Pollard-rho 算法及其并行实现

张程
西北农林科技大学
杨凌, 陕西, 中国
operatorcheng@nwsuaf.edu.cn

1 简介

给定两个质数，要算出它们的乘积很简单，但反过来给定一个整数，要将其分解成多个质数相乘就不那么容易了，特别是当给定的整数非常大，大到几十位甚至上百位时，以现有的算法和计算能力，所需的计算时间将达到几十年到上百年，这显然无法让人接受。一些加密算法正是依靠这种整数分解的复杂性实现良好的保密性的，例如 RSA 公开密钥密码体制 [4]，其基本思想是根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

对整数进行因式分解的算法有很多，其中最经典的是试除法，即从 2 到 $\sqrt{n}$ 逐一对要分解的整数 N 进行除法，如果余数为零，则说明找到了这个整数的一个因数，其时间复杂度为 $O(\sqrt{n})$。Pollard-rho 算法是用于将一个合数分解成多个质数因子的算法，由 John Pollard 于 1975 年发明 [3]，其时间复杂度为 $O(n^{\frac{1}{4}})$。该算法是一种随机算法，基于随机数进行求解，因此在有限的时间内，该算法不一定能找到一个问题的解，但如果找到一个解，那么这个解必定是问题的解，且算法的成功率随着求解时间的增加而增加。

2 算法原理

Pollard-rho 算法是一种随机算法，如果只按照随机算法的基本思想对整数分解的问题进行求解，就随机生成一些数，判断它们是不是要分解的数的因子，如果是，输出该因子并退出，如果不是，则继续下一轮的随机数生成和判断，直到找到因子或执行到一定时间或次数。但这种方法的效率比试除法还要低，因此需要加入其他方法来提高效率。Pollard-rho 算法则是通过引入生日悖论的思想来提高找到因子的概率。

2.1 生日问题

考虑这样一个问题，现在一个房间里有 n 个人，$n < 366$，假设生日是均匀分布的，且不考虑特殊因素，如闰年和双胞胎等，问房间中有两个人的生日相同的概率 $P(n)$ 是多少。直接计算 $P(n)$ 比较困难，我们可以先从问题的反面入手。设 $\bar{P}(n)$ 表示 n 个人中任意两个人的生日都不同的概率，则 $\bar{P}(n)$ 可以通过下式计算：

$$\bar{P}(n) = \frac{365}{365} \cdot \frac{364}{365} \cdots \frac{365-n+1}{365} = \frac{365!}{365^n(365-n)!} \quad (1)$$

因此 $P(n)$ 可以通过下式计算得出：

$$P(n) = 1 - \frac{365!}{365^n(365-n)!} \quad (2)$$

通过式 (2) 可以计算出 n 与 $P(n)$ 的对应关系，如表 1 所示。从表 1 中可以看出，当 $n = 23$ 时，房间中有两个人生日相同的概率已经超过了 50%，而当 $n = 60$ 时，几乎可以肯定房间中必然存在生日相同的两个人。实际的出生记录表明，在不同日期出生的人数不同，在这种情况下，可以证明一群人中有两个人生日相同的人数阈值为 23 或更少 [1]。通过生日问题可以得出，对于随机问题来说，少量提高采样数量就能大幅提高命中几率。

n	23	30	40	50	60	70
P(n)	50.7%	70.6%	89.1%	97.0%	99.4%	99.9%

表 1: n 与 $P(n)$ 的对应关系

2.2 Pollard-rho 算法思想

Pollard-rho 算法的核心思想即是利用了生日悖论来提高随机算法的效率，我们不再随机生成一个数看它是否是因子，而是随机生成 k 个数，看其中任意两个数之差的绝对值是否是因子。此时虽然找到因子的概率提高了，但是算法的复杂度也提高了，因为 k 个数中任意两个数的组合有 C_k^2 种，我们还是要做大约 k^2 次比较和除法。为了避免这种情况出现，我们可以不停地生成随机数，而只求连续的两个随机数之差的绝对值，并判断结果与要分解的整数的最大公约数是否大于 1，如果是则说明找到了一个因子，该因子就是两个随机数的差的绝对值。我们可以使用一个模 n 的多项式生成一系列的伪随机数：

$$g(x) = (x^2 + 1)\ mod\ n \tag{3}$$

通过给定初始值并进行迭代，我们可以通过式 (3) 得到一系列的数，如 $x_1 = 2$，$x_2 = g(x_1)$，$x_3 = g(g(x_1))$ 等。由于这些生成的序列中可能的值的数量是有限的，所以序列 $\{x_k\}$ 最终会进入循环，如图 1 所示，这也是算法名称中“rho”的由来。我们可以通过 Floyd 的循环

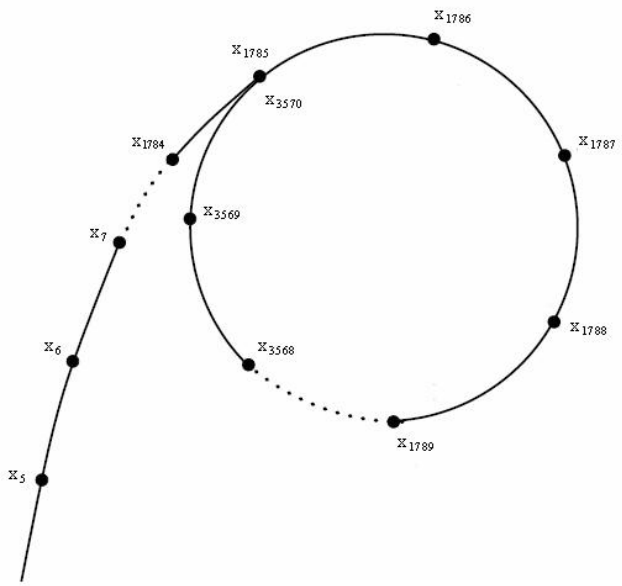

图 1: 序列进入循环示意图，形似希腊字母 ρ

查找算法来检测当前状态是否进入了循环，该算法被 Donald Knuth 认为是 Robert W. Floyd 发明 [2]，其主要思想是让乌龟和野兔在一个环形跑道上同一地点出发，以不同的速度前进，当野兔第一次追上乌龟时，就知道野兔至少已经领先乌龟一圈了。应用到 Pollard-rho 算法中，我们可以使另一个序列 $\{y_k\}$ 以原始序列 $\{x_k\}$ 的两倍速度生成，即 $x_2 = g(x_1)$，$y_2 = g(g(x_1))$，当第一次 $x_i = y_i$ 时，我们就知道已经进入了循环，此时需要停止算法或重新选择不同的 $g(x)$。

3 算法实现

在本节中首先使用 C++ 语言对 Pollard-rho 算法进行串行实现，然后对该算法的可并行性进行分析，设计并行方案，并使用 CUDA 开发套件实现 Pollard-rho 算法的并行化。

3.1 串行实现

为了提高算法中伪随机数生成的稳定性和随机性，使用 C11 标准新增的 random 库来生成伪随机数，random 库解决了传统伪随机数生成方法的随机性差等问题，可以生成任意区间的伪随机数，且具有良好的随机性。串行实现的算法流程图如图 2 所示，首先输入要分解的整数 N，由于 C++ 语言的数据大小限制，N 的范围为 0 到 18446744073709551615。然后通过随机数生成函数生成范围为 1 到 N 的两个随机数，设置主循环的次数，存入变量 r1 和 r2 中，然后进入主循环，通过辗转相除法求出 r1 和 r2 的差的绝对值与 N 的最大公约数，存入 g[i] 中。循环结束后，再逐一检查 g[i]，如果 g[i] 大于 1，说明找到了整数 N 的一个因子，输出 g[i] 后算法停止。

3.2 并行实现

在串行实现的算法流程图中有三个循环，分别是生成数据、求最大公约数和检查结果。其中求 k 个数与 N 的最大公约数占据了算法中的大部分运算，且求各个数与 N 的最大公约数之间没有相互依赖，适合并行执行。首先设计核函数，核函数的功能为使用辗转相除法计算 g[i] 与 N 的最大公约数，并将结果写回

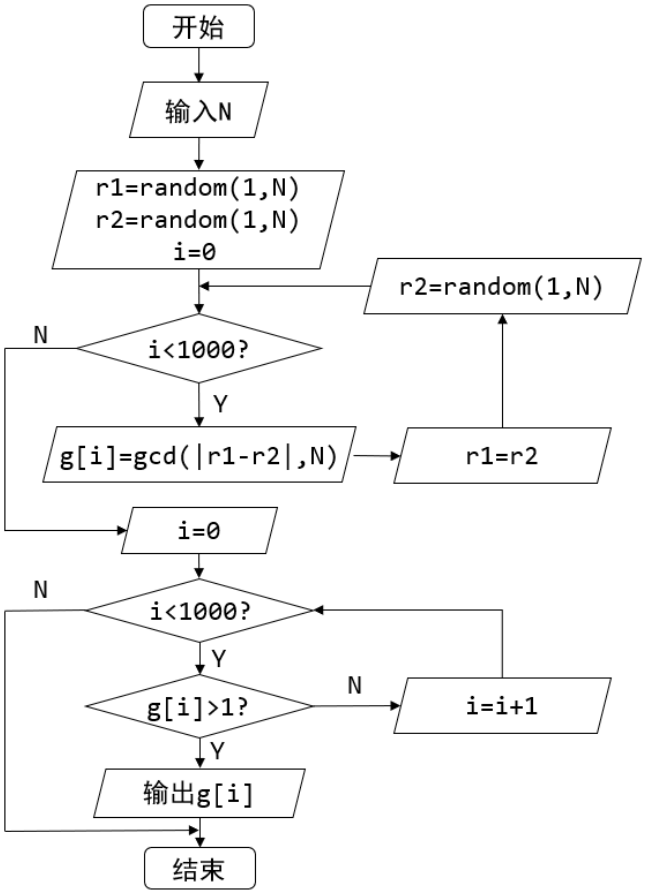

图 2: 串行实现的算法流程图

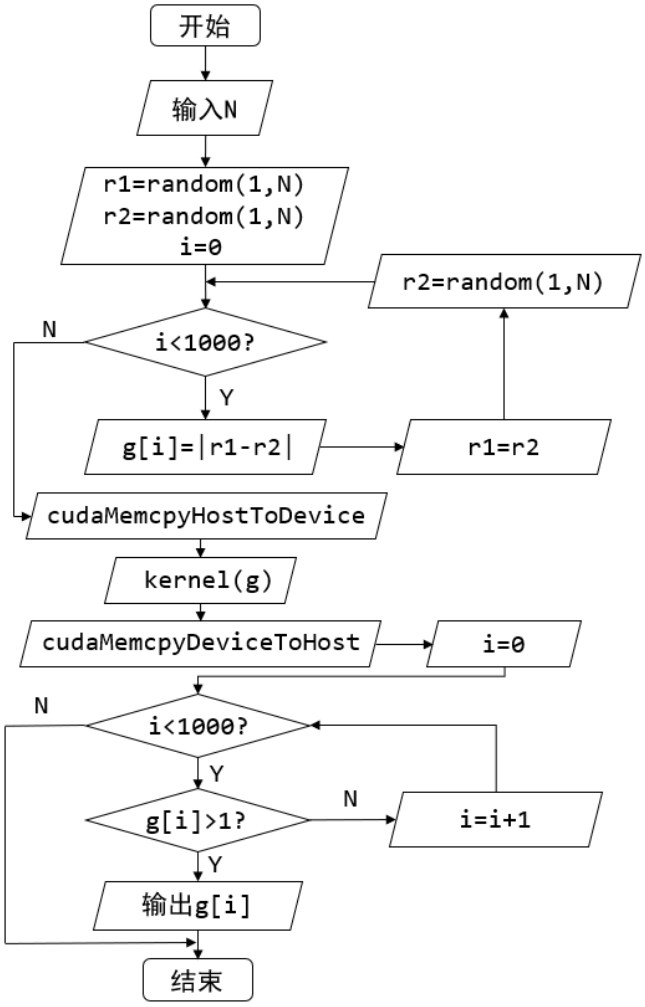

图 3: 并行实现的算法流程图

g[i]。在主机端，首先生成 k 个差值并存入数组 g 后，将数组 g 从主机内存上拷贝到显存上，然后启动核函数并等待核函数运行完成，再将计算结果从显存拷贝回主机内存中，最后检查计算结果，停止算法或开始下一轮计算，并行实现的算法流程图如图 3 所示。

4 结果与讨论

由于 Pollard-rho 算法是随机算法，需要用到随机数，为了使串行实现和并行实现的计算效率能够相互比较，在同一次实验中将串行实现和并行实现的随机数种子设为相同，所有实验数据都基于以下平台得出：

- CPU：Intel(R) Core(TM) i9-9900K
- GPU：NVIDIA GeForce RTX 2080 Ti
- CUDA 版本：11.1
- 操作系统：Windows 10 Professional 1909

实验选取了若干个的整数进行分解，这些整数都是由两个质数 p 和 q 相乘得到，在串行实现中禁用了代码优化，在并行实现中，调用了 4096 个线程块，每个线程块调用 1024 个线程。

实验中，对于每个要分解的整数，选取三个不同的数作为算法的随机数种子，并分别记录串行实现和并行实现求出 p 或 q 所花费的时间，单位为秒，以及它们的比值，实验结果如表 2 所示，绘制出对应的 GPU-CPU 时间折线图如图 4 所示。从实验结果中可以看出，当串行实现的运行时间比较短时，并行实现的运行时

N	p	q	seed	CPU	GPU	$\frac{CPU}{GPU}$
			303313	6.062	0.609	9.954
27674379981892801	841697141	32879261	20201205	19.671	1.547	12.716
			202117179110	7.546	0.703	10.734
			303313	24.563	1.781	13.792
1053051529365810619	841001333	1252140143	20201205	34.437	2.406	14.313
			202117179110	87.141	5.860	14.870
			303313	38.375	2.703	14.197
396690236970105433	763108249	519834817	20201205	24.016	1.750	13.723
			202117179110	107.281	7.172	14.958
			303313	39.672	2.875	13.799
297660826718818309	438468101	678865409	20201205	151.812	10.219	14.856
			202117179110	4.750	0.500	9.5
			303313	1.766	0.313	5.64
15922938483806544299	3309250267	4811645297	20201205	843.703	56.562	14.916
			202117179110	113.328	7.890	14.36
			303313	19.125	1.563	12.236
3024886511795737	63575821	47579197	20200512	5.813	0.625	9.301
			202117179110	1.485	0.266	5.583
			303313	4.407	0.508	8.675
2474508161911391	52478707	47152613	20200512	7.282	0.719	10.128
			202117179110	4.344	0.500	8.688

表 2: 串行实现和并行实现对不同的 N 进行分解所需时间 (单位：秒) 和比值

间优势并不是很大，但随着计算时间的增加，并行实现的加速效果越来越明显，当最高加速比可以接近 15。

5 总结与展望

Pollard-rho 算法是由 John Pollard 于 1975 年提出的用于寻找大整数的因子的算法，该算法基于生日悖论的思想提高找到因子的概率，算法的时间复杂度为 $O(n^{\frac{1}{4}})$。该算法属于随机算法中的拉斯维加斯算法，对于给定的问题，并不一定能找到问题的解，但如果找到一个解，则这个解必定是问题的正确解，随着算法运行时间的增加，算法找到问题的解的概率也越高。

本文详细介绍了 Pollard-rho 算法的核心思想和算法流程，并分别以串行和并行两种方式实现了该算法，选取了若干个由两个质数 p 和 q 相乘得到的 N 分别作为串行实现和并行实现的输入，记录从算法开始到结束的运行时间，得到了串行实现和并行实现的时间对比结果。从对比结果中可以看出，与串行实现相比，并行实现的运行时间明显减少，加速比大约在 13 至 14，最高可接近 15。由此可见 Pollard-rho 算法适合并行实现，但由于 C++ 语言的数据类型限制，输入的整数 N 最大只能支持 64 位，即 N 最大为 18446744073709551615。在主机环境中，可以使用第三方库如 GNU 高精度算术运算库 (The GNU Multiple Precision Arithmetic Library) 处理更大的整数，但设备环境中则无法调用主机环境的函数。在将来可以探索在并行实现中处理更大的整

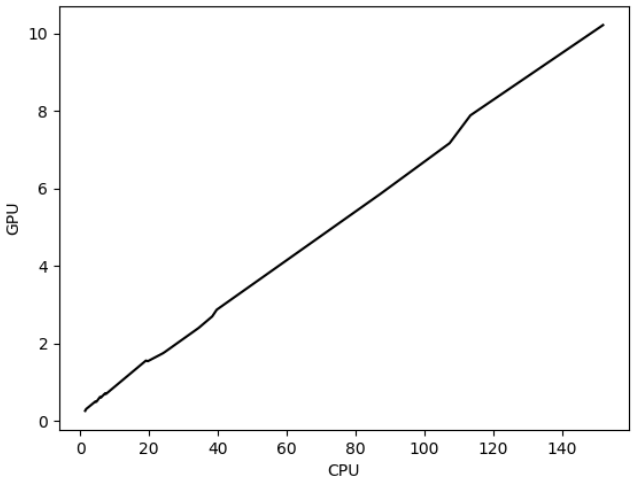

图 4: GPU-CPU 时间折线图

数的方法，也可以对当前的并行实现进一步优化，达到更高的加速比。

REFERENCES

[1] Mario Cortina Borja and John Haigh. The birthday problem. *Significance*, 4(3):124–127, 2007.

[2] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1969. exercises 6 and 7.

[3] J. M. Pollard. A monte carlo method for factorization. *Bit Numerical Mathematics*, 15(3):331–334, 1975.

[4] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Acm*, 21(2):120–126, 1978.