也增加)和增加感兴趣辐射区域的边长(相应地,线程块内线程数目也增加).在实验中,模拟恒星数据通过随机生成程序得到,该程序可以生成像平面上随机分布的恒星文件,文件格式包含两个参数:恒星的星等和恒星的像平面坐标.正如第 2 节所述,每个线程块的线程数目与恒星的感兴趣辐射区域的大小相同;线程块的数目与像平面上的恒星数目相同.

图 6 显示了测试 1 中模拟器的实验性能.测试 1 增加像平面的恒星数目直到 $2^{17}$,这个恒星数目受到模拟器可提供的内存的限制.感兴趣辐射区域大小为 10×10,也即每个线程块上具有 100 个线程.其图片大小为 1024×1024.

两个模拟器的运行结果比较如图 6 所示.随着恒星数目的增大,模拟器 1 的执行时间按照线性方式快速增长;而并行模拟器 2 的时间消耗增长缓慢.当线程块的数目较少时,GPU 代码的执行性能与串行模拟器相比优势并不明显.这是因为,在配置该参数时,数据并行粒度较低,GPU 上的并行计算资源没有被充分利用.然而,随着线程块数目的增大,模拟器的数据并行粒度也在快速增加,因此,并行 GPU 代码与串行代码相比,性能快速得以提升.在恒星数目达到 $2^{14}$ 时,模拟器 2 相比于模拟器 1 获得了近 270 倍的加速比.此后,加速比一直保持稳定,这是因为线程块数目已经充分利用了 GPU 的计算管线.此外,内核程序的片上执行速度也获得较高数值,峰值达到了 95.07 gigaflops.从程序吞吐量方面来看,并行模拟器可以每秒处理约 950 亿次的像素灰度浮点来计算.在进一步分析 GPU 平台程序的各部分运行时间(见表 1)之后,我们可以看到内核执行时间随着恒星数目的增大而同步上升.而内存通信消耗保持基本稳定的状态,只有小幅度的浮动.这是因为,基于 Fermi 架构的 GTX480 提供了很

高的 CPU-GPU 和 DRAM 的通信带宽,增加的恒星数据集相对传输带宽较小,并且图像像素数组的大小保持不变,因此,传输消耗的变化并不显著.

图 7 给出了在测试 2 下模拟器的实验性能.测试 2 不断增加辐射区域的长度,直到 32×32.这个区域的大小与线程块上的线程数目是对应的.由于每个线程块上的线程数目最大为 1 024,这使得 ROI 的边长大小限制在 32.在模拟器中恒星数目设定为 8 192,也即每个线程网格中有 8 192 个线程块,同时,图像大小设定为 1024×1024.

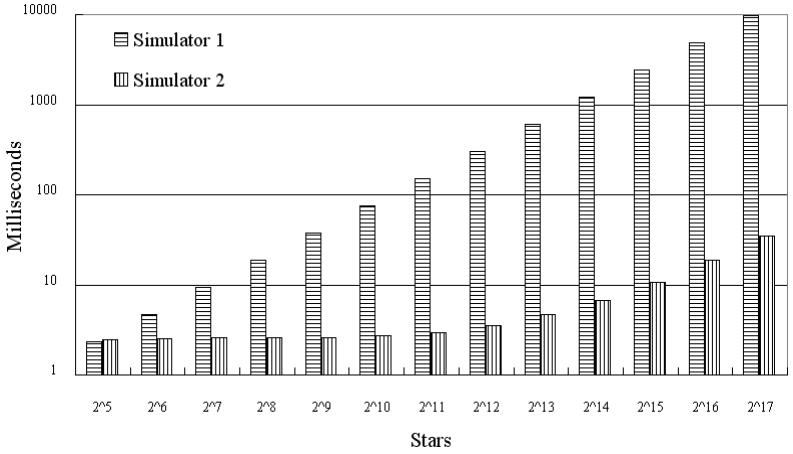


Fig.6 Simulation performance for sequential, parallel simulator: Test 1 (ROI's size is 10×10)

图 6 串行和并行模拟器的模拟性能:测试 1(ROI 大小为 10×10)

**Table 1** Breakdown of execution time on GPU for parallel simulator: Test 1

**表 1** GPU 上并行模拟器的各部分执行时间:测试 1

| | Star | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time (ms) | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ | $2^{11}$ | $2^{12}$ | $2^{13}$ | $2^{14}$ | $2^{15}$ | $2^{16}$ | $2^{17}$ |
| Kernel | 0.05 | 0.06 | 0.07 | 0.11 | 0.17 | 0.29 | 0.54 | 1.04 | 2.04 | 4.03 | 8.02 | 16.03 | 32.03 |
| Memory transmission | 2.43 | 2.43 | 2.43 | 2.43 | 2.53 | 2.59 | 2.64 | 2.65 | 2.53 | 2.66 | 2.62 | 2.82 | 2.73 |

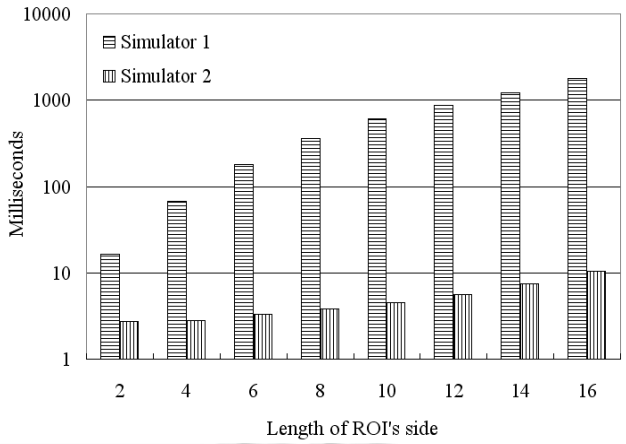


Fig.7 Simulation performance for sequential, parallel simulator: Test 2 (stars=8192)

图 7 串行和并行模拟器的模拟性能:测试 2(stars=8192)

当线程块内线程数目增加时,GPU 代码的执行时间呈指数式增加,一旦所有的 GPU 资源被全部占用,那么执行时间随着线程块数目呈线性增加.在该测试中,模拟器 2 相对于模拟器 1 获得了近 180 倍的加速比.虽然随着 ROI 的增大原子操作增多,相应的消耗也在增加;然而线程数目增加的速度更快,使得大规模的并行线程隐藏了同步延迟.通过分析 GPU 平台程序各个部分的运行时间(见表 2),我们可以看到,内核执行的时间随着 ROI 的

增加而同步增大,然而,CPU 与 GPU 间的通信消耗仍然保持稳定的状态.这是因为恒星的数目和像平面像素数组保持不变,因此设备内存与 CPU 全局内存的输入/输出通信保持一个稳定的状态.但是,值得注意的是,模拟器 2 的内存之间的通信耗费相对于内核执行仍然较高,特别是在线程数目较小时,这种比例更加明显.因此,在模拟一个小规模的星空图像时(恒星数目在 $0\sim2^7$),CPU 串行模拟器具有较好的性能优势,因为这种情况下 GPU 的并行资源没有被很好地利用,同时还增加了异构系统的通信耗费.

**Table 2** Breakdown of execution time on GPU for parallel simulator: Test 2

**表 2** GPU 上并行模拟器的各部分执行时间:测试 2

| Time (ms) | ROI | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 |
| Kernel | 0.27 | 0.35 | 0.81 | 1.26 | 2.04 | 2.95 | 5.02 | 7.87 |
| Memory transmission | 2.43 | 2.46 | 2.41 | 2.53 | 2.47 | 2.50 | 2.44 | 2.55 |

## 4 结论及展望

本文分析和设计了模拟星空图像的两种模拟器.第 1 种模拟器(模拟器 1)基于 CPU 平台串行执行高斯弥散灰度计算模型的计算过程.第 2 种模拟器(模拟器 2)采用 CUDA 架构设计基于 GPU 的并行模拟器.模拟器 2 相比模拟器 1 最高获得了接近 270 倍的加速比.实验结果表明,由于灰度计算模型本身具有良好的二维数据并行特性,并且 GPU 平台可以很好地进行星图灰度模拟计算.第 1 层数据的并行粒度存在于像平面的恒星之间,它对应了 CUDA 线程结构中的线程块;第 2 层并行粒度存在于每颗恒星的感兴趣辐射区域的像素之间,它对应了 CUDA 中的线程块内线程.

然而,需要指出的是,并行模拟器 2 模拟参数受到 GPU 可用资源限制(设备内存,GPU 与 CPU 通信带宽).内存大小限制了像平面可模拟的恒星数目.CPU-GPU 通信耗费部分影响了 CPU/GPU 异构系统的应用程序性能.在以后的工作中,将进一步减小模拟所需内存和传输代价以期能够处理更大的星图模拟,并获得更好的性能.同时,虽然 GPU 已经提供了很好的并行计算环境,GPU 集群能够提供更高并行粒度的执行环节,因此我们将致力于将并行模拟器扩展到多 GPU 上,以获得更多内存执行空间和更好的性能.

**致谢** 该项目得到国家高技术研究发展计划(863 计划)(2009AA01Z303)和国家自然科学基金(60303020)的资助.同时,我们也向 Jose M.Cecilia 所做的创造性的工作以及对本文工作的启发致以谢意.

**References**:

[1] Liebe CC. Star trackers for attitude determination. IEEE AES Systems Magazine, 1995,10(6):10−16.

[2] Poropat GV. Effect of system point spread function, apparent size, and detector instantaneous field of view on the infrared image contrast of small objects. Optical Engineering, 1993,32(10):2598−2607.

[3] Liebe CC. Accuracy performance of star tracker-atutorial. IEEE Trans. on Aerospace and Electronic Systems, 2002,38(2):587−597.

[4] KIM H-Y, Junkins JL. Self-Organizing guide star selection algorithm for star trackers: thinning method. In: Woerner DV, ed. Proc. of the 2002 IEEE Aerospace Conf. Montana: IEEE Press, 2002. 2275−2283.

[5] Zhang SD, Sun HH, Wang YJ, Jia XM, Chen H. Design of high precision star image locating method used in star sensor technology. In: Proc. of the 2010 Int'l Conf. on Computer, Mechatronics Control and Electronic Engineering. Changchun: IEEE Press, 2010. 411−414.

[6] Yang YD, Wang JY, Zhu YT. High-Precision simulation of star map using forward ray tracing method. In: Proc. of the 9th Int'l Conf. on Electronic Measurement & Instruments. Beijing: IEEE Press, 2009. 541−544.

[7] Garland M, Grand SL, Nickolls J, Anderson J, Hardwick J, Morton S, Phillips E, Zhang Y, Volkov V. Parallel computing experiences with CUDA. IEEE Micro, 2008,28(4):13−25.

[8] Cecilia JM, Garcia JM, Guerrero GD, *et al*. Simulating a P system based efficient solution to SAT by using GPUs. The Journal of Logic and Algebraic Programming, 2010,79(6):317−325.

[9] Liu TY, Wang SC, Liu XM. Attitude information deduction based on single frame of blurred star image. In: Proc. of the 2nd Int'l Conf. on Future Computer and Communication. Wuhan: IEEE Press, 2010. 642−646.

[10] NVIDIA. NVIDIA CUDA Programming Guide 3.2: CA, Nvidia Corporation, 2010. 81–93.

[11] http://code.google.com/p/high-performance-star-image-processing/

**李超**(1988－),男,江苏赣榆人,硕士,CCF 学生会员,主要研究领域为高性能并行计算,GPU 通用计算,程序并行与优化.

**郑昌文**(1969－),男,研究员,博士生导师,CCF 高级会员,主要研究领域为计算机仿真,信息处理.

**张云泉**(1973－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为大型并行数值软件,并行程序设计和性能评价,并行计算模型,并行数据挖掘.

**胡晓慧**(1960－),男,研究员,主要研究领域为信息系统集成.