

基于 CUDA 平台的伪随机数产生器系统研究

郭海凤^{1,2}

(1. 金陵科技学院 信息技术学院, 江苏 南京 211169;
2. 江苏省信息分析工程实验室, 江苏 南京 211169)

摘要:针对线性同余算法在产生海量随机数序列时,速度较慢的情况,文中提出并实现了一种产生伪随机数的改进方法,即基于 CUDA 平台的并行线性同余法,测试证明改进算法可行。该方法通过利用 GPU 的并行架构,实现并行产生伪随机数。通过实验结果比较,与传统的线性同余算法相比,改进算法产生的随机序列周期较长、速度较快,当其产生大量伪随机数序列时其加速比接近 100,解决了传统线性同余算法产生伪随机数周期短、速度较慢的问题,系统仿真效果较优。

关键词:伪随机数;线性同余;并行计算;加速比

中图分类号:TP391.9

文献标识码:A

文章编号:1673-629X(2013)02-0115-04

doi:10.3969/j.issn.1673-629X.2013.02.029

Research on Pseudo-random Number Generator Based on CUDA Platform

GUO Hai-feng^{1,2}

(1. College of Information Technology, Jinling Institute of Technology, Nanjing 211169, China;
2. Jiangsu Information Analysis Engineering Laboratory, Nanjing 211169, China)

Abstract:For linear congruence algorithm (LCA) has low speed of producing huge amounts of random number sequences, propose a parallel pseudo-random number generator based on CUDA platform. Test proved the improved algorithm is feasible. The method by using GPU parallel architecture generates parallel pseudo random number. Compared with traditional linear congruence algorithm, improved algorithm has longer cycles and faster speed of generating random sequence, its acceleration ratio is close to 100, solves the slow speed of traditional algorithm. And its system simulation shows it is better than the traditional one.

Key words:pseudo-random number; linear congruence; parallel computing; accelerator

0 引言

随机数广泛应用于概率统计、信息安全、遥测遥控、随机性能仿真、数字通信以及码分多址系统等重要领域。而在这些领域研究中,很难将所有的随机信息全部采集到,因此需要随机数发生器来模拟现实中的随机现象。随机数的质量与产生速度直接决定着实际的仿真效果^[1,2]。

目前,线性同余算法是伪随机数发生器较为常用的一种算法,它产生出来的随机数序列周期性长,随机性较好^[3]。由于该算法是建立在串行计算的基础上产生随机数的,当需要产生海量的随机数时,计算速度势必减慢,具体数据可参见表 1,在很多应用中难以达到

实际仿真的效果,不能满足现实需求。若要使伪随机数产生器更真实地模拟出现实中的随机现象,势必提高其产生随机数序列的速度。

文中在基于线性同余算法产生伪随机数的基础上提出了一种改进算法,基于 CUDA。在改进算法中,将串行产生随机数方式改为并行产生方式,提高了产生随机数的速度。通过建立系统仿真模型实验,与普通的线性同余算法产生的随机数进行了分析和对比。结果表明,改进算法效果较优。

1 CUDA 平台体系结构

基于线性同余法产生随机数之所以速度慢的原因主要在于其在产生随机数序列时是串行化的,随机序列具有一定的序列性和周期性,所用的时间是各个随机数所用的时间之和^[4]。这里,可以假设产生每个随机数所用的时间为 n ,随着计算机硬件技术的不断发展,CPU 计算速度也越来越快,即 N 的值变的越来越

收稿日期:2012-05-03; **修回日期:**2012-08-06

基金项目:国家自然科学基金资助项目(61075049);江苏省高校自然科学基金项目(11KJD520006)

作者简介:郭海凤(1980-),女,硕士,研究方向为数据挖掘、信息处理、信息检索。

小(接近 0)。在产生一定数量的随机序列时在时间上已无明显的区别,速度都基于毫秒级。在仿真效果上基本上能满足系统需求。但是当产生海量(上亿级)的随机数时,即使 N 的取值再小,其累加之和也是不小的数字,速度差别越来越明显,在仿真效果中,产生随机序列的时间已不能满足系统的需求。若要加快随机数的产生速度,则需要采用并行处理的方式。

CUDA 是 NVIDIA 的 GPGPU 模型,为加速图像的实时处理而设计的一种运行在 GPU 上的开发平台,现代的芯片已经具有高度的可程序化能力,并且具有相当高的内存带宽以及大量的浮点计算单元,可以大量处理并行化的问题,尤其是在大型浮点型数据计算方面优势比较显著。CUDA 是一种由 NVIDIA 推出的通用并行计算架构,若将其用于产生随机数序列仿真,势必可以起到事半功倍的效果^[5]。

在 CUDA 的平台下,一个并程序总的来说可以分为两个部分:host 和 device,其中 host 是指在 CPU 上面运行的部分,而 device 则是指在显示芯片上执行的程序,又称之为“kernel”。程序执行时,会将 host 端的数据复制到显卡的内存中,由显卡芯片来执行计算 device 端的程序,待计算完成后,host 端程序将计算结果从显卡芯片中取回^[6,7]。因此在计算很少的执行单元时,其 host 端与 device 端的互存的复杂度和仅仅在 CPU 上计算相当,甚至还要低,因此对于少量的计算来说,GPU 计算并不能够显示出绝对的优势,这点在后面的实验中通过数据也得到了证明。

在 CUDA 架构下显示芯片执行的最小的单位是 thread,数个 thread 可以组成一个 block。一个 block 中的 thread 能存取同一块共享的内存,而且可以快速进行的动作。每一个 block 所能包含的 thread 数目是有限的,执行相同程序的 block,可以组成 grid^[8]。在产生随机数序列时,对于一台计算机可认为其是一个 grid,即在一个 grid 情况下产生多个 block,不同 block 中的 thread 无法存取同一个共享的内存,因此无法直接互通或者互相同步,不过,利用这个模式,可以让程序不用担心显示芯片执行的 thread 数目限制^[9]。因此可以基于 CUDA 平台实现产生伪随机数序列。这里可以把 thread 作为产生随机序列的一个随机数,block 作为一个随机算法,grid 看做 GPU。这样在 CUDA 平台下产生伪随机数序列模型如图 1 所示。

由于每个 block 随机种子的选取不同,这样,产生的随机序列独立性较强、随机周期变长、随机数质量提高了。

2 伪随机数产生算法

随机数按特点可分为真随机数和伪随机数。真随

机数是一系列毫无规律可循的随机数,具有随机性、不可预见性、无周期性^[10]。真随机数产生器一般都是基于无源器件中的热噪声,通过噪声放大器,经过一系列处理数据处理后得到^[11]。伪随机数则具有一定的规律性,甚至通过推演可以预测到。伪随机数产生器一般都是基于软件技术来实现。与真随机数相比,伪随机数在实现技术上相对简单一些,应用也更为广泛,目前基于软件所产生的随机数都是伪随机数^[12]。

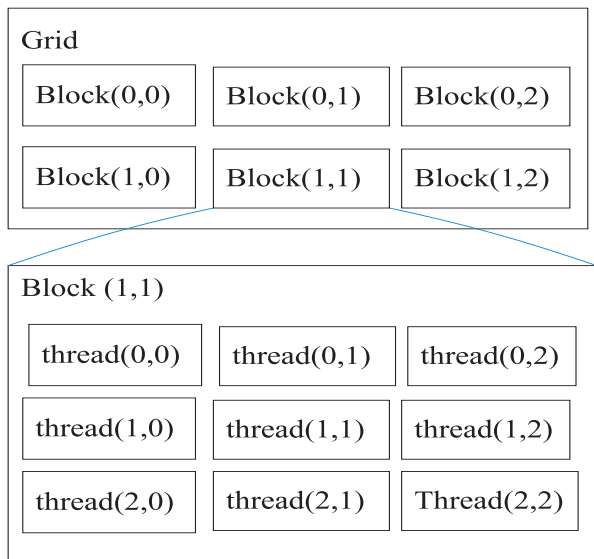


图 1 CUDA 平台下产生伪随机数序列模型

2.1 线性同余算法

线性同余算法是产生伪随机数常用的一种算法,其递推公式如(1)所示:

$$x_n = (a * x_{n-1} + c) \% m \quad (1)$$

可以把式(1)写成式(2)所示:

$$x_n = (a * x_{n-1}) \% m + c \% m \quad (2)$$

式(2)中,模数 m 和乘数 a 决定着产生随机数序列的概率分布情况;随机序列分布周期与 c 无关,增量常数 c 对随机数的序列影响不大。模数 m 的选取要尽可能的大,这样随机输出序列周期才能足够长^[8]。在实际应用中, m 通常选择计算机能表示的最大整数,如接近或等于 2^{31} 。

2.2 基于 CUDA 平台产生随机数

为了提高 GPGPU 的运行效率,平衡线程之间的负载,需要对 CUDA 平台的 GPU 并行计算产生的随机数^[5]序列进行任务划分,假设需要产生伪随机数的数量为 N ,开辟的线程(thread)数为 T ,那么每个线程所产生的随机数序列长度为 N/T ,那么算法的复杂度为 $N/T + T$,而在串行计算中,其复杂度为 N 。若想在基于 CUDA 平台取得优势,线程数 T 起着至关重要的作用。于是 GPGPU 模型优化可以理解为取函数的最大值求解,即令 $f(T) = N/T + T - N$,由于 N 为常数,因此只需要求出 $f(T)$ 取得最大值时 T 的取值即可。需要

说明的是,在目前的 CUDA 中每个 block 的 thread 数目最多为 512,当 $T > 512$ 时候,需要开辟新的 block。

2.3 算法实现

由上述知,线性同余算法随机序列 x_n 的实现关键在于式(3)的实现。

$$x_n = a * x_{n-1} \% m \tag{3}$$

迭代式(3)可写成式(4)形式

$$x = f(x) = (a \cdot x) \% m \tag{4}$$

令 $m = a \cdot q + r$,其中 $q = \lfloor \frac{m}{a} \rfloor$, $r = m \% a$,则

$$f(x) = a(x \% m) - r \cdot \lfloor \frac{x}{q} \rfloor + m(\lfloor \frac{x}{q} \rfloor - \lfloor \frac{a \cdot x}{m} \rfloor) \tag{5}$$

(5)式较好地解决了高次同余溢出问题,但是在计算中引入了 $\frac{a \cdot x}{m}$ 式,增加了浮点型计算,精度要求

更加精细,通过 CPU 计算时,需要更多的内存开销,尤其是当产生较多伪随机数时,速度会很大程度上降低。由于 CUDA 平台的 GPU 计算在处理浮点型数据计算方面较有优势,若能通过 CUDA 平台来并行计算随机数产生序列,势必在产生随机数序列速度上提高优势。由于式(5)是建立在串行的基础上,并不适合 CUDA 平台进行并行计算,这里需要将(3)式并行出两个序列:

$$f(x_1) = (a_1 \cdot x_1) \% m \tag{6}$$

$$f(x_2) = (a_2 \cdot x_2) \% m \tag{7}$$

令参数 $\alpha = a_0/a_1$, $\beta = a_0/a_2$,于是有

$$a_0 \cdot x_0 = \alpha \cdot a_1 \cdot x_0 = a_1 \cdot x_1,$$

$$a_0 \cdot x_0 = \beta \cdot a_2 \cdot x_0 = a_2 \cdot x_2$$

由(5)式知随即数产生序列不在受乘数限制,因此只要保持 m 一致, $f(x_0)$, $f(x_1)$, $f(x_2)$ 相互独立。以此类推,当增加线程数目来进行并行计算时候,由于彼此间相互独立,不受随机序列的影响。这样就可以利用 CUDA 平台的 GPU 并行计算对产生随机数序列进行任务划分了。

2.4 相关实验

随机数的种子不同,从不同线程产生的随机数序列也就不同,因此随机数种子是算法的伪随机性关键。种子参数可以任意选择,常常将它设为计算机当前的日期或者时间,为了防止各个线程产生的随机序列相同,可以让每个线程的种子参数增加其它系数,文中实验中种子参数 $x_n = n \cdot f_t$,其中 f_t 表示当前的时间。表

1 是分别用线性同余法分别基于 CPU 平台和 CUDA 平台计算相同数目的随机数时所进行的性能比较。

为了便于统计随机序列的分布情况,试验中将随机数取值范围限定在 0 ~ 100 之间,分别统计产生不同随机数时每个随机数的频率分布,图 2 是产生 262144 个随机数时每个随机数序列的频率分布图。

表 1 算法性能比较

测试集数目(10^3)	运行时间(S)		加速比
	CPU	GPU	
512	0	0.0014	0
2048	0.047	0.0118	3.980
16384	0.3589	0.0415	8.6482
65536	1.42	0.0617	23.2026
262144	5.719	0.0651	87.8044

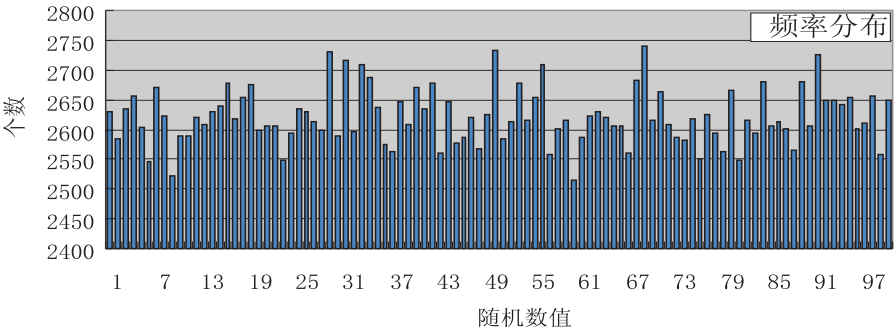


图 2 随机数序列频率分布

2.5 结果分析

衡量随机数序列的主要指标有周期性、相关性、随机性(也有称之为“均匀性”)以及系统产生随机数所需要的时间^[5]。

2.5.1 周期性比较

决定周期性因素主要在于初始种子的选择,与系统开发平台关系不是很大。由于文中基于 GPU 平台产生的随机数是多线程并行计算的,初始种子的选择为系统的当前时间,因此当产生少量的随机数序列时,在周期性方面与基于 CPU 计算的随机序列周期性相当;而当产生较多的随机序列时,GPU 的线程一次性并不能够负载完毕,这样系统的时间就会有一定的差别,因而各个线程的周期性不同。这样就打破了原来随机序列中固有的周期性质,周期性会增加。

2.5.2 相关性比较

GPU 体系结构产生随机数的算法是并行计算的,上述中也证明了各个并行线程之间是相互独立的,因此基于 GPU 的并行计算所产生的随机数之间也是相互独立的;在 CPU 计算中,是基于串行计算的,相互间存在着联系^[6],而实际的随机序列之间是相互独立的,因此在相关性方面,GPU 计算要优于 CPU。

2.5.3 随机性比较

通过实验统计,分别统计产生不同随机数时每个随机数的频率分布图(如图 2 所示),可以很直观地看出,基于 CUDA 平台的 GPU 计算产生随机数序列时服从均匀分布的。这点与 CPU 的串行计算差别不大。这样就能够保证该算法产生的随机序列可行。

2.5.4 加速比

通过消耗时间对比则会发现 CUDA 平台的巨大优势,如表 1 所示,当产生数额较少的随机序列时,并不能够充分发挥出 GPU 的优势,在计算速度上甚至低于 CPU,这是因为在 CUDA 的架构下,程序分为 host 端和 device 端,host 端是指在 CPU 上运行的部分,在内存占用方面与 CPU 计算悬殊不大,尽管 device 端在并行计算上有优势,但是由于计算线程少,使用效率低,因而在运行速度上显示不出优势;而随着产生随机数数额的增多,其优势逐渐显示出来,产生 262144000 个随机数时,加速比已经达到 87.8。

通过上述的周期性比较、相关性比较、随机性比较以及计算加速比,可以很明显地看出,基于 CUDA 平台的 GPU 并行计算在产生大量随机数序列方面优于传统的 CPU 计算。

3 结束语

伪随机数是系统仿真中重要的技术之一,尤其对其产生速率的研究意义深远,通过实验比较,文中提出的基于 CUDA 平台产生伪随机数算法在产生大量随机数时可将其产生速率提高近百倍,具有一定的实用价值。但目前基于 CUDA 平台的 GPU 并行计算方面尚有许多限制,对于许多不能并行优化的工作,所带来的帮助不大,另外目前显示芯片并没有分开的整数运算

单元,在整数计算方面,效率较差。除此之外,目前的 GPU 并行计算模型仅仅处于发展阶段,若想更深层次的应用,仍需要进一步的学习和探讨。

参考文献:

- [1] 朱晓玲,姜 浩. 任意概率分布的伪随机数研究和实现[J]. 计算机技术与发展,2007,17(12):116-118.
- [2] 杨 益,方潜生. 基于 Handel-C 的伪随机数发生器的设计与实现[J]. 计算机技术与发展,2006,16(12):124-126.
- [3] 左颢睿,张启衡,徐 勇,等. 基于 GPU 的并行优化技术[J]. 计算机应用研究,2009,26(11):4115-4119.
- [4] 辛 茜,曾晓洋,张国权,等. 真随机数发生器的系统建模与仿真[J]. 系统仿真学报,2005,17(1):53-56.
- [5] 谭彩凤,马安国,邢座程. 基于 CUDA 平台的遗传算法并行实现研究[J]. 计算机工程与科学,2009,31(1):165-170.
- [6] 马 华,张晓清,张鹏鸽. 一种基于线性同余算法的伪随机数产生器[J]. 纯粹数学与应用数学,2005,21(3):202-207.
- [7] Wong M L, Wong T T. Parallel Hybrid Genetic Algorithms on Consumer Level Graphics Hardware[J]. Evolutionary Computation, 2006, 21(16):2972-2980.
- [8] 陈国良. 并行算法的设计与分析[M]. 北京:高等教育出版社,2002.
- [9] Matthews R. On the derivation of algorithm "chaotic" encryption algorithm[J]. Cryptologia, 1989, 8(1):29-41.
- [10] Ferretti A, Rahman N K. A study of coupled logistic map and its applications in chemical physics [J]. Chemical Physics, 1988, 119(2/3):275-288.
- [11] 楼久怀. 不同分布的随机数发生器的研究和设计[D]. 杭州:浙江大学,2006.
- [12] 栾忠兰,吕 强. 伪随机数发生器的统计性质检验及其应用[J]. 计算机应用与软件,2011,27(10):168-170.
- [8] 曹莉华,柳 伟,李国辉. 基于多种主色调的图像检索算法研究与实现[J]. 计算机研究与发展,1999,36(1):96-100.
- [9] 王耀南. 小波神经网络的遥感图像分类[J]. 中国图象图形学报,1999,4(5):368-371.
- [10] 陈耀东,王 挺,陈火旺. 半监督学习和主动学习相结合的浅层语义分析[J]. 中文学习学报,2008,22(2):70-75.
- [11] 毛建旭,王耀南. 基于神经网络的遥感图像分类[J]. 测控技术,2001,20(5):29-31.
- [12] Blum A, Mitchell T. Combining labeled and unlabeled data with co-training[C]//Proceedings of the Workshop on Computational Learning Theory. [s.l.]:[s.n.],1998:92-100.
- [13] 陈东泳,钟尚平. 基于半监督学习的 JPEG 图像通用隐写检测方法[J]. 计算机技术与发展,2009,19(2):169-172.

(上接第 114 页)