文章编号：2095-4980(2022)05-0492-06

可逆线性同余随机数发生器

卫丽华[1]，管致锦[*2]，朱鹏程[1]

(1.宿迁学院 信息与计算科学系，江苏 宿迁 223800；2.南通大学 信息科学技术学院，江苏 南通 226019)

摘　要：随机数的恢复是仿真系统实现时间回溯能力的关键问题之一，仿真系统通常包含大量随机数，为了恢复之前任意时刻的系统状态，就必须对随机数的状态演化进行跟踪和记录，这种跟踪和记录工作需要极大的时间和空间代价。为解决仿真系统中的随机数恢复问题，基于可逆计算的思想，提出一种可逆线性同余随机数发生器，并分别通过反函数、通项公式、逆分布函数实现该随机数发生器，使其既可正向生成任意分布的随机数列，又可反向恢复之前的随机数。实验结果表明该可逆随机数发生器在总的时空性能上较目前通用的Checkpointing实现方式优越。

关键词：可逆计算；线性同余法；随机数；仿真系统

中图分类号：TP301.6　　**文献标志码：**A　　**doi：**10.11805/TKYDA2020126

Reversible linear congruential generator

WEI Lihua[1]，GUAN Zhijin[*2]，ZHU Pengcheng[1]

(1.Department of Information and Computational Sciences，Suqian University，Suqian Jiangsu 223800，China；
2.College of Information Science and Technology，Nantong University，Nantong Jiangsu 226019，China)

Abstract：The recovery of random numbers is one of the key issues for the simulation system to realize the time traceability. The simulation system usually contains a large number of random numbers. In order to restore the system state at any time before, it is necessary to track and record the state evolution of the random number. This kind of tracking and recording work requires a great time and space cost. In order to solve the problem of random number recovery in the simulation system, based on the idea of reversible computation, a reversible linear congruential random number generator is proposed, and the random number generator is realized through the inverse function, general term formula, and inverse distribution function, respectively. It can not only generate random numbers in the forward direction, but also recover the previous random numbers in the reverse direction. The experimental results show that the reversible random number generator is superior to the current general checkpointing implementation in terms of overall space-time performance.

Keywords：reversible computation；linear congruence approach；random number；simulation system

大型仿真系统的每一场景都包含大量数据，其中包含很多随机数[1]。概率分布数列经常被计算机用来模拟物理系统模型，随机数发生器用于生成符合一定概率分布的数列。目前关于随机数发生器的主要研究有：1) 在一定精确度范围内，增加随机数的长度；2) 提高发生器的生成速度；3) 生成复杂分布数列；4) 降低多个数列间的相互影响。Jhessica Clawdia 等[2]使用线性同余发生器(Linear Congruential Generator，LCG)算法生成随机数。该算法可以避免生成重复随机数，增加随机数的长度。Mina Mishra 等[3]主要对 Matlab 随机数发生器(Random Number Generator，RNG)和线性同余发生器(LCG)随机数发生器算法进行了测试，测试结果表明 LCG 算法具有抗密码分析攻击的能力和良好的密钥敏感性。Massoud Sokouti 等[4] 提出了一种基于遗传算法思想的线性同余(LCG)方法，增加了生成随机数的随机性。D Apdilah 等[5]分别将线性同余发生器(LCG)和二次同余发生器(Quadratic Congruential Generator，QCG)与一次性密码本(One-Time Pad，OTP)算法组合用于生成随机数密钥，结果表明

收稿日期：2020-03-26；**修回日期：**2020-09-10
基金项目：宿迁市科技资助项目(S201819)
*通信作者：管致锦　email:116451890@qq.com

OTP 与 LCG 的组合，算法更快。

Bertrand Teguia[6]提出了一种基于素数分布的算法随机整数发生器，是一种新型分布的随机数发生器。然而，关于随机数发生器的可逆方面很少被研究或考虑。在仿真系统运行时，若要回溯至之前某一时刻的场景，就必须恢复和该场景相关的所有数据。目前较通用的做法是使用 Checkpointing 技术[7−8]预先保存相关数据，但这种方法会使得系统内存消耗和访存操作随着数据量的增大而急剧上升。Permulla[9]在粒子碰撞的仿真实验中首次使用可逆计算恢复碰撞前的粒子状态，实验结果表明该方法在数据密集且内存紧张的场景下在时间性能和空间性能上均优于 Checkpointing 技术。如何可逆地恢复随机数是将可逆计算应用于仿真系统的关键问题之一，仿真系统中涉及的基于各种概率分布的随机数均以均匀分布随机数为基础，本文拟设计一种可逆线性同余随机数发生器 (Reversible Linear Congruential Generator，RLCG)，该随机数发生器对其生成的随机数序列既可正向遍历，又可反向遍历，且反向遍历无需额外内存开销。

1　线性同余随机数发生器

线性同余随机数发生器利用数论中的同余运算产生随机数，其递推公式为：

$$\begin{cases} x_n = \left(ax_{n-1} + c\right) \bmod M \\ r_n = x_n / M, n = 1, 2, \cdots, \end{cases} \tag{1}$$

式中：a 为乘子，$0<a<M$；c 为增量，$0\leqslant c<M$；M 为模数，$M>0$，通常取 2 的整数次幂；x_0 为种子，$0\leqslant x_0<M$；为满足较好的统计性质，a 和 M 以及 c 和 M 均互质。通常用表达式 LCG(M,a,c,x_0)表示上述随机数发生器，该随机数发生器的周期和统计性质由 M、a 和 c 这 3 个参数所决定，在这方面已经有很多研究[9]。本文将重点关注如何在不影响统计性质的基础上实现可逆线性同余随机数发生器，并用 LCG(M,a,c,x_0)表示可逆发生器。

2　可逆线性同余随机数发生器（RLCG）

目前各类编程语言函数库中提供的 LCG 实现均只有正向功能，即只能根据当前随机数生成下一个随机数，而不能恢复之前的随机数，因此这些 LCG 实现不能应用于可逆环境中。以下将提供两种 RLCG 的实现，分别命名为 RLCG1 和 RLCG2。

2.1　RLCG1 的实现

实现 RLCG 的关键是找到函数 $f(x):x_n=\left(ax_{n-1}+c\right) \bmod M$ 的逆函数 $f^{-1}(x)$。为找到该反函数，首先列出几个有关线性同余方法的定义和定理。

定义 1　若 $M|a-b$，即 $a-b=kM$，则称 a 模 M 同余 b，记作 $a\equiv b(\bmod M)$。a,b 为整数，M 为正整数。

定理 1　令 M 为正整数，若 $a\equiv b(\bmod M)$，$c\equiv d(\bmod M)$，则 $a+c\equiv b+d(\bmod M)$ 且 $ac\equiv bd(\bmod M)$。

定理 2　若 a 和 M 是互质的整数，则存在整数 s，使得 $sa\equiv 1(\bmod M)$，将 s 称为 a 对模 M 的逆，记作 $\bar{a}$。

根据以上定理，可以推导出**引理 1**。

引理 1　若 a 和 M 互质，函数 $f(x):x_n=\left(ax_{n-1}+c\right) \bmod M$ 的反函数 $f^{-1}(\mathrm{x})$ 为：$x_{n-1}=b\left(x_n-c\right) \bmod M$，且 $b=\bar{a} \bmod M$，其中 $\bar{a}$ 为 a 对模 M 的逆。

证明：

$$\begin{aligned} & x_n=\left(ax_{n-1}+c\right) \bmod M \underset{\text{定义1}}{\Rightarrow} x_n \equiv ax_{n-1}+c(\bmod M) \\ \underset{\text{定理1}}{\Rightarrow} & x_n-c \equiv ax_{n-1}(\bmod M) \underset{\text{定理1}}{\Rightarrow} \bar{a}(x_n-c) \equiv \bar{a}ax_{n-1}(\bmod M) \\ \underset{\text{定理2}}{\Rightarrow} & \bar{a}(x_n-c) \equiv x_{n-1}(\bmod M) \underset{\text{定理1}}{\Rightarrow} b(x_n-c) \equiv x_{n-1}(\bmod M) \\ & \underset{\text{定义1}}{\Rightarrow} x_{n-1}=b(x_n-c) \bmod M \end{aligned}$$

可以使用扩展欧几里得算法求解上式中的 $\bar{a}$。RLCG1 的 Java 实现如程序 1 所示。程序 1 中 next() 函数生成正向随机数，和传统 LCG 算法完全一样；previous() 函数用来反向恢复随机数，其中的表达式和引理 1 中的反函数公式略有出入，这是由于 Java 中允许取余结果为负，而 LCG 中要求余

程序1 RLCG1的Java实现关键代码

```
public class RLCG1 {
    //正向生成随机数
    public long next(){
        x=(a*x+c)%M;
        return x;}
    //反向恢复随机数
    public long previous(){
        x=((b*(x−c))%M+M)% M;
        return x;}
    //扩展欧几里得算法求a的逆模
    public long exGCD (long a, long m){
        if (m==0){
            s=1; t=0;
            return s;}
        long t=exGCD (m, a%m), temp=s;
        s=t; t=temp−a/m*t;
        return s;}
    }
```

数非负；exGCD()函数通过递归的方式实现扩展欧几里得算法以求解$\bar{a}$。

生成长度为n的随机数序列其时间开销为$O(n)$，空间开销为$O(1)$。从程序1可见，next()函数和previous()函数的复杂度相同，因此在RLCG1中反向恢复随机数和正向生成随机数具备相同的时空开销，其时空复杂度曲线如图1所示。

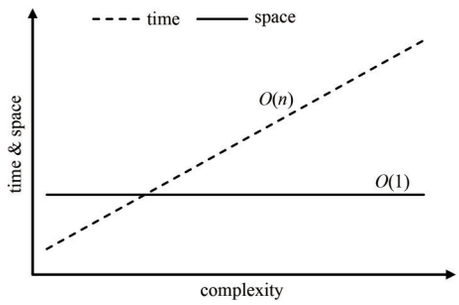

Fig.1 Time-space complexity curve for RLCG1 recovering random numbers

图1 RLCG1恢复随机数的时空复杂度曲线

2.2 RLCG2的实现

RLCG1只能根据当前随机数依次恢复前驱随机数，即如要得到位于当前随机数前n个位置的随机数，须连续调用n次previous()函数，为实现跳跃式恢复前面任一位置的随机数须考虑另一种方法，如找到随机数序列中任一随机数的通项公式。

定理3 若a,b以及M均为正整数，则等式$(a*b)\bmod M=\left((a\bmod M)*(b\bmod M)\right)\bmod M$和$(a+b)\bmod M=(a\bmod M+b\bmod M)\bmod M$恒成立。

引理2 已知$x_n=\left(ax_{n-1}+c\right)\bmod M$，则可推出$x_n=\left(a^n x_0+\left(1+a+\cdots+a^{n-1}\right)c\right)\bmod M,\ n=1,2,\cdots$。其中，$a,b$及$M$均为正整数。

证明

$$x_n=\left(ax_{n-1}+c\right)\bmod M\Rightarrow \text{存在整数}k,\ \text{使得}x_n+k_nM=ax_{n-1}+c,\ \text{且}0\leqslant x_n<M\Rightarrow \text{存在整数}k,$$

$$\text{使得}x_n-ax_{n-1}=c-k_nM,n=1,2,\cdots\Rightarrow\begin{cases}x_1-ax_0=c-k_1M & (1)\\ x_2-ax_1=c-k_2M & (2)\\ \vdots & \vdots\\ x_{n-1}-ax_{n-2}=c-k_{n-1}M & (n-1)\\ x_n-ax_{n-1}=c-k_nM & (n)\end{cases}\text{将}(1)\times a^{n-1},(2)\times a^{n-2},(3)\times a^{n-3},\cdots,(n-1)\times a\text{，：}$$

$$\Rightarrow\begin{cases}a^{n-1}x_1-a^nx_0=a^{n-1}c-a^{n-1}k_1M & (1)\\ a^{n-2}x_2-a^{n-1}x_1=a^{n-2}c-a^{n-2}k_2M & (2)\\ \vdots & \vdots\\ ax_{n-1}-a^2x_{n-2}=ac-ak_{n-1}M & (n-1)\\ x_n-ax_{n-1}=c-k_nM & (n)\end{cases}\text{，将}(1)+(2)+\cdots+(n-1)+(n)\text{得：}$$

$$\Rightarrow x_n-a^nx_0=\left(1+a+\cdots+a^{n-1}\right)c-\left(k_n+ak_{n-1}+\cdots+a^{n-1}k_1\right)M\Rightarrow x_n+\left(k_n+ak_{n-1}+\cdots+a^{n-1}k_1\right)M=a^nx_0+\left(1+a+\cdots+a^{n-1}\right)c$$

等式两边同时对M作取余运算：$\overset{0\leqslant x_n<M}{\Rightarrow}\ x_n=\left(a^nx_0+\left(1+a+\cdots+a^{n-1}\right)c\right)\bmod M$

显然，引理2所提供的公式仅需一个位置信息n就能够计算出随机数序列中的任何一个数。引理2为跳跃式恢复之前任一位置的随机数提供了可能，但由于LCG公式中的a一般为非常大的数，当n也较大时直接计算该式很容易生成溢出，从而导致出现错误的结果。根据定理3上述公式可以归结为当a和n较大时如何求的问题，使用同余幂算法可以很好地解决该问题。RLCG2的JAVA实现如程序2所示。

如程序2所示，next()用以正向生成随机数，除了每次调用都要更新位置n外与RLCG1的next()函数相同；previous(i)函数用以恢复领先当前i个位置的随机数；calNth(n)根据引理2的公式计算第n个随机数；modExp()函数是同余幂算法的实现，用以当a和n较大时求$a^n\bmod M$的问题。RLCG2恢复随机数的时空复杂度如图2所示，可见虽然RLCG2可以跳跃式地恢复之前的随机数，但其时间性能却比RLCG1采用依次恢复的方式差，尤其当待恢复的随机数在序列中位置靠后时，这种差异将越发明显。

程序2 RLCG2的Java实现关键代码

```
public class RLCG2 {
 //正向生成随机数
 public long next(){
        n=n+1; x=(a*x+c)%M;
        return x;  }
 //恢复领先当前i个位置的随机数
 public long previous (long i){
        n=n-i; x=calNth (n);
        return x;  }
 //计算第n个随机数
 private long calNth (long n){
        long mul=1;
        for (long i=1; i<n; i++){
              mul=(mul+modExp (a, i, M))% M;
        }
        x=(modExp (a, n, M)*(x0 % M))% M;
        x=(x+(mul*c)% M)% M;
        return x;  }
 //同余幂算法
 private long modExp (long base, long exp, long M){
  …     }
}
```

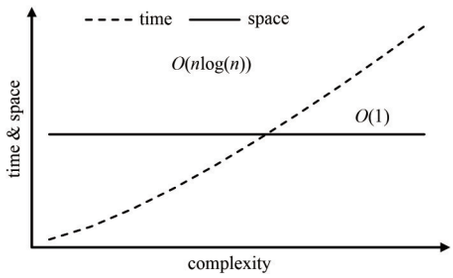

Fig.2 Time−space complexity curve for RLCG2 recovering random numbers
图2 RLCG2恢复随机数的时空复杂度曲线

3　基于CDF的可逆随机数发生器

为适应各种仿真环境对随机数不同分布的需求，利用累计分布函数(Cumulative Distributions Function，CDF)在闭区间内是可逆的特性，可以较容易地将RLCG生成的均匀分布的随机数列映射到所需的分布。给定PDF函数$p()$对应的CDF函数$c_p()$，可将分布采样为$x_r=c_p^{-1}(r)$，其中r是RLCG生成的均匀分布随机数，且$r\in[0,1]$。下面以生成指数分布的可逆随机数为例。假设给定指数分布概率密度函数(Probability Density Function，PDF)如下所示：

$$p(x)=\begin{cases}\lambda e^{-\lambda x}, & x\geqslant 0\\ 0, & x<0\end{cases}$$

得到累计分布函数CDF：$c_p(\mathrm{x})=\begin{cases}1-e^{-\lambda x}, & x\geqslant 0\\ 0, & x<0\end{cases}$

求得逆累计分布函数CDF：$c_p^{-1}(r)=x_r=\begin{cases}\dfrac{-\log(1-r)}{\lambda}=\dfrac{-\log r}{\lambda}, & r\in[0,1]\\ 0\end{cases}$

生成任意分布的可逆随机数生成算法如表1所示。其中$R_u()$，$R_u^{-1}()$为RLCG正反函数，$c_p^{-1}(r)$为累计分布函数CDF的反函数。

表1 基于CDF的可逆随机数发生器

Table 1 CDF-based reversible random number generator

positive	reverse
$R_c()$:	$R_c^{-1}()$:
$r\leftarrow -R_u()$	$r\leftarrow -R_u^{-1}()$
$x_r\leftarrow -c_p^{-1}(r)$	$x_r\leftarrow -c_p^{-1}(r)$
return x_r	return x_r

4　RLCG的验证和比较

以LCG(232, 22 695 477, 1, 0)为例，对RLCG1和RLCG2的可逆性以及时空性能进行验证，实验平台的主要配置如下：Intel i7-4710HQ CPU 2.5 GHZ，8G RAM。

4.1 可逆性的验证

通过RLCG1和RLCG2的*next*()函数分别生成50个随机数，可发现生成的随机数序列相同，如图3(a)所示。然后，通过两者的*previous*()函数恢复之前的随机数，恢复的随机数序列同样相同，如图3(b)所示。可发现图3和图4严格对称，说明RLCG1和RLCG2都正确地实现了可逆生成和恢复随机数的目的。

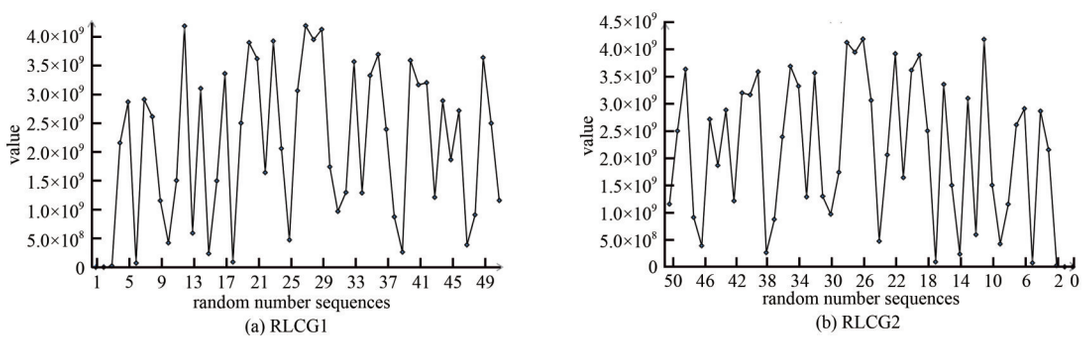

Fig.3 RLCG1 and RLCG2 generating random number sequences
图3 RLCG1和RLCG2生成随机数序列

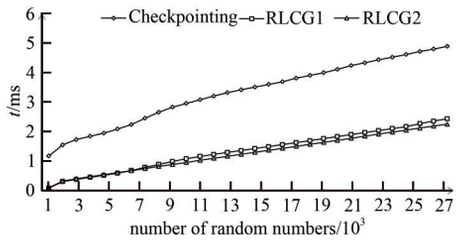

Fig.4 Forward time performance comparison
图4 正向时间性能比较

4.2 性能的验证

为验证RLCG1和RLCG2的时空性能，将其与使用Checkpointing技术实现的RLCG算法进行时空性能的比较，分别使用RLCG1、RLCG2和Checkpointing的*next*()函数依次生成由30 000个随机数组成的随机数序列，以序列中位置是1 000整数倍的随机数时间为采样点，其时间性能比较如图4所示，可发现正向生成随机数时RLCG1和RLCG2在时间消耗上优于Checkpointing，这是由于Checkpointing在每次生成随机数时需要额外的时间将该随机数存入堆栈以便后续恢复。

分别使用RLCG1、RLCG2和Checkpointing的previous()函数依次恢复由*next*()函数生成的30×10^3个随机数组成的序列，其时间性能比较如图5所示，可见Checkpointing略优于RLCG1，而RLCG2在恢复序列中位置为29×10^3~30×10^3的10^3个随机数时耗时较多，为2.97 s，是其他采样点时间的10^3倍，这和RLCG2的算法分析相符，即位置在序列中越靠后其耗时就越长，但当恢复的数据位置逐渐靠前时开始优于Checkpoingting，并逐渐接近RLCG1。

至于RLCG1、RLCG2和Checkpointing对内存的消耗，从3种不同实现的代码即可分析出，RLCG1仅需一个变量x保存当前随机数；RLCG2不仅需要变量x保存当前随机数，还需要一个变量n存储当前随机数在序列中的位置；前两者对内存的消耗是常量级的，而Checkpointing需要保存每一个生成的随机数，因此对内存的消耗随着随机数序列长度而线性增长，如图6所示。

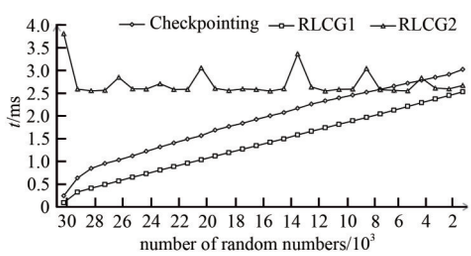

Fig.5 Reverse time performance comparison
图5 反向时间性能比较

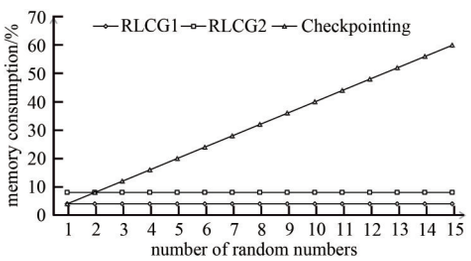

Fig.6 Space performance comparison
图6 空间性能比较

5 结论

从实验结果可知，RLCG1在生成随机数时其时间性能优于Checkpointing，在恢复随机数时其时间性能接近Checkpointing，且在内存开销上随着随机数序列的增长明显优于Checkpointing，同时通过累计分布函数的反函数，将生成的均匀随机数转换为任意分布随机数，且保持可逆性。因此在数据密集型仿真环境中恢复系统场景时可使用本文提出的可逆随机发生器恢复相应的随机参数，满足不同场景需求，具有较高的实践指导意义。

参考文献：

[1] 苏耀鑫,高秀峰. 基于矩阵的无线传感器网络SNEP改进[J]. 太赫兹科学与电子信息学报, 2018,16(6):1072-1079. (SU Yaoxin, GAO Xiufeng. Research on improvement of SNEP protocol based on matrix[J]. Journal of Terahertz Science and Electronic Information Technology, 2018,16(6):1072-1079.)

[2] CLAWDIA J,KHAIRINA N,HARAHAP M K. Implementasi algoritma kriptografi One Time Pad(OTP) dengan dyamic key Linear Congruential Generator(LCG)[J]. KOMIK(Konferensi Nasional Teknologi Informasi dan Komputer), 2017,1(1):12-14.

[3] MISHRA M,MANKAR V. Text encryption algorithms based on pseudo-random number generator[J]. International Journal of Computer Applications, 2015,111(2):1-6.

[4] SOKOUTI M,SOKOUTI B,PASHAZADEH S,et al. Genetic-based random key generator(GRKG):a new method for generating more-random keys for one-time pad cryptosystem[J]. Neural Computing and Applications, 2013:1667-1675.

[5] APDILAH D, HARAHAP M K, KHAIRINA N, et al. A comparison of one time pad random key generation using linear congruential generator and quadratic congruential generator[J]. Journal of Physics:Conference Series, 2018,1007(1):1–6.

[6] TABUGUIA Bertrand Teguia. An algorithmic random-integer generator based on the distribution of prime numbers[J]. Research Journal of Mathematics and Computer Science, 2019:1–9.

[7] MANSOURI Houssem, PATHAN Al Sakib Khan. Checkpointing distributed computing systems: an optimisation approach[J]. International Journal of High Performance Computing and Networking, 2019,15(3/4):202–209.

[8] BENKAOUHA Haroun, BADACHE Nadjib, ABDELLI Abdelkrim, et al. A novel hybrid protocol of checkpointing and rollback recovery for flat MANETs[J]. International Journal of Autonomous and Adaptive Communications Systems, 2017,10(1):114–138.

[9] PERUMALLA Kalyan S, PROTOPOPESCU Vladimir A. Reversible simulations of elastic collisions[J]. ACM Transactions on Modeling and Computer Simulation, 2013,23(2):1–25.

作者简介：

卫丽华(1984–)，女，江苏省南通市人，硕士，讲师，主要研究方向为可逆计算、软件测试.email: angelirene@163.com.

朱鹏程(1982–)，男，江苏省泰兴市人，硕士，讲师，主要研究方向为可逆计算、可逆编程语言.

管致锦(1962–)，男，江苏省连云港市人，博士，教授，博导，主要研究领域为可逆计算逻辑综合.