杭州电子科技大学

硕 士 学 位 论 文

题　　目：基于 GPGPU 计算的数字图像加密技术研究

研 究 生　温万里

专　　业　通信与信息系统

指导教师　游 林　教授

完成日期　2013 年 12 月

杭州电子科技大学硕士学位论文

基于 GPGPU 计算的数字图像加密技术研究

研 究 生： 温万里

指导教师： 游 林 教授

2013 年 12 月

Dissertation Submitted to Hangzhou Dianzi University

for the Degree of Master

Digital Image Encryption Based on GPGPU

Candidate: **Wen Wanli**

Supervisor: **Prof. You Lin**

December，2013

杭州电子科技大学
学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明： 所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。申请学位论文与资料若有不实之处，本人承担一切相关责任。

论文作者签名： 日期： 年 月 日

学位论文使用授权说明

本人完全了解杭州电子科技大学关于保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属杭州电子科技大学。本人保证毕业离校后，发表论文或使用论文工作成果时署名单位仍然为杭州电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。（保密论文在解密后遵守此规定）

论文作者签名： 日期： 年 月 日

指导教师签名： 日期： 年 月 日

摘 要

随着互联网技术和数字多媒体技术的快速发展，如何在网络中高效、安全的传输数字图像引起了人们的广泛关注。由于数字图像具有数据量大、冗余性高、图像像素相关性强等内在特性，使得传统的加密算法（如 DES 和 AES 等）并不能满足图像传输的实时性要求。混沌是一种确定性的非线性系统，它具有类随机性、遍历性和对初始状态的高度敏感性等良好特点，已被广泛运用于数字图像加密。随着混沌的引入，数字图像加密技术得到了空前的发展，各种各样的图像加密算法层出不穷。这些算法的出现极大的丰富了图像加密技术的内容，在保护数字多媒体安全方面体现了极大的应用价值。但是，当前的绝大多数图像加密算法都是在 CPU 上开发的，也就是说，这些算法没有考虑到图像加密的可并行性，因而只能在 CPU 上串行执行。这样导致的直接后果就是虽然开发出来的图像加密算法远比一些传统加密算法要快，但是仍然不能满足人们在实时图像加密中对加密速度的要求，因而无法得到普遍应用和推广。对于实时图像传输，若加密算法的运行速度很慢，即使这些算法的安全性再高，也不具有实用价值。因此，如何设计一种高效、安全的图像加密算法是图像加密技术领域遇到的一个重要挑战。

通用图形处理器（GPGPU）作为一种新的大规模并行处理器，其表现出的强大的并行计算能力得到了人们的广泛应用。为了充分利用 GPGPU 强大的性能，提高图像加密算法的执行速度，本文研究了基于 GPGPU 计算的数字图像加密技术，提出了两种基于开放计算语言（OpenCL）的并行图像加密算法，即基于混沌—自逆矩阵的图像加密算法（CSIM）和基于混沌—比特级置乱的图像加密算法（CBLP）。其中，CSIM 算法包含像素位置置换运算和像素矩阵乘法运算，位置置换用于置乱（Confusion）图像的像素位置，矩阵乘法用于扩散（Diffusion）图像的像素值；CBLP 算法包含像素位置置换、比特旋转和异或运算，位置置换用于置乱图像的像素位置，比特旋转和异或用于扩散图像的像素值。

为全面评估 CSIM 和 CBLP 算法的性能，本文分别从加密效率、密钥空间及敏感性、灰度直方图、信息熵、相邻像素相关性以及明文敏感性等几个方面对两种算法进行了仿真实验。实验结果表明 CSIM 和 CBLP 算法在加密性能上都是安全的，都可以有效抵抗穷举攻击、统计攻击、差分攻击以及已知/选择明文攻击等。而且 CSIM 和 CBLP 算法所具有的高度并行性使得它们非常适合使用 GPGPU 进行加速，所以两种算法均能达到很高的加密效率。因此，本文所提出的两个图像加密方案都是高效、安全的加密系统，都可以有效地应用于实际加密当中，具有较好的应用价值。

关键词：数字图像加密，混沌密码学，GPGPU 计算，OpenCL，安全性分析.

ABSTRACT

With the rapid development of Internet and digital multimedia technology, the transmission of the confidential digital image through network has attracted much attention. Due to the intrinsic properties of images such as bulky data capacity, high redundancy and strong correlation among image pixels, the classical encryption algorithms, such as DES and AES, can hardly meet the requirements of real-time image transmission. As a deterministic nonlinear phenomenon, the Chaos which has many characteristics, such as pseudo-randomness, ergodicity and highly sensitivity to the initial state, has been widely used in digital image encryption technology. Recent years, a variety of chaos-based image encryption algorithms have been reported, which greatly enrich the content of the image cryptography. However, the majority of the current image encryption algorithms do not take into account the parallelism of the image encryption, which means they can only run on the CPU in serial manner. As a result, the current image encryption algorithms can still not meet the requirements of speed and therefore they cannot be widely employed in real-time image transmission. For real-time image transmission, there is no any application value for an encryption algorithm with low speed. How to design an efficient and secure image encryption algorithm is a major challenge for the protection of image content.

As a new massively parallel processor, the General-Purpose GPU (GPGPU) has been applied to various fields for its powerful parallel computing capability. In order to take full advantage of the power of GPGPU, the digital image encryption technology solutions based on GPGPU computing is researched in this paper. And two OpenCL-based parallel image encryption algorithm, i.e., CSIM and CBLP encryption algorithm are proposed. The CSIM algorithm consists of the pixel position permutation operation and matrix multiplication operation, and the image will be confused by the pixel position permutation and diffused by the matrix multiplication operation. The CBLP algorithm includes the pixel position permutation operation, bit rotation and XOR operation, and therefore the image is confused by the pixel position permutation and diffused by the bit rotation and XOR operation.

Some experiments have been made to verify the security and efficiency of the CSIM and CBLP algorithm from the aspects of encryption efficiency, key space and sensitivity, histogram, information entropy, the correlation of adjacent pixels and plaintext sensitivity. Simulation results and numerical analysis show that both CSIM and CBLP algorithm are secure enough to resist some kinds of attacks, such as brute-force attacks, statistical attacks, differential attacks and known/ chosen-plaintext attack. Moreover, that the high parallelism of our algorithms makes CBLP and

CSIM are suitable to be accelerated by GPGPU, which as a result, the two algorithms can achieve high efficiency in encryption/decryption process. Therefore, both the two image encryption schemes proposed in this paper are efficient and secure enough to be applied to the practical application.

Keywords: Digital Image Encryption, Chaos Cryptography, GPGPU Computing, OpenCL, Security Analysis.

目　录

第 1 章 绪论

随着互联网技术和数字多媒体技术的不断进步，数字多媒体数据（如数字图像、数字视频和音频等）的应用越来越广泛。作为数字多媒体的一个重要组成部分，数字图像占据了互联网内容和应用的很大一部分，已成为计算机相关技术的一个重要方面。同时，数字图像和其他数字多媒体，特别是数字视频，有着很多相似之处，对于数字图像的安全性研究可以直接对其他数字多媒体安全的研究起到借鉴和指导的作用[1]。因此，本文将研究多媒体数据安全中有关图像加密的问题。

本章主要内容安排如下：(1) 课题的研究背景与意义；(2) 数字图像加密技术研究现状；(3) 通用 GPU 计算技术研究现状；(4) 本文主要研究内容与安排。

1.1 课题研究背景与意义

自上世纪末以来，随着互联网技术和各种数字多媒体技术的产生和发展，网络中传输的多媒体数据给人们的工作、学习和生活带来了革命性的影响。然而网络技术的使用，为人们带来了无尽的便捷和好处的同时，越来越多的数据安全问题随之暴露了出来。网络中传输的数据不仅仅只有文本数据，还有音频、视频和图像等其他数字多媒体数据。这其中，数字图像作为数字多媒体的重要组成部分已经被广泛地应用到了人们的日常生活中，因此如何保证它的安全性就显得尤为重要了。

在众多的安全技术与手段中，数据加密技术显然是最基本、最重要也是最常用的方法。但是由于图像数据本身的特点，如数据量大、相关性强等特性，使得传统的基于文本的加密方法（如 AES 何 DES 等）难以适应对图像的实时加密，这主要表现在以下两个方面[1~9]：

(1) 相对文本来说，图像的尺寸一般比较大，例如一幅 1024×1024 大小的无压缩灰度图片就有 1MB，使用传统的加密算法对图像进行加密需要较长的时间，难以和实时图像传输的速率相匹配。

(2) 图像中各像素与其相邻像素之间往往存在着很强的相关性（通常能达到 95%以上），并且这种相关性在相邻的 8 到 16 个像素之间都是存在的。采用传统的加密算法，只对小区域的数据进行加密是不可取的，需要对大区域（最好是整幅图像）的数据进行快速加密才行[7]。

因此，当前学术界正研究将一些新颖的方法引入密码学领域，如将混沌引入到密码学领域是当今信息科学和非线性科学相结合的重点课题之一，基于混沌的密码学业已成为密码学领域的一个重要研究分支。正因为传统的加密算法并不适合图像加密，混沌加密技术的出现，为数字图像加密技术的发展提供了新的思路，开辟了新的研究方向。随着混沌的引入，数字图像加密技术得到了空前的发展，各种各样的加密算法层出不穷[10-36,48-50,58-59]。这些算法的出现极大的丰富了图像加密技术的内容，在保护数字多媒体安全方面体现了极大的应用价值。

但是，当前的绝大多数图像加密算法都是基于 CPU 开发的，也就是说，这些算法没有考虑到图像加密的可并行性，只能在 CPU 上串行执行。这样导致的直接后果就是虽然开发出来的图像加密算法远比一些传统基于文本的加密算法要快，但是仍然不能满足人们在实时图像加密中对加密速度的要求。对于实时图像处理，若加密算法的运行速度很慢，即使这些算法的安全性再高，也不具有实用价值。

对于速度的提升，传统的办法是升级 CPU，但是高性能的 CPU 不仅价格昂贵，而且对数值计算速度的提升效果并不明显，这将导致很多基于 CPU 开发的图像加密算法无法得到广泛的应用和推广。

作为一种高度并行化的处理器架构，图形处理器（Graphics Processing Unit，GPU）所具有的强大的并行能力促使了人们开始用它做通用计算任务以实现对通用数据的处理。人们对通用计算的强烈需求导致了通用 GPU（General-Purpose GPU，GPGPU）[6]计算技术的产生和发展。GPGPU 的出现使得运算速度的提升有了新的途径，GPGPU 价格较低并且非常适合并行计算，它能带来比 CPU 更高的计算速度和更低的硬件成本投入。在需要很高计算能力的密码学应用领域，以往只能通过设计特殊的硬件，如 FPGA 来加速密码算法，这无疑需要增加额外的开销来购买硬件[58]。然而，如果利用 GPGPU 的计算潜力则几乎不需要增加开销，这也是目前 GPGPU 计算快速普及并得到人们广泛关注的又一重要原因[46-47,49-50,58]。总之，如何设计高效、安全的图像加密算法是图像加密技术领域遇到的一个重要挑战。而 GPGPU 所特有的并行计算特性如果能应用到我们所设计的图像加密算法中，将会显著提高加密算法的运行速度，大幅降低实时图像加密的处理时间，极大地提高生产效率。

综上所述，本文将重点研究基于 GPGPU 计算和混沌的数字图像加密技术，设计和实现几种高效、安全的并行图像加密算法，并通过给出大量的仿真实验结果展示算法的安全性和有效性以及本文研究工作的正确性与合理性。同时，我们也希望本文的研究工作能为 GPGPU 技术在数字图像加密技术领域的应用起到一定的推进作用。

1.2 数字图像加密技术研究现状

混沌映射所具有的遍历性、类随机性以及对初始状态的敏感依赖性等基本特性和密码学的许多要求有着很重要的联系，使其特别适用于保密通信等领域[8-9,54]。混沌映射的特点吸引了密码学家的关注，提出了基于混沌映射的图像加密算法。与传统加密算法相比，基于混沌的图像加密算法主要有密钥空间大，实现简单和具有鲁棒性等优点，因此基于混沌的图像加密算法已成为密码学领域中一个重要的研究方向。

目前来说，对于数字图像的加密一般考虑使用基于像素位置置乱、基于像素值扩散或者两者结合的方法。

对于基于像素位置置乱的图像加密方法，国内外专家学者开展了很多细致而深入的研究工作，取得了丰硕的研究成果。目前已有的置乱方法包括有：基于 Arnold 变换（也称猫映射）

[10-12]、生命游戏[13]、骑士巡游变换[14-16]和希尔伯特变换[17]等置乱方案。

生命游戏是一种特殊的图像矩阵变换，Wang 等[13]研究了一种基于生命游戏的图像置乱算法。Ian Parberry[14]提出了骑士巡游变换，Delei[15]和 Thanikaiselvan[16]使用骑士巡游变换实现了图像的置乱，并将其用于信息隐藏中。

德国数学家 D.Hilbert 在 1891 年构造了能填满一个单位正方形的曲线，称为 Hilbert 曲线。林雪辉等[17]研究了基于 Hilbert 曲线的数字图像置乱方法，引入了“重合度”和“Hilbert 曲线的平移”等二个概念，利用 Hilbert 曲线平移获得了更多新的置乱路径，并根据重合度变化图探讨了置乱变换的规律。

Arnold 变换是周期性图像置乱算法的代表，但目前这些算法的性能和效率已经不能满足人们的实际需求，研究人员正在探索更加有效，更加可靠的图像置乱加密方案。尽管单纯的图像置乱方案的安全性不高，但基于像素位置置乱的加密方法可以应用在数字图像加密处理过程的预处理和后处理中，如先对明文图像进行置乱预处理，再进行其他的加密操作，如基于像素灰度值扩散的加密等，可以得到较为满意的效果，因此关于有效图像置乱方案的研究仍是有意义的。

基于像素灰度值扩散的加密较单纯的像素位置置乱加密更有意义，因为它改变了图像像素序列的实际内容。但是采用将二者结合的图像加密方法更受研究者青睐，Lian 等[19]提出了一种基于 Logistic 映射与修正标准（Modified Standard Map）映射的图像加密算法，该算法分别使用一维 Logistic 映射和修正标准映射来扩散图像的灰度值以及置乱图像的像素位置，同时他们指出至少要进行四轮置乱—扩散操作才能达到较好的加密效果。Wong 等[20]使用了一种简单的序列加移操作（Add-and-Shift）在置乱过程中也加入了扩散操作，在一定程度上提升了算法的安全性。而且该算法比 Lian 的算法所需要的加密轮数要少，所以加密效率也得到了进一步提升。2009 年，Wang 等[25]用 Tent 映射构造了一个带动态 S 盒的分组密码算法，该算法利用动态 S 盒扩散图像的像素值，用 Tent 映射置乱像素的位置，从而达到了良好的加密效果。

高维混沌映射通常具有多参数、高随机性等特点，具有比一维混沌映射更加复杂的动力学性态，基于高维混沌系统的加密算法更加安全可靠。Gao 等[26,28-29]通过使用一个置乱矩阵来置乱像素位置，然后使用一个高维混沌映射扩散像素值设计了一种新型的图像加密算法。Arroyo 等[30]在对 Gao 的算法进行分析后，指出 Gao 提出的算法存在未考虑混沌系统的时间步长、密钥空间较小以及加密效率不高等缺陷，而且算法不能有效抵抗选择明文攻击。许克兵等[31]给出了一种基于二维混沌映射的图像加密方法并指出使用高维的混沌映射即使减少加密的迭代次数也能达到很好的加密效果。Sun 等[32]提出了一种基于 Lorenz 混沌映射和 Arnold 变换的图像加密算法，该算法利用 Arnold 变换对图像进行置乱，然后利用 Lorenz 混沌映射扩散置乱后的像素值。Sun 的算法具有密钥敏感性和大密钥空间等优点，但是赵亮等[33]人经过分析后指出 Sun 的算法不能有效抵抗已知/选择明文攻击，为此他们首先分析了 Sun 的算法的安全缺陷，而后从安全性和效率上分别进行了改进并提出了相应的算法。为进一步改进 Sun

的算法，许冰等[34]提出了一种将位置置乱和像素替换加入到每次迭代中，并使加密数据流与明文信息相关的改进算法，分析和仿真实验表明，该算法修正了原 Sun 的算法的安全漏洞，减少了迭代次数，具有较高的加密效率。Wu 等[35]提出了一种基于二维 Logistic 映射的图像加密算法，该算法在 Fridrich 经典图像加密架构的基础上进行了改进，通过在图像扩散之后加入混沌矩阵转置操作，有效改善了图像的置乱程度。Zhang 等[36]提出了一种基于可逆二维混沌映射和依赖扩散（Dependent Diffusion）技术的图像加密算法，该算法同样改进了 Fridrich 加密架构，通过依赖扩散技术使得在图像的扩散过程中同时完成了图像置乱，实验证结果表明该算法具有比 Fridrich 算法更高的安全性和更快的加密速度。

1.3 通用 GPU 计算技术研究现状

在上世纪末，GPU 主要用在专业图形处理领域，但随着时间的推移，现代 GPU 已经不再局限于图形处理了。GPGPU 计算技术的发展已经引起业界广泛关注，事实也证明在浮点运算、并行计算等部分计算方面，GPU 可以提供数十倍乃至于上百倍于 CPU 的性能，例如 Intel 公司推出的顶级 Core i7 965 处理器，在默认情况下其浮点计算能力也只有 NVIDIA GeForce GTX 280 的 1/13。

作为一种异构协同计算模型[45]，GPGPU 通过把程序中可并行的部分交给 GPU 处理，把不可并行的或没有必要并行的部分仍交由 CPU 处理而大大提高了程序的执行速度。GPGPU 的发展历史可分为两个阶段。2007 年之前的 GPGPU 计算都必须依赖于具体的图形库（如 OpenGL 与 DirectX）来操纵 GPU，由此形成了各种图形开发语言，如 Stanford 大学的 RTSL[37] 和 Brook[38]，OPENGL ARB 的 GLSL[39]，Microsoft 公司的 HLSL[40]以及 NVIDIA 公司的 Cg[41] 等编程语言。2007 年之后，各大硬件厂商向用户提供了直接用于通用计算的编程接口，使得程序员无需再依赖于图形库就可以操纵 CPU 硬件。这些与 C 语言类似的编程框架大幅降低了用户使用 GPU 进行通用计算的门槛。

到目前为止，这种编程框架主要有 NVIDIA 公司的 CUDA[42]，Apple 公司的 OpenCL [43,45] 以及 Microsoft 公司提出的 DirectCompute[44]等。由于新一代 GPU 通用计算体系适应性强、功能全面而且性能优越，现已全面取代了基于图形库的通用计算[46]。开放计算语言（Open Computing Language，OpenCL）是 2008 年 12 月提出的面向 CPU、GPU 和其他处理器组合构成的计算机系统进行编程的第一个行业标准（2013 年最新版为 2.0），支持的 CPU 包括 x86、ARM 和 PowerPC，同时支持 AMD 和 NVIDIA 的显示卡。OpenCL 是一种统一的程序开发平台，各类开发设计者均能 在其上快速编写轻量级的应用程序，因而在游戏开发、数据加密以及数学科学等各种领域都有广阔的应用发展空间[47,57]。

在互联网高度发达的今天，人们对实时图像加密的需求越来越强烈。图像的实时加密也就意味着图像加密算法在安全性能得到保证的前提下，图像的加密/解密速度必须足够的快。用 GPGPU 计算来加速密码算法尤其是图像加密算法，国内外专家学者们已经开始做了一些

研究[48-50]。

Wang 等[48]提出了一个基于消息传递接口模型（Message Passing Interface，MPI）的分布式混沌图像加密算法，该算法通过使用多台双核计算机组成一个主从式并行计算网络并行化加密数字图像。但是 MPI 技术实现起来十分复杂，而且 Wang 在他们的算法实现中使用了多台计算机，因此用 MPI 来加速密码算法使得算法的实现成本过高，不利于算法的推广应用。为解决这样的问题，孙冰[49]在其论文中介绍了如何在 ATI GPGPU 上实现针对彩色数字图像加密的 AES 算法，成功解决了对高清图像的快速加解密。但是孙冰的算法是基于 AES 实现的，而 AES 并不特别适合图像的加密处理，如果我们能用 GPGPU 去加速专门针对图像设计的加密算法的话，则加密速度一定会更快，算法也将更加高效。为此，Vihari 等[50]提出了基于 NVIDIA CUDA 架构的并行图像加密算法，在该算法中图像的置乱和扩散部分均成功的使用 CUDA 进行了并行化。实验结果证实，Vihari 的算法能极大的提升图像加密的速度。

以上的研究工作要么是在 CPU 上加速图像加密算法，要么是在 GPGPU 上加速 AES 等传统密码算法，Vihari 的工作为 GPGPU 加速图像加密算法作了开拓性的工作，但是他使用的 GPGPU 架构是 CUDA，众所周知 CUDA 只能用在 NVIDIA GPU 上，对于其他的硬件平台则无能为力。然而从当前的硬件发展趋势来看，未来肯定是异构多核平台的天下[45]，这意味着我们所开发的算法必须保证能够在各种异构的平台上自由移植，并且能充分而合理的利用到整个异构平台的所有计算资源[52]。OpenCL 的出现让并行计算具有了更加灵活和统一的编程环境，这也使得 OpenCL 非常适合实现运算复杂的密码算法，因此研究基于 OpenCL 的图像加密算法具有一定的现实意义和实践价值。关于这方面的研究内容目前尚未见有报道，本文拟对此做一定程度的探索。

1.4 本文主要研究内容与安排

利用混沌系统的优良特性对图像进行加密是一种行之有效的加密方法，利用 GPGPU 大规模加速科学计算是当前并行计算的一个重要研究分支，OpenCL 作为未来在异构平台上进行 GPGPU 计算的行业标准框架更是吸引了越来越多的专家学者的广泛关注。本文将通过对基于混沌的图像加密技术以及基于 OpenCL 的 GPGPU 计算技术进行一定程度的研究，试图找出两种技术的契合点。通过将 OpenCL 技术与图像的混沌加密技术相结合，提出几种基于 OpenCL 的混沌图像加密算法，为 GPGPU 计算在数字图像加密技术领域的应用起到一定的推进作用。

全文共分 5 章，具体安排如下：

第一章为绪论，主要阐述了课题的研究背景和意义，简要介绍了数字图像加密技术和 GPGPU 计算的研究现状，并就混沌理论与 GPGPU 计算技术应用于图像加密技术的实际出发，提出本文要解决的问题然后指出后文工作所要解决的重点。

第二章简要介绍了混沌理论、混沌密码学以及 OpenCL 技术的基础知识，指出了数字图

像加密算法的几个设计原则以及使用 OpenCL 的意义。

第三章为本文的核心章节，研究了基于二维 Logistic 混沌映射和 OpenCL 的并行图像加密算法，提出了两个具有高度并行性的算法——CSIM 和 CBLP 算法，并分别给出了两种算法的 CPU 实现和 GPU 实现。

第四章为本文的核心章节，针对第三章提出的两个并行图像加密算法进行了大量的仿真实验和算法分析，指出了 CSIM 和 CBLP 算法均具有良好的安全性和较高的加密速度，可以有效的应用于实际加密当中。

第五章为全文总结。

第 2 章 混沌密码学与 OpenCL 技术基础

2.1 混沌理论

2.1.1 混沌定义

混沌理论是学者对自然界中许多不可解释与预期现象的一种诠释，对于复杂多变的万事万物，甚至于社会科学领域都是这一派学者所追求的主要课题。这些不同向度的系统都具有共同的特点，即在它们变幻无常的演绎背后所呈现的是某种无法理解的不确定规则。混沌论的学者就是试图去了解和掌握这些存在于秩序、复杂和混沌边缘的变幻原理。

到目前为止，对到底什么是混沌，尚无统一的定义，但最被广泛接受的是李天岩和 J.A. Yorke 给出的数学定义[5]，这是第一次给予混沌以严格的科学定义。

定义 2.1.1：（Li-Yorke 定义）[5]设在区间 I 上有一连续映射 $f(x)$，如果满足以下所有条件，可以认为该映射有混沌现象。

(1) f 的周期点的周期没有上界；

(2) I 上存在不可数的子集 S，满足下列条件：

1) 对 $\forall x, y \in S$，且 $x \neq y$ 时，有：

$$\lim_{n\to\infty} \sup | f^n(x) - f^n(y) | > 0 \tag{2.1}$$

2) 对 $\forall x, y \in S$ 时，满足：

$$\lim_{n\to\infty} \inf | f^n(x) - f^n(y) | = 0 \tag{2.2}$$

3) 对 $\forall x \in S$ 和 f 的任意周期点 y，且 $x \neq y$，有：

$$\lim_{n\to\infty} \sup | f^n(x) - f^n(y) | > 0 \tag{2.3}$$

定义 2.1.2：（Devaney 定义）[8-9]设有度量空间 V 和其上的映射 $f: V \to V$，若满足以下三个条件，称映射 f 在空间 V 上混沌：

(1) 对初始状态的敏感依赖性：$\forall \delta > 0$，对于 $\forall \varepsilon > 0$ 和 $\forall x \in V$，在 x 的 ε 邻域内存在 y 以及自然数 n，使得 $d(f^n(x), f^n(y)) > \delta$；

(2) 拓扑传递性：V 上任意开集 $Z, Y \in V$，$\forall k > 0$ 使 $f^k(Z) \cap Y \neq \varnothing$。

(3) f 的周期点集在 V 中稠密。

无论是 Li-Yorke 定义的混沌，还是 Devaney 定义的混沌，他们所反映的内在本质是一样的，都刻画了混沌的基本性质。简单地说，混沌是非线性动力系统中出现的一种确定性、貌似无规则的运动，这种运动即非周期又不收敛，且对初始值极其敏感。

2.1.2 混沌的度量与判定

混沌运动作为非线性的动力系统具有很多复杂的特性，本节将从研究描述混沌系统深刻物理内涵的特征量开始，以定量的角度来刻画混沌系统并从这些物理量上给出一些混沌的判据和准则[1~5]。

(1) Lyapunov 指数

Lyapunov 指数 λ 通常与系统的混沌程度有关，对于一维非线性系统来说，λ 反映了两个相邻的初值所产生的混沌轨道随着时间的推移的平均发散程度。若假设两个轨道初始相距 d_0，则经过 n 次迭代后两轨道相距为 $d_n = d_0 e^{\lambda n}$，其中 λ 为 Lyapunov 指数。显然，若 $\lambda < 0$，则系统处于稳定状态，即两轨道互相靠拢；若 $\lambda > 0$，则系统将处于不稳定状态，即处于混沌状态。

在一维动力系统 $x_{n+1} = F(x_n)$中， Lyapunov 指数可以写为：

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n-1} \ln | F'(x_i) | \tag{2.4}$$

对于连续时间系统的 Lyapunov 指数，设 $X = [x_1, x_2, \ldots, x_n]^T \in R^n$，其中为 n 维相空间上一个向量，设 $F = [f_1, f_2, \ldots, f_n]^T$ 是一个含有非线性函数的向量函数，$X_0 = [x_{10}, x_{20}, \ldots, x_{n0}]^T$ 为系统相空间中的一个起始点，系统可用下面的微分方程来表示：$X = f(X)$，设 $f'(x)$表示 f 的雅克比矩阵，即：

$$f'(x) = \frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{2.5}$$

在初值 X_0 处得一阶线性化近似值为：

$$\partial \frac{dX(t)}{dt} = \frac{d}{dt} \delta X(t) = J(X_0) \delta X \tag{2.6}$$

求解上式得解为：

$$\begin{aligned} \delta X &= [\delta x_1, \delta x_2, \cdots, \delta x_n]^T \\ &= [\delta x_{10}\, e^{\lambda_1 t}, \delta x_{20}\, e^{\lambda_2 t}, \cdots, \delta x_{n0}\, e^{\lambda_n t}]^T \end{aligned} \tag{2.7}$$

其中，λ_i 可通过下式计算出，即：

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\| \partial x_i \|}{\| \partial x_{i0} \|} \tag{2.8}$$

由公式式(2.7) ~ (2.8)可以看出，Lyapunov 指数 λ 反映了混沌系统中各变量在时间演变过程中伸缩变化的平均变化率。事实上不管是何种系统，若有 λ 小于 0，则系统对初始状态不敏感；若 λ 等于 0，则系统处于稳定边界；若 λ 大于 0，则系统将对初始状态极为敏感，此时系统处于混沌状态，当出现两个或两个以上 λ 大于 0 时，系统将处于超混沌状态[1~6, 55]。

(2) 测度熵

信息熵是信息论中最常用的参数，在信息论中熵被定义为[54]：

$$S = -K_B \sum_i P_i \ln P_i \tag{2.9}$$

其中 $K_B > 0$ 为常数，P_i 是系统处于第 i 个状态的概率，熵 S 是系统无序程度的度量，当 $S > 0$ 时系统总存在一些无法认识的方面。

由测度熵的取值可以判断系统运动的无规则运动程度，它代表了信息的平均丢失率。当测度熵等于 0 时，系统做确定性运动；当测度熵趋于无穷大时，系统做随机运动；当测度熵是大于 0 的常数时，系统做混沌运动，且熵越大系统的混沌程度越严重。在一维映射中，测度熵等于 Lyapunov 指数，而在多维映射中，测度熵是所有正的 Lyapunov 指数之和。从上述分析可知，测度熵是系统中信号不可预测性的一个量度，测度熵越大，则系统的随机性就越强[55]。

2.2 混沌密码学

2.2.1 混沌与密码学的关系

混沌作为一门新兴的学科，在金融学、气象学、生物学乃至国际关系学中的应用都极大地引起了专家学者的兴趣。将混沌学与密码学这一历史悠久的学科有机结合起来便催生出了崭新的混沌密码学。香农博士早在其经典文章[54]中就提出了密码学与混沌理论之间的天然联系，并给出了在密码算法设计中的两个通用规则：置乱（Confusion）与扩散（Diffusion）。置乱是为了掩盖明、密文和密钥之间的关系，尽量复杂化密文和密钥之间的统计关系，使破译者无法直接从密文推出密钥；而扩散则是为了将明文中的冗余度分散到密文中，以便于隐藏明文统计信息。图像加密就是通过置乱和扩散两个步骤实现的，即通过打乱像素的位置实现置乱目的，通过置换像素值达到扩散的目的。几乎所有的图像加密的流程都大同小异，图 3.1 给出了基于经典的 Fridrich 架构[18]的图像加密算法框图。

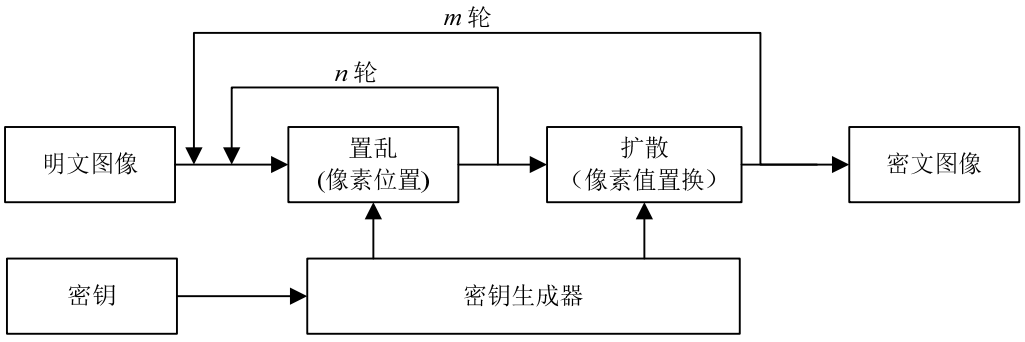


图 2.1 图像混沌加密系统结构

混沌系统的参数敏感依赖性和类随机性与现代密码体制的扩散性相对应，而混沌系统的轨道混合特性与初状态敏感性和轨道发散相关联，这和现代密码体制的扩散特性是对应的

[1,56]。

随着混沌理论的不断发展和完善，混沌密码学正得到越来越多的研究和重视，特别是应用混沌密码学对数字多媒体信息（如数字图像等）进行加密已成为新兴的研究热点。

2.2.2 数字图像混沌加密系统的设计原则

一般来说基于混沌理论的数字图像加密算法应关注混沌系统的动力学特性、算法的安全性、执行效率以及实现的难易程度等几个问题。对这些问题的疏漏很可能会使得所设计的算法产生安全纰漏。

(1) 混沌系统的退化

混沌系统会因为量化噪声的存在使得其表现的动力学特性出现一定程度上的退化。例如系统的 Lyapunov 指数、遍历性等都存在不稳定的特性。使用多个混沌系统相连或者使用高精度计算机，可以在某种程度上防止混沌系统的退化进而改善混沌系统的稳定性。

(2) 安全性

由于人们对图像混沌加密技术的深入研究，出现了非常多的分析方法，一丝的安全纰漏就能被攻击者利用。图像加密的最基本要求就是安全性，而一个图像加密算法的优劣可以从灰度直方图、信息熵、密钥空间及敏感性、相邻像素相关性以及明文敏感性等几个衡量图像加密算法性能的指标反映出来。一般来说，一个优秀的加密算法应该是：密文图像的自相关度和与明文图像的相似度要越小越好，密图的灰度平均变化值要接近灰度级一半，密图信息熵要大且不动点比要小。另外在设计算法时，还应确保算法能抵御各种潜在的攻击，如对密钥的穷举攻击以及差分和已知/选择明文攻击等。

由于混沌系统是一种确定性的非线性系统，攻击者总可以通过各种方法找出混沌系统的相关信息，所以为了提高混沌加密系统的安全性，在设计算法的时候应尽量使用复杂的混沌系统，这样可在一定程度上隐藏混沌系统的统计信息。

(3) 加密效率

加密算法不应给数字图像的实时传输和存取带来过大的时间开销，这就要求所设计的算法必须足够的快速，否则设计的算法就失去了实际意义。然而算法的执行速度和安全性往往是矛盾的，例如使用复杂的混沌系统虽然可以提高加密算法的安全性，却是以降低算法的执行速度为代价的，这样就没法满足图像的实时加密。然而算法的安全性相对于速度来说更加重要，所以在设计算法的时候在能保证安全性的前提下采取各种措施来提高算法的执行速度。

(4) 算法实现

图像混沌加密技术不单单用在平面图像加密上，更重要的是用在实时图像加密上，这就要求在设计算法的时候应尽量使得算法简单。只有安全、简单、效率高的算法才有实用价值，另外在选择硬件时也应做到简便实用。图像混沌加密技术不单单用在平面图像加密上，更重要的是用在实时图像加密上，这就要求在设计算法的时候应尽量使得算法简单。只有安全、简单、效率高的算法才有实用价值，另外在选择硬件时也应做到简便实用。

2.3 OpenCL 的产生背景

OpenCL 是一个由非盈利性技术联盟 Khronos Group 进行管理的异构编程架构[43,57]，最初由 Apple 公司开发并拥有其商标权，后在与 AMD、IBM、Intel 和 NVIDIA 技术团队的合作之下逐步完善至今。截止到 2013 年 12 月，OpenCL 标准的最新版本为 2.0。使用 OpenCL，可以编写能访问所有可用编程资源(如 DSP 和 Cell/B.E.处理器)的应用程序，包括 CPU、GPU 和其他处理器。到目前为止，Apple、AMD、Intel、IBM、NVIDIA 和其他业界巨头都已经实现了 OpenCL 标准，该标准在个人计算机（PC）、服务器、手持式/嵌入式设备、高性能计算，甚至是云计算系统等诸多领域都具有无限的应用潜力[45]。

图 2.2 OpenCL 商用标志

2.4 OpenCL 基础架构

OpenCL 的基础架构主要包括有以下几个部分：

- 平台模型（Platform Model）：异构系统的高层描述。
- 执行模型（Execution Model）：指令流在异构平台上执行的抽象表示。
- 内存模型（Memory Model）：OpenCL 中的内存区域集合以及 OpenCL 计算期间这些内存区域的交互规则。
- 编程模型（Programming Model）：程序员设计算法来实现一个应用时使用的高层抽象。

2.4.1 平台模型

OpenCL 平台模型定义了使用 OpenCL 的异构平台的一个高层表示，如图 2.3 所示。OpenCL 平台总是包括一个宿主机（Host）。宿主机与 OpenCL 程序外部的环境交互，包括 I/O 或程序用户的交互。

宿主机与一个或多个 OpenCL 设备（Device）进行连接，而这些设备用于执行指令流（或内核）。因此，OpenCL 设备通常称为计算设备（Compute Device）。设备可以是 CPU、GPU、DSP 或硬件提供以及 OpenCL 开发商支持的其他任何处理器。OpenCL 设备进一步划分为计算单元（Compute Unit），而计算单元还可以更进一步划分为一个或多个处理单元。设备上的计算都在处理单元（Process Element）中完成，在后面提到工作组（Work-Group）和 OpenCL 内存模型时，就会明白为什么 OpenCL 设备划分为处理单元和计算单元。

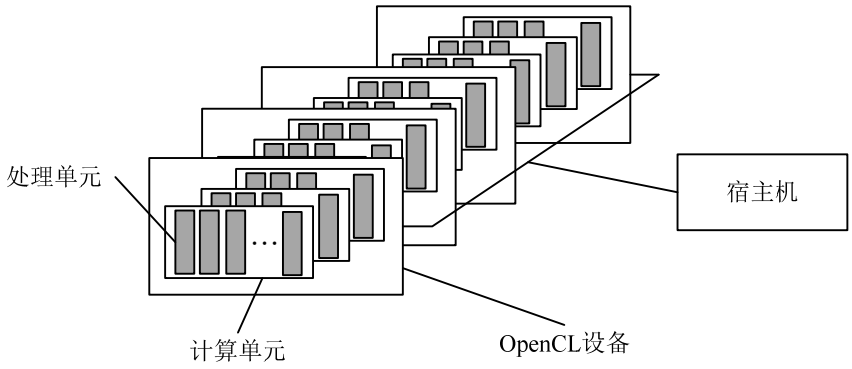


图 2.3 OpenCL 平台模型示意图

2.4.2 执行模型

OpenCL 应用程序由两个不同部分组成：一个宿主机程序（Host Program）和一个或多个内核（Kernel）程序组成的集合。宿主机程序在宿主机上运行。OpenCL 并没有定义宿主机程序如何工作的具体细节，只是定义了它与 OpenCL 中定义的对象如何交互。内核在 OpenCL 设备上执行，它们完成 OpenCL 应用的具体工作。内核通常是一些简单的函数，将输入内存对象转换为输出内存对象。

OpenCL 执行模型定义了内核如何执行，而内核是在宿主机上定义的。宿主机程序发出一个命令，提交内核到一个 OpenCL 设备上执行。由宿主机发出这个命令时，OpenCL 在运行时系统会创建一个整数索引空间。对应这个索引空间中的各个点将分别执行内核的一个实例。

考虑如图 2.4 所示的一个 2 维的 NDRange，使用小写字母 g 表示给定下标 x 或 y 时各维度中一个工作项的全局 ID。大写字母 G 指示索引空间各维度的大小。因此，各工作项在全局 NDRange 索引空间中有一个坐标(g_x, g_y)，全局索引空间的大小为(G_x, G_y)，工作项坐标值取值范围为$[0\ldots (G_x-1), 0\ldots(G_y-1)]$。将这个 NDRange 索引空间划分为工作组，小写字母 w 表示工作组 ID，大写字母 W 表示各个维度中工作组的个数。OpenCL 要求各个维度中工作组的数目能够整除 NDRange 索引空间各个维度的大小，因为这样可以保证所有工作组都是满的且大小一样。 x 和 y 方向的工作组大小用于为各个工作项定义一个局部索引空间（一个工作组内的索引空间称为局部索引空间）。局部索引空间中各维度的大小用大写字母 L 表示，用小写字母 l 表示工作组中的局部 ID。

在图 2.4 中，大小为 $G_x \times G_y$ 的 NDRange 索引空间将划分为 $W_x \times W_y$ 空间上的工作组，其索引为(w_x, w_y)。各个工作组的大小为 $L_x \times L_y$，这里可以得到：

$$\begin{aligned} L_x &= G_x / W_x \\ L_y &= G_y / W_y \end{aligned} \tag{2.10}$$

可以根据工作项的全局 ID (g_x, g_y)来定义一个工作项，或者结合局部 ID (l_x, l_y)和工作组 ID (w_x, w_y)来定义：

$$\begin{aligned} g_x &= w_x \times L_x + l_x \\ g_y &= w_y \times L_y + l_y \end{aligned} \tag{2.11}$$

使用如下关系即可求出局部 ID 和工作组 ID，其中符号“/”和“%”分别表示整除和取模操作。

$$\begin{aligned} w_x &= g_x / L_x \quad l_x = g_x \% L_x \\ w_y &= g_y / L_y \quad l_y = g_y \% L_y \end{aligned} \tag{2.12}$$

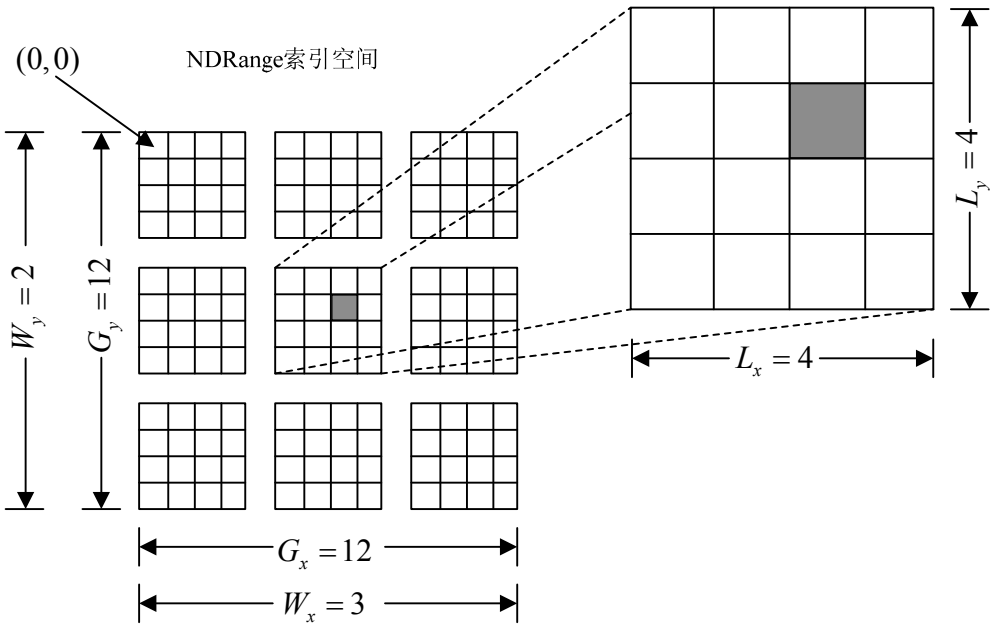


图 2.4 二维 NDRange 中全局 ID、局部 ID 和工作组索引之间关系举例

2.4.3 内存模型

执行模型定义了内核如何执行，内核与宿主机以及其他内核如何交互。而 OpenCL 的内存模型则定义了在 OpenCL 中支持的内存对象以及如何安全的使用这些内存对象的原则。

OpenCL 的内存模型定义了宿主机内存、全局内存、常量内存、局部内存以及私有内存等 5 种不同的内存区域。图 2.5 给出了这些内存区域的位置以及它们与平台和执行模型的关系，表 2.1 则描述了宿主机和内核是如何管理和使用这些内存的。

表 2.1 OpenCL 的内存分配与访问权限

		宿主机	全局	常量	局部	私有
主机端	分配	动态	动态	动态	动态	-
	访问	可读写	可读写	可读写	-	-
设备端	分配	-	-	静态	静态	静态
	访问	-	可读写	只读	可读写	可读写

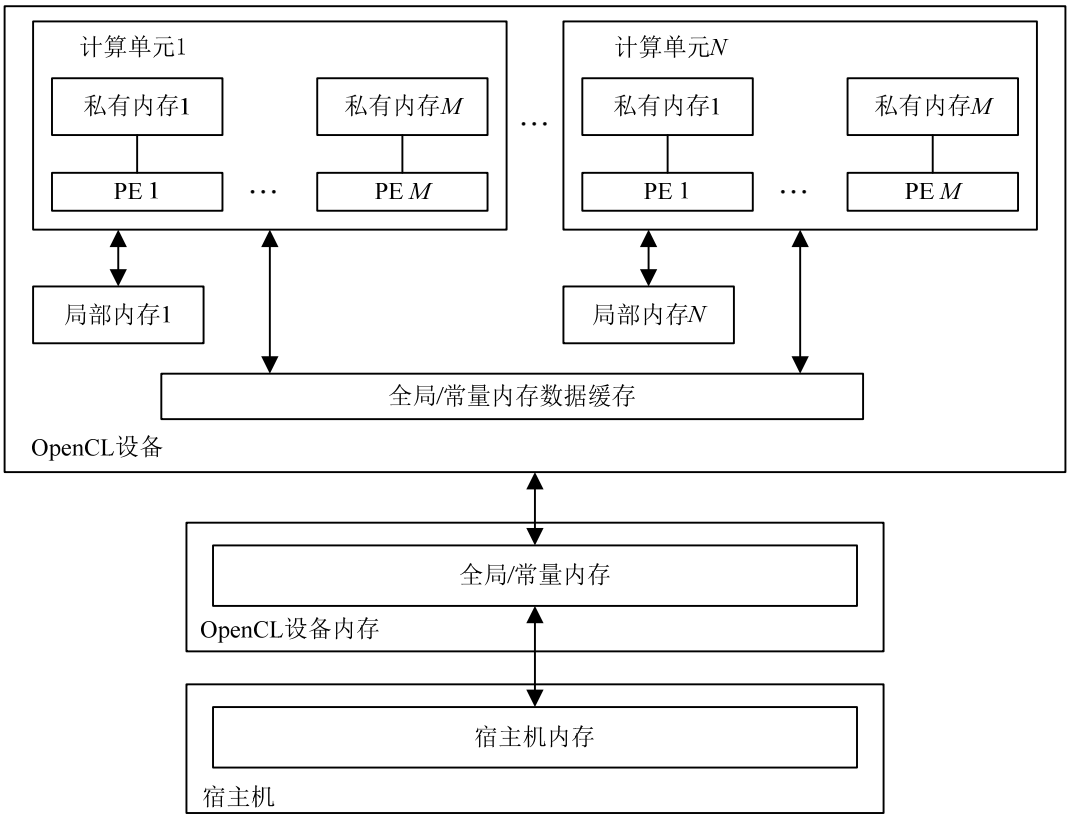


图 2.5 各类存储区域在平台中的位置

2.4.4 编程模型

OpenCL 的执行模型定义了一个 OpenCL 应用是如何映射到处理单元、内存区域以及宿主机的，这是一个以硬件为中心的模型。那么实际应用中的算法是如何映射到 OpenCL 中的呢？这就需要了解 OpenCL 的编程模型。编程模型就是程序员如何考虑他们的算法，因此这种模型本质上比执行模型更为灵活。

OpenCL 定义了基于任务并行和基于数据并行两种不同的并行编程模型。

在基于任务并行的程序设计模型中，程序员直接定义和处理并发任务。问题分解为可以并发运行的任务，然后再映射到一个并行计算计算机的处理单元来执行。如果任务是完全独立的，使用这个模型最为容易，不过这个模型也可以用于共享数据的任务。如果要利用一组任务来计算，只有当最后一个任务完成时这个任务才算完成。

在基于数据并行的程序设计模型中，程序员应从可以并发更新的数据元素集合角度来考虑问题。并行性表述为将相同的指令流（即一个任务）并发地应用到各个数据元素，并行性体现在数据中。本文提出的基于混沌理论的图像加密算法就是一种可进行基于数据并行划分的算法，将在第 3 和 4 章详细讨论。

2.5 本章小结

本章首先简要介绍了混沌理论和混沌密码学的基础知识，给出了在设计数字图像加密算法中的几个基本设计原则。作为异构计算的工业标准，OpenCL 能够使得我们所开发的算法必须可以在各种异构平台上自由移植，并且能充分而合理的利用到整个异构平台的所有计算资源，因此我们着重介绍了 OpenCL 技术的产生背景以及 OpenCL 的体系架构等基础知识，并指出了在加速数值计算中使用 OpenCL 的意义。

第 3 章 基于 OpenCL 与混沌映射的图像加密算法

在前面介绍的数字图像加密理论和 OpenCL 技术的基础之上，本章将提出两个新的基于 Logistic 映射的并行图像加密算法——CSIM 算法和 CBLP 算法。两种算法均具有高度的并行性与良好的安全性，在本章我们将通过给出这两种算法的详细设计原理和步骤来说明算法的并行特性，下一章我们将通过给出大量的仿真实验结果来表明算法具有良好的安全性能。

3.1 Logistic 混沌系统

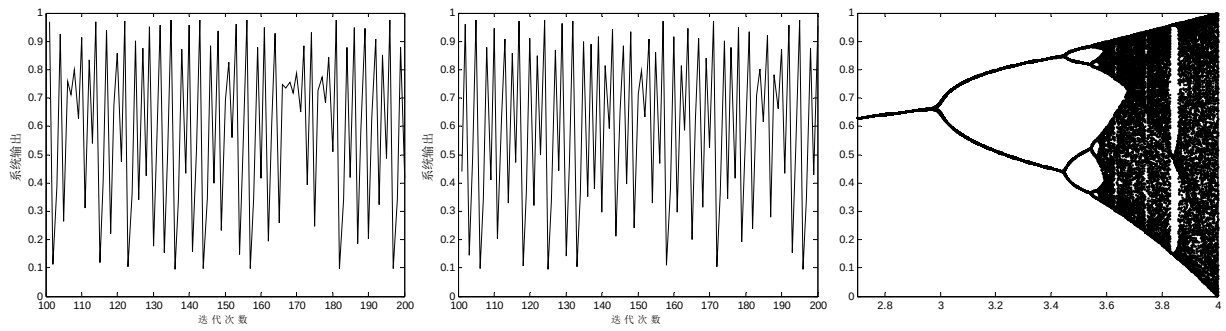
3.1.1 一维 Logistic 映射

一维 Logistic 映射是目前研究的最广泛的混沌映射，其数学表达式为：

$$x_{n+1} = \mu x_n (1 - x_n) \tag{3.1}$$

其中 $0 \le x_n \le 1$，μ 为控制参数，且 $0 < \mu \le 4$。当 $0 < \mu \le 1$ 时，式(3.1)有一个定常解 0，即无论初始值取何值，通过多次迭代，序列最终将收敛于 0。当 $1 < \mu \le 3$ 时，定常解为 0 和 $1-1/\mu$，且多次迭代后序列将收敛于这两个值中的一个，当 $3 < \mu \le 4$ 时，系统将由倍周期走向混沌。特别是当 $\mu \ge 3.5699456\ldots$ 时系统将进入混沌状态，即迭代生成的序列将处于一种伪随机分布状态，而且当 μ 越接近于 4 时，系统的混沌性越强，$\mu=4$ 时系统进入超混沌状态。一维 Logistic 映射轨迹如图 3.1(a)~(b)所示。

对于复杂的混沌系统可通过在其相空间中观察系统的状态轨迹来研究其特性。相空间是一种用来表示一个系统所有可能状态的空间，相空间中的点表示系统可能出现的状态。随着控制参数的变化，混沌系统中的的简单吸引子[1]会渐变为奇异吸引子，此时系统将处于混沌状态。这种控制参数变化到某个临界值而使系统的动力学性态发生定性变化的现象称为分岔，它是混沌系统内部固有的一种特性。因此可以利用分岔图来研究混沌系统的演变过程，一维 Logistic 映射的分岔图如图 3.1(c)所示。



(a) x_0=0.3333, μ=3.9　(b) x_0=0.5656, μ=3.9　(c) $x_n \in [0, 1], \mu \in [2.8, 4.0]$

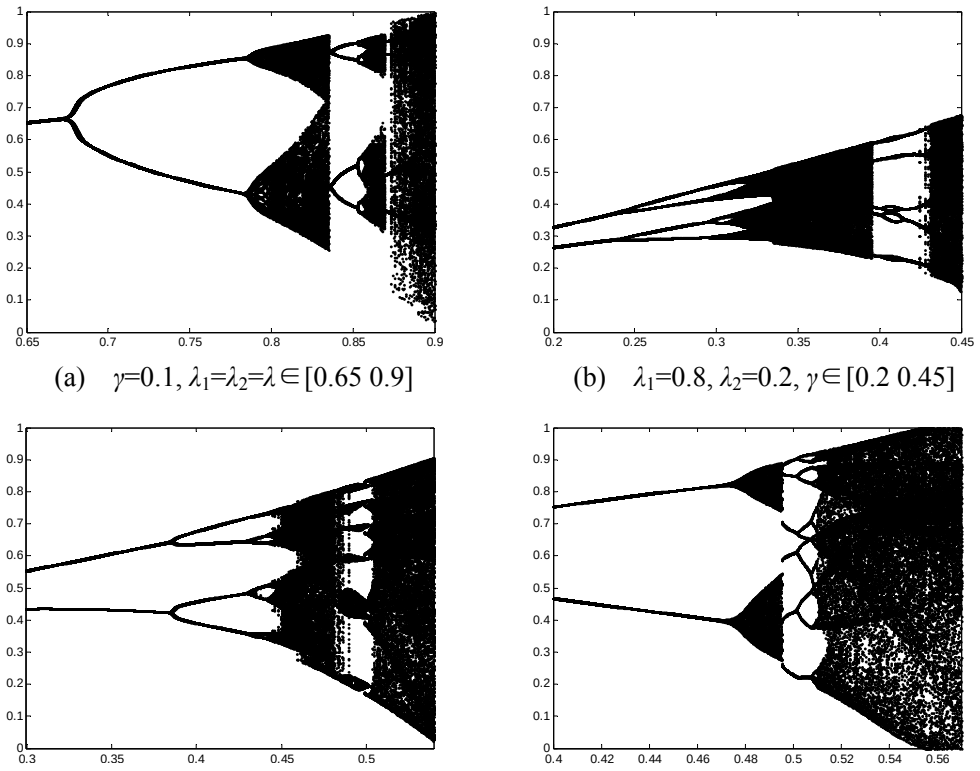
图 3.1 一维 Logistic 映射分叉与轨迹图

3.1.2 二维 Logistic 映射

在高维混沌映射中，二维 Logistic 映射不仅具有一维 Logistic 映射形式简单的优点，而且又具有高维混沌系统多参数、高随机性等特点，是一种性能优良的可用于信息加密的混沌系统。二维 Logistic 映射是一个复杂的混沌动力系统，其数学表达式为：

$$\begin{cases} x_{n+1} = 4\lambda_1 x_n(1-x_n) + \gamma y_n \\ y_{n+1} = 4\lambda_2 y_n(1-y_n) + \gamma x_n \end{cases} \tag{3.2}$$

在式(3.2)中，$0<x_n<1$，$0<y_n<1$，x_0 和 y_0 为系统初值，系统的动力学行为是由控制参数 λ_1, λ_2 和 γ 决定的。图 3.2 是二维 Logistic 系统的几种相空间图，其中图(a)表示系统通过阵发方式进入混沌；图(b)表示系统通过 Hopf 分岔方式进入混沌；图(c)表示系统通过叉状分叉方式进入混沌；图(d)表示系统通过 Feigenbaum 途径进入混沌，但同时又具有阵发方式的特征。从图中可以看出，二维 Logistic 系统的行为非常复杂，完全满足作为加密应用的条件。



(a) γ=0.1, $\lambda_1=\lambda_2=\lambda\in$[0.65 0.9]　(b) λ_1=0.8, λ_2=0.2, $\gamma\in$[0.2 0.45]

(c) λ_1=0.7, λ_2=0.3,$\gamma\in$[0.3 0.45]　(d) λ_1=0.6, λ_2=0.4,$\gamma\in$[0.4 0.57]

图 3.2　二维 Logistic 映射的相空间（以 x_n 为纵坐标）

3.2 基于混沌-自逆矩阵的图像加密算法

本节提出一种基于混沌-自逆矩阵（Chaotic & Self-Invertible Matrix，CSIM）的并行图像加密算法——CSIM 算法。该算法具有和经典的 Fridrich 架构类似的置乱和扩散部件，并且包含有像素位置置换和矩阵乘法等两种运算，其中位置置换完成图像的置乱功能，矩阵乘法完成图像的扩散功能。

3.2.1 自逆矩阵的构造

自逆矩阵是这样一种矩阵，即对于一个 N 阶矩阵 **A**，若满足 $\mathbf{A}=\mathbf{A}^{-1}$，则称 **A** 为自逆矩阵，其构造方法如算法 1 所述，其中 p 为正整数，k 为尺度常量，**I** 为单位矩阵，$\mathbf{A}_{22}$ 为任意 $N/2$ 阶矩阵。

算法 1. 自逆矩阵 **A** 的生成[52]

输入：矩阵 $\mathbf{A}_{22}$、p 和 k

输出：N 阶自逆矩阵 **A**

1. 生成 $\mathbf{A}_{11}$，令 $\mathbf{A}_{11} \leftarrow -\mathbf{A}_{22} \bmod p$
2. 生成 $\mathbf{A}_{12}$，令 $\mathbf{A}_{12} \leftarrow (\mathbf{I}-\mathbf{A}_{11})k \bmod p$
3. 生成 $\mathbf{A}_{21}$，令 $\mathbf{A}_{21} \leftarrow (\mathbf{I}+\mathbf{A}_{11})/k \bmod p$
4. 构造自逆矩阵 **A**，令 $\mathbf{A} \leftarrow [\mathbf{A}_{11}\ \mathbf{A}_{12};\ \mathbf{A}_{21}\ \mathbf{A}_{22}]$

3.2.2 算法原理与步骤

CSIM 算法具体的加密流程框图如图 3.3 所示，算法的具体原理和步骤如下。

(1) 初始化待加密图像与密钥

将待加密的灰度图像保存在矩阵**A**中，用M和N表示其行数和列数。密钥key为：$key=\{key_1, key_2\}$，其中$key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma\}$，$key_2=\{\alpha_0, \eta\}$，$x_0$、$y_0$和$\alpha_0$是Logistic映射的初值，$\lambda_1$、$\lambda_2$、$\eta$和$\gamma$是Logistic映射的控制参数。

(2) 生成置乱矩阵

置乱矩阵共有两个：行置乱矩阵和列置乱矩阵，分别用**R**和**C**表示，其生成过程如下。

①用密钥key_1作为二维Logistic映射的控制参数和初始值以生成初始密钥流$\{x'_j\}$和$\{y'_j\}$，$j=0,1,\ldots,N-1$。为使系统发散，需要在输入初始状态后迭代排除前100个值，得到两个二维Logistic映射的新的初始值x'_0和y'_0。然后将x'_0和y'_0带入式(3.2)并迭代N次后即可得混沌序列$\{x'_j\}$和$\{y'_j\}$。

②用密钥key_2作为一维Logistic映射的控制参数和初始值以生成初始密钥流$\{\alpha'_j\}$，$j=0,1,\ldots,N-1$为使系统发散，需要在输入初始状态后迭代排除前100个值，得到一维Logistic映射的新的初始值α'_0。然后将α'_0带入式(3.1)并迭代N次后即可得长度为N的初始密钥流$\{\alpha'_j\}$。

③令$\beta'_j=(x'_j+y'_j)/2$，形成序列$\{\beta'_j\}$。将$\{\alpha'_j\}$和$\{\beta'_j\}$作为二维Logistic映射的初始值，并以密钥key_1中的λ_1，λ_2和γ作为控制参数带入式(3.2)并分别迭代M次，即可生成元素个数均为$M\times N$的双精度序列$\{x_i\}$和$\{y_i\}$，$i=0,1,\ldots,MN-1$。例如，将(α'_1, β'_1)带入式(3.2)迭代M次后可生成序列$\{x_i\}$和$\{y_i\}$，$i=0,1,\ldots,M-1$；将(α'_2, β'_2)带入式(3.2)迭代M次后可生成序列$\{x_i\}$和$\{y_i\}$，$i=M, M+1,\ldots,2M-1$；将(α'_N, β'_N)带入式(3.2)迭代M次后可生成序列$\{x_i\}$和$\{y_i\}$，$i=(N-1)M$，$(N-1)M+1,\ldots,MN-1$等。

④分别对序列$\{x_i\}$和$\{y_i\}$执行放大取模操作，即$\{r_i\}=\{x_i\}\times 10^{14} \bmod M$，$\{c_i\}=\{y_i\}\times 10^{14} \bmod$

N。然后将序列$\{r_i\}$和$\{c_i\}$分别转换成矩阵**R**和**C**即可。

$$\mathbf{R}=\begin{pmatrix} r_0 & r_1 & \cdots & r_{N-1} \\ r_N & r_{N+1} & \cdots & r_{2N-1} \\ \vdots & \vdots & \vdots & \vdots \\ r_{(M-1)N} & r_{(M-1)N+1} & \cdots & r_{MN-1} \end{pmatrix},\ \mathbf{C}=\begin{pmatrix} c_0 & c_M & \cdots & c_{(N-1)M} \\ c_1 & c_{M+1} & \cdots & c_{(N-1)M+1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{M-1} & c_{2M-1} & \cdots & c_{NM-1} \end{pmatrix}$$

(3) 生成扩散密钥矩阵

①用$\{p'_k\}$和$\{t'_k\}$分别表示$\{\beta_j\}$前半部分和后半部分，对$\{p'_k\}$和$\{t'_k\}$分别执行放大取模和线性变换操作，即$\{p_k\}=\{p'_k\}\times 10^{14} \bmod 256$，$\{t_k\}=\{t'_k\}\times 0.3+3.7$，得序列$\{p_k\}$和$\{t_k\}$，$k=0, 1, \ldots, N/2-1$。取$\{\alpha'_j\}$的前半部分，并表示为$\{\alpha_k\}$，即$\{\alpha_k\}=\{\alpha'_j\}$，$k$，$j=0, 1, \ldots, N/2-1$。

②将$\{\alpha_k\}$、$\{t_k\}$和$\{p_k\}$分别作为一维 Logistic 映射的初始值、控制参数和迭代次数并带入算法 2 可生成长度为 $N/2\times N/2$ 的混沌序列$\{\kappa_k\}$，$n=0, 1, \ldots, N/2\times N/2-1$。然后将$\{\kappa_k\}$写成矩阵形式有：

$$\mathbf{K}_{22}=\begin{pmatrix} \kappa_0 & \kappa_1 & \cdots & \kappa_{N/2-1} \\ \kappa_{N/2} & \kappa_{N/2+1} & \cdots & \kappa_{N-1} \\ \vdots & \vdots & \vdots & \vdots \\ \kappa_{(N/2-1)N/2} & \kappa_{(N/2-1)N/2+1} & \cdots & \kappa_{N/2\times N/2-1} \end{pmatrix}$$

算法 2. 混沌序列$\{\kappa_k\}$的生成

输入：序列$\{\alpha_k\}$、$\{t_k\}$和$\{p_k\}$，$k=0, 1, \ldots, N/2-1$

输出：序列$\{\kappa_k\}$，$n=0, 1, \ldots, N/2\times N/2-1$

```
1.  for k ← 0 to N/2 − 1
2.      κ_k ← L^1D(α_k, t_k, p_0) × 10^14 mod 256
3.      for m ← 1to N/2 − 1
4.          κ_{mN/2+k} ← L^1D(κ_{(m-1)N/2+k}, t_k, p_m) × 10^14 mod 256
5.      end
6.  end
```

③结合②中生成的 $\mathbf{K}_{22}$ 和算法 1，即可得扩散密钥矩阵 **K**。

(4) 图像置乱

从原始图像矩阵**A**中的第一行象素开始，对**A**中的行向量$\mathbf{a}=\{a_{11},a_{12},\ldots,a_{1N}\}$，按照矩阵 **R** 中对应的行向量$\mathbf{r}=\{r_1,r_2,\ldots,r_N\}$进行坐标变换，对其他的行向量重复这个过程，即可完成行方向的置乱，得到加密矩阵$\mathbf{A}_1$。同理将矩阵$\mathbf{A}_1$中的列向量按照**C**中对应的列向量进行坐标变换，对其他的列向量重复这个过程，即可完成列方向的置乱，得到加密矩阵$\mathbf{A}_{in}$。

(5) 图像扩散

使用式(3.3)即可完成图像的扩散操作，其中 $\mathbf{A}_c$ 表示密图，**K** 为步骤(3)中生成的自逆子密

钥矩阵，$\mathbf{A}_{in}$ 为步骤(4)中生成的图像。

$$\mathbf{A}_c = \mathbf{A}_{in}\mathbf{K} \mod 256 \tag{3.3}$$

(6) 轮加密操作

对图像$\mathbf{A}_c$重复执行第(4)~(5)步*Rnd*次即可完成图像的加密全部过程。*Rnd*的大小可由实验测得，我们通过实验测出当$Rnd \geq 3$时，CSIM算法的各项性能指标（如密钥敏感性、明文敏感性、灰度直方图以及相邻像素相关性等）就能达到令人满意的效果。

通过以上的步骤可知，CSIM 算法是个对称的加密算法，因此该算法的解密过程与加密过程类似，只需对加密过程实施相反的操作即可恢复原始图像。

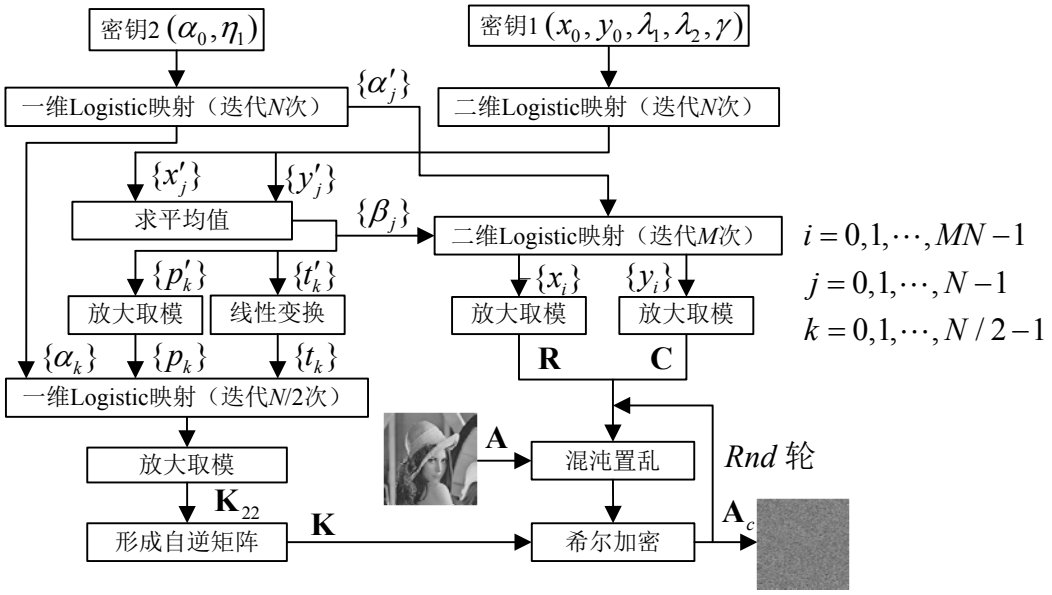


图 3.3 CSIM 算法的加密过程

3.2.3 CPU 实现

图 3.3 给出了 CSIM 算法的加密过程框图，根据 CSIM 的设计原理和步骤，算法 3 给出了 CSIM 在 CPU 上的实现伪代码（即算法的串行实现）。

算法 3. 内核 CSIM 算法的 CPU 实现

输入：原始图像 **A** 密钥 $key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma\}$，$key_2=\{\alpha_0, \eta\}$，$M$，$N$

输出：密图 $\mathbf{A}_c$

```
1. for j ← 0 to N – 1 //产生子密钥矩阵
2.     (x'_j, y'_j) ← L^2D(x_0, y_0, λ_1, λ_2, γ, 100 + j)
3.     α'_j ← L^1D(α_0, η, 100 + j)
4.     β'_j ← ( x'_j + y'_j) /2
5.     for i ← 0 to M – 1
6.         (x_{i+jM} , y_{i+jM}) ← L^2D(α'_j, β'_j, λ_1, λ_2, γ, i + 1)
```

7. $\mathbf{R}(i, j) \leftarrow x_{i+jM} \times 10^{14} \bmod M$
8. $\mathbf{C}(i, j) \leftarrow y_{i+jM} \times 10^{14} \bmod N$
9. **end**
10. **end**
11. **for** $k \leftarrow 0$ **to** $N/2 - 1$ //产生自逆矩阵
12. $p_k \leftarrow \beta'_k \times 10^{14} \bmod 256$
13. $t_k \leftarrow \beta'_{k+N/2} \times 0.3 + 3.7$
14. $\mathbf{K}_{22}(0, k) \leftarrow L^{1D}(\alpha_k, t_k, p_0) \times 10^{14} \bmod 256$
15. **for** $m \leftarrow 1$ **to** $N/2 - 1$
16. $\mathbf{K}_{22}(m, k) \leftarrow L^{1D}(\mathbf{K}_{22}(m-1, k), t_k, p_m) \times 10^{14} \bmod 256$
17. **end**
18. **end**
19. $\mathbf{K} \leftarrow \mathbf{K}_{22}$
20. **for** $r \leftarrow 0$ **to** $Rnd - 1$
21. **for** $i \leftarrow 0$ **to** $M - 1$ // 列置乱
22. **for** $j \leftarrow 0$ **to** $N - 1$
23. $\mathbf{A}(i, j) \leftrightarrow \mathbf{A}(\mathbf{R}(i, j), j)$
24. **end**
25. **end**
26. **for** $i \leftarrow 0$ **to** $M - 1$ // 行置乱
27. **for** $j \leftarrow 0$ **to** $N - 1$
28. $\mathbf{A}(i, j) \leftrightarrow \mathbf{A}(i, \mathbf{C}(i, j))$
29. **end**
30. **end**
31. **for** $i \leftarrow 0$ **to** $M - 1$ //图像扩散
32. **for** $j \leftarrow 0$ **to** $N - 1$
33. $tmp \leftarrow 0$
34. **for** $k \leftarrow 0$ **to** $N - 1$
35. $tmp \leftarrow tmp + \mathbf{A}(i, k) \times \mathbf{K}(k, j)$
36. **end**
37. $\mathbf{A}_c(i, j) \leftarrow tmp \bmod 256$
38. **end**
39. **end**
40. **end**

3.2.4 GPU 实现

从上一节可知，CSIM 算法的计算开销主要集中在子密钥矩阵生成、图像置乱和扩散这三个阶段。因此，如果能并行化这三个阶段，那么就可以在不降低安全性的前提下大幅度减少算法的执行时间，有利于提高算法的效率。同时也注意到，CSIM 算法的这三个阶段都具有高度的并行性，这为并行化该算法提供了可能性。

(1) 子密钥矩阵生成的并行化实现

置乱矩阵 **R** 和 **C** 可以用内核函数*__kernel void ChaosKeyMatGenRC*()生成，内核的伪代码如算法 4 所示。在算法 4 中，OpenCL 主机参数 *NDRange* 设定为 1 维，工作项和工作组参数分别设定为 *global_work_size*=[*N*, 0, 0]，*local_work_size*=[*N*, 0, 0]，即 OpenCL 使用 GPU 的 1 个工作组，且该工作组中包含 *N* 个工作项，所以算法 4 执行操作 $j \leftarrow get_global_id(0)$后，$j = 0, 1, \ldots, N-1$。

密钥矩阵 **K** 可以用内核函数*__kernel void ChaosKeyMatGenK*()生成，内核的伪代码如算法 5 所示。在算法 5 中，*NDRange* 设定为 1 维，工作项参数 *global_work_size*=[*N*/2, 0, 0]，*local_work_size* =[*N*/2, 0, 0]，即内核使用 1 个包含有 *N*/2 个工作项的工作组，因此算法 5 执行操作 $k \leftarrow get_global_id(0)$后，$k = 0, 1, \ldots, N/2-1$。

算法 4. 内核*__kernel void ChaosKeyMatGenRC*()

输入：密钥 $key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma\}$, $key_2=\{\alpha_0, \eta\}$, M, N

输出：密钥矩阵 **R** 和 **C** 以及混沌序列$\{\alpha'_j\}$和$\{\beta'_j\}$, $j = 0, 1, \ldots, N-1$

1. $j \leftarrow get\_global\_id(0)$
2. $(x'_j, y'_j) \leftarrow L^{2D}(x_0, y_0, \lambda_1, \lambda_2, \gamma, 100+j)$
3. $\alpha'_j \leftarrow L^{1D}(\alpha_0, \eta, 100+j)$
4. $\beta'_j \leftarrow (x'_j + y'_j)/2$
5. **for** $i \leftarrow 0$ **to** $M-1$
6. $\quad (x_{i+jM}, y_{i+jM}) \leftarrow L^{2D}(\alpha'_j, \beta'_j, \lambda_1, \lambda_2, \gamma, i+1)$
7. $\quad \mathbf{R}(i, j) \leftarrow x_{i+jM} \times 10^{14} \bmod M$
8. $\quad \mathbf{C}(i, j) \leftarrow y_{i+jM} \times 10^{14} \bmod N$
9. **end**

算法 5. 内核*__kernel void ChaosKeyMatGenK*()

输入：密钥 $key_2=\{\alpha_0, \eta\}$, M, N 以及$\{\alpha'_j\}$和$\{\beta'_j\}$, $j = 0, 1, \ldots, N-1$

输出：密钥矩阵 $\mathbf{K}_{22}$

1. $k \leftarrow get\_global\_id(0)$
2. $p_k \leftarrow \beta'_k \times 10^{14} \bmod 256$

3. $t_k \leftarrow \beta'_{k+N/2} \times 0.3 + 3.7$
4. $\kappa_k \leftarrow L^{1D}(\alpha_k, t_k, p_0) \times 10^{14} \bmod 256$
5. **for** $m \leftarrow 1$ **to** $N/2 - 1$
6. $\quad \kappa_{mN/2+k} \leftarrow L^{1D}(\kappa_{(m-1)N/2+k}, t_k, p_m) \times 10^{14} \bmod 256$
7. **end**

内核参数建立完成后，主机端代码将利用命令队列对即将在设备上执行的内核进行排队，这个过程是通过调用函数 *clEnqueueNDRangeKernel*()完成的。但要注意的是，执行内核排队并不表示这个内核会立即执行，因为内核执行会放在命令队列中，以后再由设备“消费”。也就是说，函数 *clEnqueueNDRangeKernel*()被调用之后，可能并不会在设备上立即执行内核，而是让内核等待之前的事件完成之后再执行。所以将内核 *ChaosKeyMatGenRC*()放入命令队列之后应使用函数 *clFinish*()等待该内核执行完毕，*clFinish*()是个阻塞函数，当其返回时表示所等待执行的内核已执行完毕。此时，读取 *ChaosKeyMatGenRC*()输出的数据$\{\alpha'_j\}$和$\{\beta'_j\}$以及 **R**、**C** 才是正确的。在读取$\{\alpha'_j\}$和$\{\beta'_j\}$后，内核 *ChaosKeyMatGenK*()将被放入命令队列等待执行，此时同样应使用 *clFinish*()等待内核执行完毕，当 *clFinish*()返回时，读取 $\mathbf{K}_{22}$ 到主机端，然后在 CPU 上使用算法 1 即可得密钥矩阵 **K**。

(2) 图像置乱的并行化实现

内核 *__kernel void ImageConfusion*()负责图像的逐行和逐列置乱，其实现如算法 6 所示。执行该内核之前，OpenCL 主机参数 *NDRange* 设定为 1 维，工作项和工作组参数分别设定为 *global_work_size*=[Δ, 0, 0]，*local_work_size*=[Δ, 0, 0]，其中 Δ=*max*(*M*, *N*)，因此执行操作 $j \leftarrow$*get_global_id*(0)后，$j = 0, 1, \ldots, \Delta - 1$。

需要注意的是，图像的行置乱和列置乱之间并不具有并行性，也就是说它们是有依赖关系的，在本算法中，CSIM 将先对图像执行列置乱操作，然后才执行行置乱操作。所以在该内核中当每个工作项均完成了列置乱后，所有工作项必须在全局同步点进行同步，然后才能执行行置乱。这里使用函数 *barrier*(*GLK_GLOBAL_MEM_FENCE*)对所有工作项进行同步。当建立好内核参数后，主机端代码将内核 *ImageConfusion*()放入命令队列等待执行，当函数 *clFinish*()返回时，读取密文图像 $\mathbf{A}_{in}$ 到主机端即可完成图像的置乱操作。

算法 6. 内核 *__kernel void ImageConfusion*

输入：明文图像 **A**，密钥矩阵 **R** 和 **C**，*M*，*N*

输出：置乱图像 $\mathbf{A}_{in}$

1. $j \leftarrow$*get_global_id*(0)
2. **if** $j < N$ **then** // j 为列号
3. $\quad$ **for** $i \leftarrow 0$ **to** $M - 1$
4. $\quad\quad \mathbf{A}(i, j) \leftrightarrow \mathbf{A}(\mathbf{R}(i, j), j)$

5. **end**
6. **end**
7. *barrier* (*GLK_GLOBAL_MEM_FENCE*) // 同步
8. **if** $j < M$ **then** // j 为行号
9. **for** $i \leftarrow 0$ **to** $N-1$
10. $\mathbf{A}(j, i) \leftrightarrow \mathbf{A}(j, \mathbf{C}(j, i))$
11. **end**
12. **end**
13. $\mathbf{A}_{in} \leftarrow \mathbf{A}$

(3) 图像扩散的并行化实现

图像的扩散操作是由内核函数*__kernel void ImageDiffusion*()完成的，其具体实现如算法 7 所示。在算法 7 中，主机参数 *NDRange* 设定为 1 维，工作项设定为 *global_work_size*=[*M*, 0, 0]，*local_work_size*=[*M*, 0, 0]。另外注意到在该内核中，每个工作项计算矩阵 $\mathbf{A}_c$ 的一整行，使用 $\mathbf{A}_{in}$ 的同一行计算 $\mathbf{A}_c$ 中一行中的各元素，所以将 $\mathbf{A}_{in}$ 的一行复制到私有内存可以显著降低内存移动的开销，大幅提高矩阵乘法的速度，如图 3.4 所示。

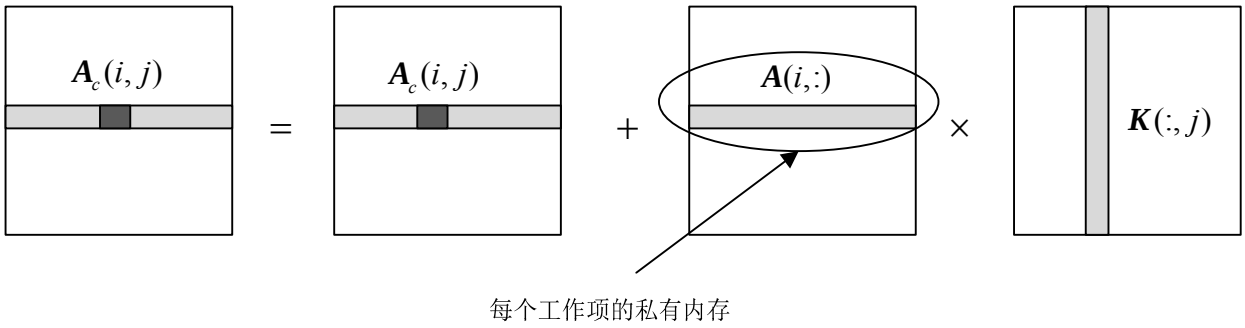


图 3.4 并行矩阵乘法示意图

算法 7. 内核*__kernel void ImageDiffusion*()
输入：图像矩阵 $\mathbf{A}_{in}$，密钥矩阵 $\mathbf{K}$, M, N
输出：密文图像 $\mathbf{A}_c$

1. $i \leftarrow$*get_global_id*(0)
2. **for** $k \leftarrow 0$ **to** $N-1$
3. $\mathbf{A}_{wrk}(k) \leftarrow \mathbf{A}(m, k)$
4. **end**
5. **for** $j \leftarrow 0$ **to** $N-1$
6. $tmp \leftarrow 0$
7. **for** $k \leftarrow 0$ **to** $N-1$

8. $tmp \leftarrow tmp + \mathbf{A}_{wrk}(k) \times \mathbf{K}(k, j)$
9. **end**
10. $\mathbf{A}_c(i, j) \leftarrow tmp \bmod 256$
11. **end**

当建立好内核参数后，主机端代码将内核 *ImageDiffusion*()放入命令队列等待执行，当函数 *clFinish*()返回时，读取密文图像 $\mathbf{A}_c$ 到主机端即可完成图像的一轮加密过程。

(4) OpenCL 主机端实现

算法 8 是 CSIM 并行算法的 OpenCL 主机端实现，也是算法的整个加密过程，由于 CSIM 是对称密码算法，其解密过程的 OpenCL 实现与加密过程是类似的，这里不再赘述。

算法 8. OpenCL 主机端实现（CSIM 加密过程）

输入：原始图像 **A** 密钥 $key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma\}$，$key_2=\{\alpha_0, \eta\}$

输出：密图 $\mathbf{A}_c$

1. [**R**, **C**] ← *clEnqueueNDRange* (*ChaosKeyMatGenRC*(key_1))
2. $\mathbf{K}_{22}$ ← *clEnqueueNDRange* (*ChaosKeyMatGenK* (key_2))
3. **K** ← $\mathbf{K}_{22}$
4. **for** $r \leftarrow 0$ **to** $Rnd - 1$ // $Rnd \geq 3$
5. $\mathbf{A}_{in}$ ← *clEnqueueNDRange* (*ImageConfusion*(**A**, **R**, **C**))
6. $\mathbf{A}_{tmp}$ ← *clEnqueueNDRange* (*ImageDiffusion*($\mathbf{A}_{in}$, **K**))
7. $\mathbf{A}_{in} \leftarrow \mathbf{A}_{tmp}$
8. **end**
9. $\mathbf{A}_c \leftarrow \mathbf{A}_{tmp}$

3.3 基于混沌-比特级置乱的图像加密算法

本节将提出一种基于混沌和比特级置乱（Chaotic & Bit Level Permutation，CBLP）的并行图像加密算法——CBLP 算法。CBLP 算法主要包含三种运算，即像素位置置换、比特旋转（Bit Rotation，BR）以及像素值异或（Exclusive OR，XOR）运算，其中位置置换负责置乱像素位置，BR 和 XOR 运算负责扩散图像的像素值。

3.3.1 算法原理与步骤

CBLP算法的加密过程如图3.6所示，具体原理和步骤如下。

(1) 初始化待加密图像与密钥

将待加密的灰度图像保存在矩阵**A**中，用M和N分别表示其行数和列数。密钥key为：$key=\{key_1, key_2\}$，其中$key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma_1\}$，$key_2=\{\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2\}$，这里$x_0$、$y_0$、$\alpha_0$和 β_0是Logistic混沌映射的初值，λ_k、η_k和γ_k为Logistic混沌映射的控制参数，$k = 1, 2$。

(2) 生成子密钥矩阵

子密钥矩阵**P**和**Q**用于控制像素在比特旋转和异或，矩阵**R**和**C**用于控制像素的逐行和逐列置乱，四个矩阵的维度均为$M \times N$。具体的生成步骤如下：

①用密钥key_1作为二维Logistic映射的控制参数和初始值以生成初始密钥流$\{x'_j\}$和$\{y'_j\}$，$j = 0, 1, \ldots, N-1$。为使系统发散，需要在输入初始状态后迭代排除前100个值，得到两个二维Logistic映射的新的初始值x'_0和y'_0 。然后将x'_0和y'_0带入式(3.2)并迭代N次即可得两个长度为N的混沌序列$\{x'_j\}$和$\{y'_j\}$。

②用密钥key_2作为二维Logistic映射的控制参数和初始值以生成初始密钥流$\{\alpha'_j\}$和$\{\beta'_j\}$，$j = 0,1, \ldots, N-1$。为使系统发散，需要在输入初始状态后迭代排除前100个值，得到两个二维Logistic映射的新的初始值α'_0和β'_0。然后将α'_0和β'_0带入式(3.2)并迭代N次即可得两个长度为N的混沌序列$\{\alpha'_j\}$和$\{\beta'_j\}$。

③分别对$\{x'_j\}$和$\{y'_j\}$执行线性变换操作，即$\{\tilde{x}_j\}=\{x'_j\}\times 0.3+3.7$，$\{\tilde{y}_j\}=\{y'_j\}\times 0.3+3.7$。然后再分别以$\{\tilde{x}_j\}$和$\{\tilde{y}_j\}$作为一维Logistic映射的控制参数，以$\{x'_j\}$和$\{y'_j\}$作为初始值带入式(3.1)并分别迭代$M$次即可生成另一对混沌序列，记为$\{x_j\}$和$\{y_j\}$，$i = 0, 1, \ldots, MN-1$。

④对得到的序列$\{x_j\}$和$\{y_j\}$执行一次放大取模操作，即有$\{r_i\}=\{x_i\}\times 10^{14} \bmod M$，$\{c_i\}=\{y_i\}\times 10^{14} \bmod N$。然后将序列$\{r_i\}$和$\{c_i\}$分别转换成矩阵形式，即可得矩阵**R**和**C**，即：

$$\mathbf{R}=\begin{pmatrix} r_0 & r_1 & \cdots & r_{N-1} \\ r_N & r_{N+1} & \cdots & r_{2N-1} \\ \vdots & \vdots & \vdots & \vdots \\ r_{(M-1)N} & r_{(M-1)N+1} & \cdots & r_{MN-1} \end{pmatrix},\quad \mathbf{C}=\begin{pmatrix} c_0 & c_M & \cdots & c_{(N-1)M} \\ c_1 & c_{M+1} & \cdots & c_{(N-1)M+1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{M-1} & c_{2M-1} & \cdots & c_{NM-1} \end{pmatrix}$$

⑤分别对$\{\alpha'_j\}$和$\{\beta'_j\}$执行线性变换操作，即$\{\tilde{\alpha}_j\}=\{\alpha'_j\}\times 0.3+3.7$，$\{\tilde{\beta}_j\}=\{\beta'_j\}\times 0.3+3.7$。然后将$\{\alpha'_j\}$、$\{\tilde{\alpha}_j\}$和$\{r_i\}$分别作为一维Logistic映射的初始值、控制参数和迭代次数并带入算法9即可生成长度为$M\times N$的混沌序列，记为$\{p_i\}$。

同理将$\{\beta'_j\}$、$\{\tilde{\beta}_j\}$和$\{c_i\}$分别作为一维Logistic映射的初始值、控制参数和迭代次数并带入算法9即可生成另一长度为$M \times N$的混沌序列，记为$\{q_i\}$。然后将序列$\{p_i\}$和$\{q_i\}$分别写成矩阵形式，即可得矩阵**P**和**Q**，即：

$$\mathbf{P}=\begin{pmatrix} p_0 & p_1 & \cdots & p_{N-1} \\ p_N & p_{N+1} & \cdots & p_{2N-1} \\ \vdots & \vdots & \vdots & \vdots \\ p_{(M-1)N} & p_{(M-1)N+1} & \cdots & p_{MN-1} \end{pmatrix},\quad \mathbf{Q}=\begin{pmatrix} q_0 & q_1 & \cdots & q_{N-1} \\ q_N & q_{N+1} & \cdots & q_{2N-1} \\ \vdots & \vdots & \vdots & \vdots \\ q_{(M-1)N} & q_{(M-1)N+1} & \cdots & q_{MN-1} \end{pmatrix}$$

算法 9. 混沌序列$\{p_i\}$和$\{q_i\}$的生成

输入：序列$\{\alpha'_j, \beta'_j\}$，$\{\tilde{\alpha}_j, \tilde{\beta}_j\}$和$\{r_i, c_i\}$, $i = 0, 1, \ldots, MN-1$，　$j = 0, 1, \ldots, N-1$

输出：序列$\{p_i\}$和$\{q_i\}$

1. $i \leftarrow 0$
2. **for** $j \leftarrow 0$ **to** $N-1$
3. $\quad p_i \leftarrow L^{1D}(\alpha_i', \tilde{\alpha}_j, r_i) \times 10^{14} \bmod 256$
4. $\quad q_i \leftarrow L^{1D}(\beta_i', \tilde{\beta}_j, c_i) \times 10^{14} \bmod 256$
5. $\quad$ **for** $m \leftarrow 0$ **to** $M-1$
6. $\quad\quad i \leftarrow i+1$
7. $\quad\quad p_{jM+m} \leftarrow L^{1D}(p_{jM+(m-1)}, \tilde{\alpha}_j, r_i) \times 10^{14} \bmod 256$
8. $\quad\quad q_{jM+m} \leftarrow L^{1D}(q_{jM+(m-1)}, \tilde{\beta}_j, c_i) \times 10^{14} \bmod 256$
9. $\quad$ **end**
10. **end**

(3) 图像置乱

从原始图像矩阵**A**中的第一行象素开始，对**A**中的行向量**a**=$\{a_{11}, a_{12}, \ldots, a_{1N}\}$，按照矩阵**R**中对应的行向量**r**=$\{r_1, r_2, \ldots, r_N\}$进行坐标变换，对其他的行向量重复这个过程，即可完成行方向的置乱，得到加密矩阵**A**′。同理将矩阵**A**′中的列向量按照**C**中对应的列向量进行坐标变换，对其他的列向量重复这个过程，即可完成列方向上的置乱。

(4) 图像扩散

图像的扩散包括两种运算：比特旋转和异或运算。对置乱后的图像矩阵**A**依次执行下列操作即可完成扩散，其中符号“<<<”和“>>>”分别表示比特左旋转和右旋转，“⊕”为比特异或操作符。

$$\mathbf{A}(i,j) = [\mathbf{A}(i,j) \oplus \mathbf{P}(i-1,j) \oplus \mathbf{A}(i-1,j)] <<< [\mathbf{P}(i,j) \bmod 8],\ i = 1, 2, ..., M-1 \tag{3.4}$$

$$\mathbf{A}(i,j) = [\mathbf{A}(i,j) \oplus \mathbf{Q}(i,j-1) \oplus \mathbf{A}(i,j-1)] >>> [\mathbf{Q}(i,j) \bmod 8],\ j = 1, 2, ..., N-1 \tag{3.5}$$

$$\mathbf{A}(i-1,j) = [\mathbf{A}(i-1,j) \oplus \mathbf{Q}(i,j) \oplus \mathbf{A}(i,j)] >>> [\mathbf{Q}(i-1,j) \bmod 8],\ i = M-1, ..., 2, 1 \tag{3.6}$$

$$\mathbf{A}(i,j-1) = [\mathbf{A}(i,j-1) \oplus \mathbf{Q}(i,j) \oplus \mathbf{A}(i,j)] >>> [\mathbf{Q}(i-1,j) \bmod 8],\ j = N-1, ..., 2, 1 \tag{3.7}$$

式(3.4)~(3.7)表示对矩阵**A**分别执行从上到下、从左到右、从下到上和从右到左等四个方向的扩散，这样做的好处之一就是能将明文**A**中任意一个像素的值尽可能扩散到整个密文图像中，有利于提高算法的明文敏感性。具体的扩散过程如图3.5所示，从图中可以看出当执行完这四个方向的扩散后，像素**A**(i,j)已被扩散到整个图像之中了。

另外要注意的是，CBLP 算法中比特的左旋转和右旋转操作在 C 语言中是没有指定的操作符的，但实际上这两种运算可以通过以下两个表达式实现，即

$$R^l(x,s,n) = x << n \mid x >> (s-n) \tag{3.8}$$

$$R^r(x,s,n) = x >> n \mid x << (s-n) \tag{3.9}$$

在式(3.8)~(3.9)中，符号“<<”、“>>”和“|”分别表示C语言中的左移、右移和按位或运算符，函数$R^l(x, s, n)$和$R^r(x, s, n)$分别表示将变量x向左和向右旋转n位，s是变量x的二进制比特位数，且$s \geqslant n$。

通过以上分析可知，CBLP 算法也是个对称的加密算法，因此该算法的解密过程与加密过程是类似的，只需对加密过程实施相反的操作即可恢复原始图像。

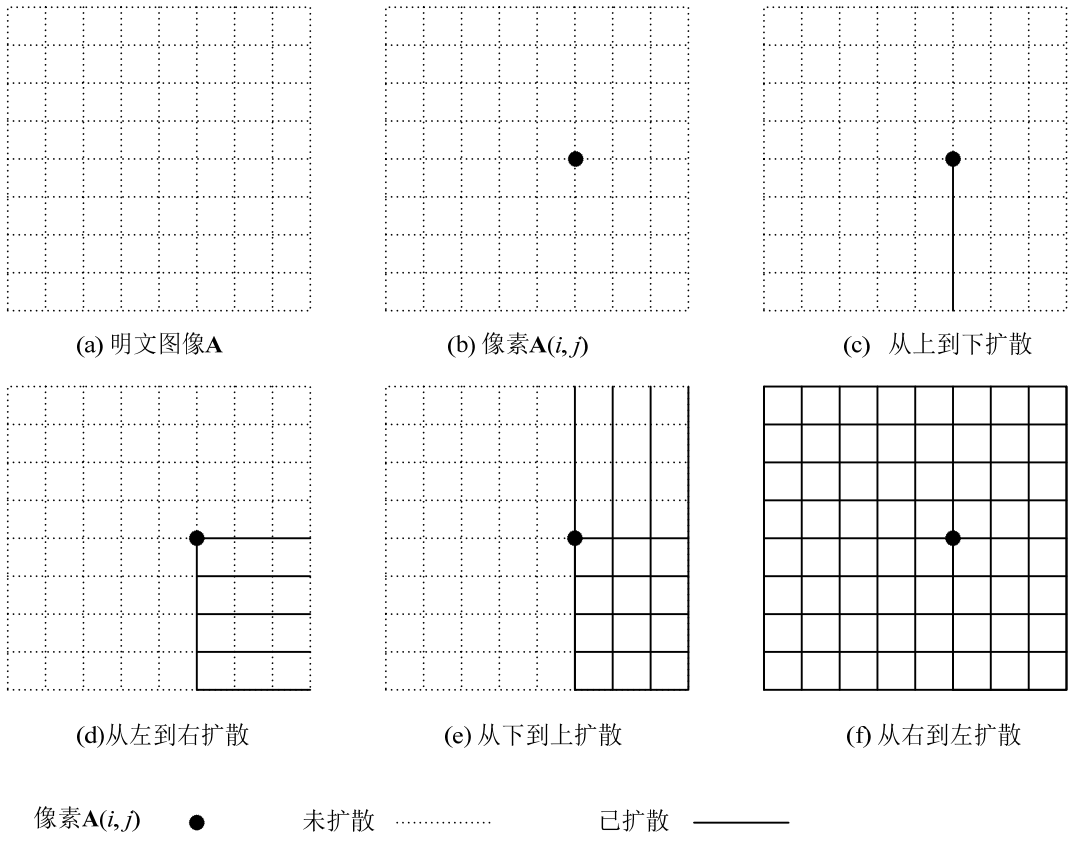


图3.5 CBLP算法的扩散过程

3.3.2 CPU 实现

图 3.6 是CBLP算法的加密过程框图，根据CBLP的设计原理和步骤，算法10给出了CBLP在CPU上的实现（即算法的串行实现）。

算法 10. CBLP 算法的 CPU 实现

输入：原始图像 **A**，$key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma_1\}$, $key_2=\{\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2\}$, M, N

输出：密图 $\mathbf{A}_c$

1. **for** $j \leftarrow 0$ **to** $N-1$ //子密钥生成
2. $(x, y) \leftarrow L^{2D}(x_0, y_0, \lambda_1, \lambda_2, \gamma_1, j + 100)$
3. $(\alpha, \beta) \leftarrow L^{2D}(\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2, j + 100)$
4. $(x', y') \leftarrow (x, y) \times 0.3 + 3.7$

5. $(\alpha', \beta') \leftarrow (\alpha, \beta) \times 0.3 + 3.7$
6. **for** $i \leftarrow 0$ **to** $M-1$
7. $r \leftarrow L^{1D}(x, x', 100+i) \times 10^{14} \bmod M$
8. $s \leftarrow L^{1D}(y, y', 100+i) \times 10^{14} \bmod N$
9. **if** $i == 0$ **then**
10. $p \leftarrow L^{1D}(\alpha, \alpha', r) \times 10^{14} \bmod 256$
11. $q \leftarrow L^{1D}(\beta, \beta, s) \times 10^{14} \bmod 256$
12. **else**
13. $p \leftarrow L^{1D}(p, \alpha', r) \times 10^{14} \bmod 256$
14. $q \leftarrow L^{1D}(q, \beta, s) \times 10^{14} \bmod 256$
15. **end**
16. $\mathbf{P}(i,j) \leftarrow p$; $\mathbf{Q}(i,j) \leftarrow q$
17. $\mathbf{R}(i,j) \leftarrow r$; $\mathbf{C}(i,j) \leftarrow c$
18. **end**
19. **end**
20. **for** $i \leftarrow 0$ **to** $M-1$ // 列置乱
21. **for** $j \leftarrow 0$ **to** $N-1$
22. $\mathbf{A}(i,j) \leftrightarrow \mathbf{A}(\mathbf{R}(i,j), j)$
23. **end**
24. **end**
25. **for** $i \leftarrow 0$ **to** $M-1$ // 行置乱
26. **for** $j \leftarrow 0$ **to** $N-1$
27. $\mathbf{A}(i,j) \leftrightarrow \mathbf{A}(i, \mathbf{C}(i,j))$
28. **end**
29. **end**
30. **for** $j \leftarrow 0$ **to** $N-1$ // 方向 1：从上到下
31. **for** $i \leftarrow 1$ **to** $M-1$
32. $\mathbf{A}(i,j) \leftarrow [\mathbf{A}(i,j) \oplus \mathbf{P}(i-1, j) \oplus \mathbf{A}(i-1, j)] <<< [\mathbf{P}(i,j) \bmod 8]$
33. **end**
34. **end**
35. **for** $j \leftarrow 0$ **to** $M-1$ //方向 2：从左到右
36. **for** $i \leftarrow 0$ **to** $N-1$
37. $\mathbf{A}(j,i) \leftarrow [\mathbf{A}(j,i) \oplus \mathbf{Q}(j, i-1) \oplus \mathbf{A}(j, i-1)] >>> [\mathbf{Q}(j,i) \bmod 8]$
38. **end**

39. **end**
40. **for** $j \leftarrow 0$ **to** $N-1$ // 方向 3：从下到上
41. **for** $i \leftarrow M-1$ **downto** 1
42. $\mathbf{A}(i-1,j) \leftarrow [\mathbf{A}(i-1,j) \oplus \mathbf{Q}(i,j) \oplus \mathbf{A}(i,j)]>>>[\mathbf{Q}(i-1,j) \bmod 8]$
43. **end**
44. **end**
45. **for** $j \leftarrow 0$ **to** $M-1$ // 方向 4：从右到左
46. **for** $i \leftarrow N-1$ **downto** 1
47. $\mathbf{A}(j,i-1) \leftarrow [\mathbf{A}(j,i-1) \oplus \mathbf{P}(j,i) \oplus \mathbf{A}(j,i)]<<<[\mathbf{P}(j,i-1) \bmod 8]$
48. **end**
49. **end**

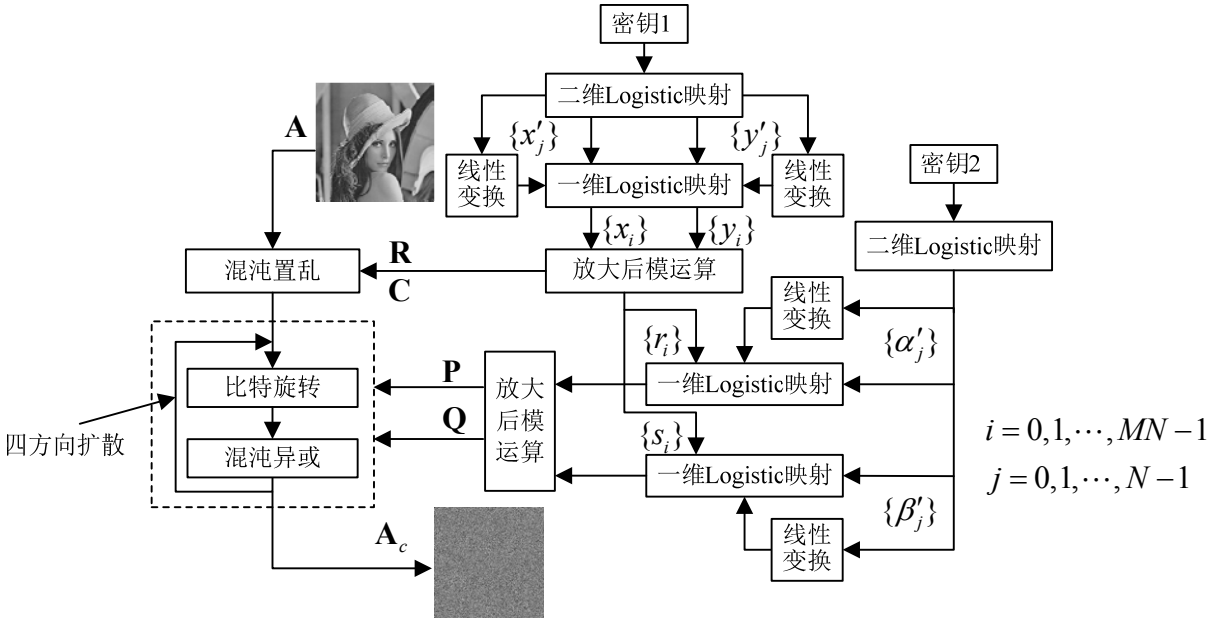


图3.6 CBLP算法的加密过程

3.3.3 GPU 实现

从前一节可知，CBLP 算法的计算开销主要集中在子密钥矩阵 **P** 和 **Q** 的生成以及图像的加密阶段。如果能对这两个阶段进行并行化实现，那么就可以在不降低安全性的前提下大幅度减少算法的执行时间。同时也注意到，CBLP 算法中的这两个阶段都具有高度的并行性，这为 CBLP 算法的并行化提供了可能性。

(1) 子密钥矩阵的并行化实现

矩阵 **P**、**Q** 和 **R**、**C** 可以用内核函数*__kernel void ChaosKeyMatGenPQRC*()生成，内核的伪代码如算法 11 所示。在算法 11 中，OpenCL 主机参数 *NDRange* 设定为 1 维，工作项和工作组参数分别设定为 *global_work_size*=[*N*, 0, 0]，*local_work_size*=[*N*, 0, 0]，所以在算法 11 中，

使用 OpenCL 内置函数 $j \leftarrow get_global_id(0)$后，$j = 0, 1, \ldots, N-1$。

算法 11. 内核*__kernel void ChaosKeyMatGenPQRC* ()

输入：密钥 $key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma_1\}$，$key_2=\{\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2\}$，$M$，$N$

输出：密钥矩阵 **P**、**Q** 和 **R**、**C**

1. $j \leftarrow get\_global\_id(0)$
2. $(x, y) \leftarrow L^{2D}(x_0, y_0, \lambda_1, \lambda_2, \gamma_1, j+100)$
3. $(\alpha, \beta) \leftarrow L^{2D}(\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2, j+100)$
4. $(x', y') \leftarrow (x, y) \times 0.3 + 3.7$
5. $(\alpha', \beta') \leftarrow (\alpha, \beta) \times 0.3 + 3.7$
6. **for** $i \leftarrow 0$ **to** $M-1$
7. $\quad r \leftarrow L^{1D}(x, x', 100 + i) \times 10^{14} \bmod M$
8. $\quad s \leftarrow L^{1D}(y, y', 100 + i) \times 10^{14} \bmod N$
9. $\quad$ **if** $i == 0$ **then**
10. $\quad\quad p \leftarrow L^{1D}(\alpha, \alpha', r) \times 10^{14} \bmod 256$
11. $\quad\quad q \leftarrow L^{1D}(\beta, \beta, s) \times 10^{14} \bmod 256$
12. $\quad$ **else**
13. $\quad\quad p \leftarrow L^{1D}(p, \alpha', r) \times 10^{14} \bmod 256$
14. $\quad\quad q \leftarrow L^{1D}(q, \beta, s) \times 10^{14} \bmod 256$
15. $\quad$ **end**
16. $\quad$ $\mathbf{P}(i, j) \leftarrow p$; $\mathbf{Q}(i, j) \leftarrow q$
17. $\quad$ $\mathbf{R}(i, j) \leftarrow r$; $\mathbf{C}(i, j) \leftarrow c$
18. **end**

(2) 图像加密的并行化实现

图像的加密操作是由内核函数*__kernel void ImageEncryption*()完成的，该内核函数非常简单，其具体实现伪代码如算法 12 所示。在该算法中主机参数 *NDRange* 设定为 1 维，工作项和工作组参数分别设定为 *global_work_size*=[N, 0, 0]，*local_work_size*=[N, 0, 0]，执行操作 $j \leftarrow get_global_id(0)$后，$j = 0, 1, \ldots, \Delta-1$，$\Delta=max(M, N)$。

算法 12. 内核*__kernel void ImageEncryption*()

输入：图像矩阵 **A**，密钥矩阵 **P**、**Q**、**R**、**C** 和 M、N

输出：密文图像 $\mathbf{A}_c$

1. $j \leftarrow get\_global\_id(0)$
2. **if** $j < N$ **then** // j 为列号，下同
3. $\quad$ **for** $i \leftarrow 0$ **to** $M-1$

```
4.          A(i, j) ↔ A(R(i, j), j)
5.      end
6.  end
7.  barrier (GLK_GLOBAL_MEM_FENCE) //同步，下同
8.  if j < M then   // j 为行号，下同
9.      for i ← 0 to N − 1
10.         A( j, i) ↔ A( j, C( j, i))
11.     end
12. end
13. barrier (GLK_GLOBAL_MEM_FENCE)
14. if j < N then
15.     for i ← 1 to M − 1
16.         A(i, j) ← [A(i, j) ⊕ P(i − 1, j) ⊕ A(i − 1, j)]<<<[P(i, j) mod 8]
17.     end
18. end
19. barrier (GLK_GLOBAL_MEM_FENCE)
20. if j < M then
21.     for i ← 1 to N − 1
22.         A( j, i)←[A( j, i) ⊕ Q( j, i − 1) ⊕ A( j, i − 1)]>>>[Q( j, i) mod 8]
23.     end
24. end
25. barrier (GLK_GLOBAL_MEM_FENCE)
26. if j < N then
27.     for i ← M − 1 downto 1
28.         A(i − 1, j) ← [A(i − 1, j) ⊕ Q(i, j) ⊕ A(i, j)]>>>[Q(i − 1, j) mod 8]
29.     end
30. end
31. barrier (GLK_GLOBAL_MEM_FENCE)
32. if j < M then
33.     for i ← N − 1 downto 1
34.         A( j, i − 1) ← [A( j, i − 1) ⊕ P( j, i) ⊕ A( j, i)]<<<[P( j, i − 1) mod 8]
35.     end
36. end
```

当建立好内核参数后，主机端代码将内核 *ImageEncryption*()放入命令队列等待执行，当

函数 *clFinish*()返回时，读取密文图像 $\mathbf{A}_c$ 到主机端即可完成图像的加密过程。解密过程的 OpenCL 实现与加密过程是类似的，这里不再赘述。

3.4 本章小结

本章首先介绍了一维和二维 Logistic 映射的基本定义，然后分别提出了两种新的并行图像加密算法——CSIM 和 CBLP。CSIM 算法包含位置置换和矩阵乘运算，位置置换完成图像的置乱功能，矩阵乘法完成图像的扩散功能。CBLP 算法包含比特旋转和异或运算，这两种运算均在像素的比特级上完成对图像的加密操作。CSIM 和 CBLP 算法都具有高度的并行性，为此我们使用 OpenCL 技术并行实现了两种算法并给出了详细的并行实现伪代码。

第 4 章 实验结果与算法分析

与所有优秀的加密算法都有针对此算法的攻击方法一样，使用本文提出的图像加密算法对图像加密后，也应针对现有的攻击方法做全面的安全性能分析以验证算法的安全性。下面分别从加密效率、密钥空间及敏感性、灰度直方图、信息熵、相邻像素相关性以及明文敏感性等几个方面来分析算法的安全性能[1]，并以此衡量算法的优劣。算法的硬件实验平台为一台联想E31工作站，使用的图形显示卡为NVIDIA GeForce GTX 580 GPU，具体参数如表4.1所示，算法实现的软件实验平台为MATLAB R2011b。实验所用的标准测试图像均来自于USC-SIP图像数据库[53]。

表4.1 硬件实验环境的参数配置

主机端参数	参数值	GPU 参数	参数值
CPU	Intel i5-3470	Compute Units	16
Clock Rate	3.2GHz	Global Memory	1503MB
RAM	4GB	Local Memory	48KB
GPU	GTX 580	Constant Buffer	64KB
OS	Windows 7，64-bit	Work-Item Sizes	1024×1024×64
OpenCL	1.0	Work-Group Sizes	1024

4.1 加密效率分析

为满足图像的实时传输，图像加密算法应该在保证传输安全性的同时具有足够低的处理时间。下面分别在CPU和GPU上实现本文提出的两个算法，即CSIM和CBLP算法，并给出算法在GPU上对CPU的加速比，作为比较我们也给出了Vihari等[50]提出的基于CUDA技术的并行图像加密算法。定义算法的加速比S为：$S = T_s / T_p$，其中T_s和T_p分别是算法在CPU和GPU上的执行时间。表4.2~4.4给出了加密不同尺寸的图像时的执行时间，从表中可以明显看出，无论是在CPU上还是在GPU上，当图像的尺寸越大时，算法消耗的时间就越多，但算法在GPU上对CPU的加速比也越大。也就是说，算法的OpenCL实现在加密大尺寸图像时比算法的CPU实现更加有效，这表明了使用OpenCL能够有效提高算法的执行效率。同时也能看出，CBLP算法较CSIM和Vihari算法更快速，这是因为CBLP算法的核心操作—比特旋转和异或—均是从像素的比特级上对图像进行加密的。CSIM算法虽然比Vihari的算法要慢，但是CSIM的结构复杂，安全性比Vihari算法要高，我们将在随后通过一系列安全性测试实验来证明。

[1] 若未特别指出，CSIM 算法的参数 *Rnd* 设置为 3。

表4.2 CSIM算法的执行效率分析

图像尺寸	时间（秒）		加速比 S
	CPU	GPU	
256×256	0.0428	0.0101	4.2364
512×512	0.2230	0.0278	8.0213
1024×1024	0.9461	0.0612	15.4589

表4.3 CBLP算法的执行效率分析

图像尺寸	时间（秒）		加速比 S
	CPU	GPU	
256×256	0.0368	0.0092	3.9964
512×512	0.0954	0.0109	8.7542
1024×1024	0.3840	0.0240	16.0014

表4.4 Vihari等算法的执行效率分析

图像尺寸	时间（秒）		加速比 S
	CPU	GPU	
256×256	0.0375	0.0081	4.6240
512×512	0.1357	0.0161	8.4274
1024×1024	0.5096	0.0340	14.9808

4.2 密钥空间及敏感性分析

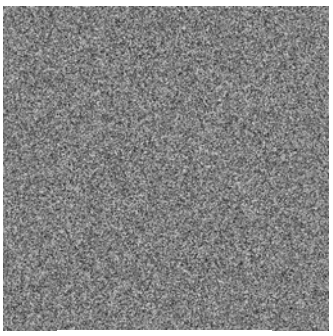
根据Kerckhoffs原则，一个密码算法的安全性与其对密钥的变化是否敏感以及密钥空间是否足够大有巨大的关系。只要加密算法具有巨大的密钥空间，就能抵挡任何穷举攻击。

CSIM算法的密钥*key*有两部分组成，即$key_{CSIM}=\{key_1, key_2\}$，其中$key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma\}$，$key_2=\{\alpha_0, \eta\}$。由于组成*key*的参数均是双精度浮点数，在计算机系统中可用8个字节表示，所以*key*中每个参数的密钥空间2^{32}，因此算法总的密钥空间大约为$(2^{32})^7$；CBLP算法的密钥*key*也有两部分组成，即$key_{CBLP}=\{key_1, key_2\}$，其中$key_1=\{x_0, y_0, \lambda_1, \lambda_2, \gamma_1\}$，$key_2=\{\alpha_0, \beta_0, \eta_1, \eta_2, \gamma_2\}$，算法总的密钥空间大约为$(2^{32})^{10}$，可见两种算法的密钥空间均不低于AES，足以抵挡任何蛮力攻击。而且，这两种算法的密钥空间均比Vihari算法的$(2^{32})^6$要大。

密钥敏感性是指密钥的微小改变（如1比特的改变）就能导致算法输出的巨大差别，显然一个安全的加密算法应对密钥的变化有足够的敏感性。以Lena图像（512×512）为例，密钥设置为key_{CSIM}={0.8765, 0.6123, 0.8, 0.2，0.3, 0.3456, 3.8765}，key_{CBLP}={0.8765, 0.6123, 0.8, 0.2, 0.3, 0.8765, 0.6123, 0.8, 0.2, 0.3}。图4.1和4.2分别给出了使用CSIM和CBLP算法加密Lena的实验结果，其中(a)为原始图像，(b)为密文图像，(c)是正确解密的图像，(d)~(l)是当密钥发生微小改

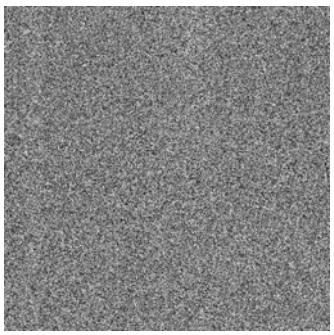
变时解密的图像。从实验结果可以看出，即使密钥发生微小改变也不能正确解密出原始图像，这说明CSIM和CBLP算法均具有很高的密钥敏感性。

(a)原始图像

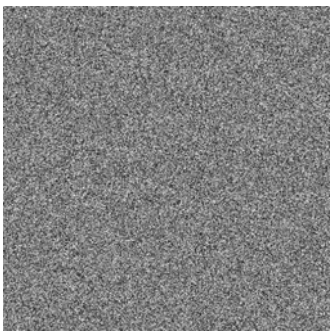


(b)加密图像

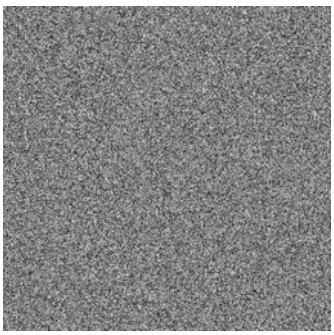
(c)正确解密图像



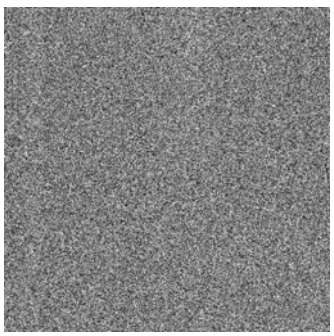
(d) x_0 减少 10^{-10}



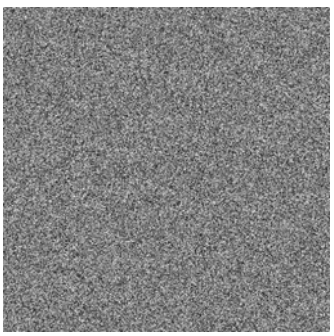
(e) γ 增加 10^{-10}



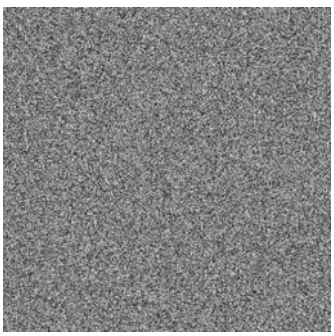
(f) y_0 增加 10^{-10}



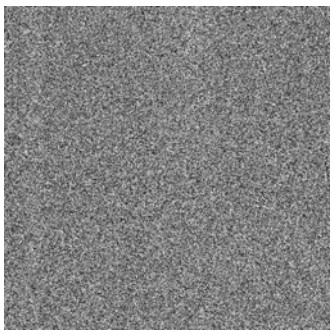
(g) α_0 减少 10^{-10}



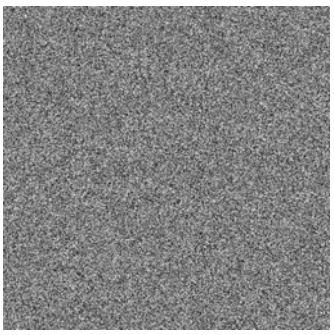
(h) η 增加 10^{-10}



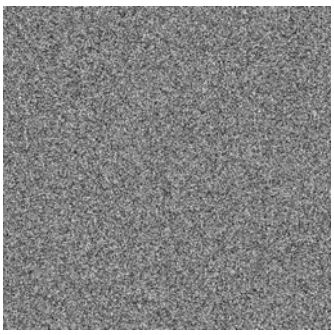
(i) γ 和 η 增加 10^{-10}



(j) α_0 减少 10^{-10} , y_0 增加 10^{-10}



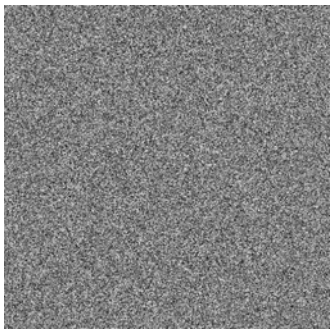
(k) x_0 和 y_0 增加 10^{-10}



(l) α_0 和 y_0 减少 10^{-10}

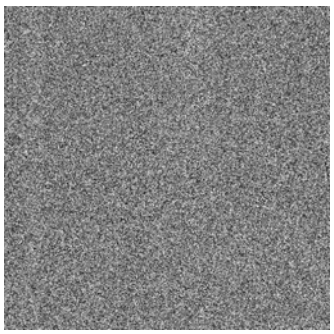
图4.1 CSIM算法的密钥敏感性测试

(a)原始图像

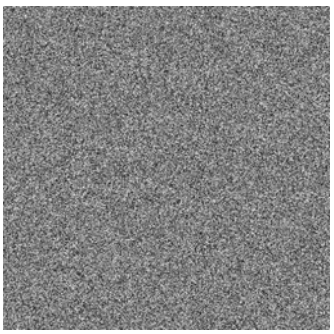


(b)加密图像

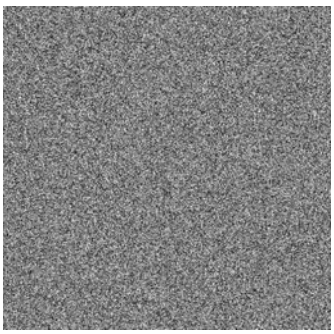
(c)正确解密图像



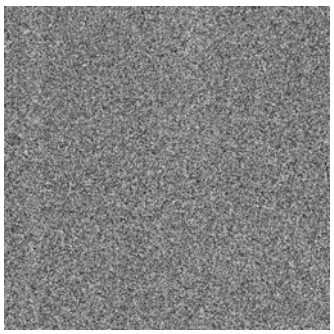
(d) x_0 减少 10^{-10}



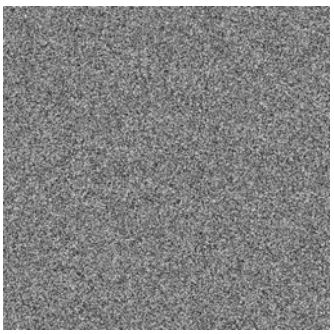
(e) y_0 增加 10^{-10}



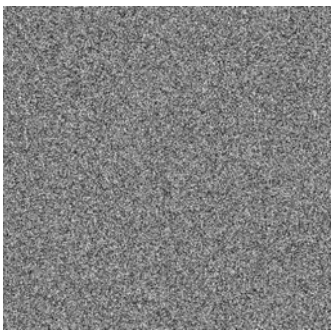
(e) γ_1 减少 10^{-10}



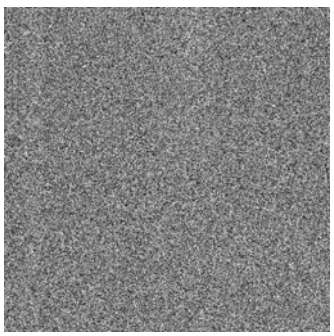
(g) α_0 增加 10^{-10}



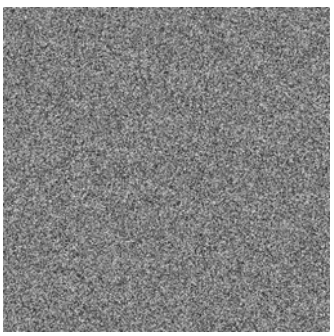
(h) β_0 减少 10^{-10}



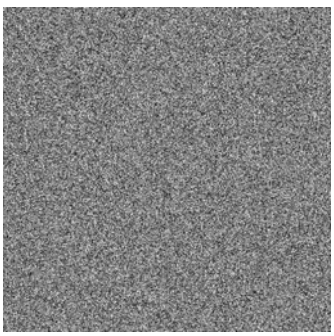
(i) γ_2 增加 10^{-10}



(j) β_0 减少 10^{-10}, y_0 增加 10^{-10}



(k) x_0 和 α_0 增加 10^{-10}



(l) γ_1 和 γ_2 减少 10^{-10}

图4.2 CBLP算法的密钥敏感性测试

4.3 直方图分析

图像的直方图[1]是图像的重要统计特征，它可以被认为是图像灰度密度函数的近似。一般来说，要精确得到图像的灰度密度函数是比较困难的，在实际应用中可以用数字图像灰度直方图来代替。灰度直方图是一个离散函数，它表示图像中具有某种灰度级的像素的个数，反映了图像中某种灰度出现的频率。在直方图中灰度级用横坐标表示，灰度级的像素个数或出现的频率用纵坐标表示。直方图的定义式如式(4.1)所示，其中N表示数字图像所有像素的个数，L为图像的灰度级，n_k表示第k个灰度级中的像素个数。

$$h_k = n_k / N, \quad k = 0, 1, 2, ..., L-1 \tag{4.1}$$

一个优秀的图像加密算法应使得加密后的密图在统计上不能提供任何有用的信息，比较理想的状态是加密算法将原始图像灰度值的不均匀分布变成灰度值的均匀分布；使密文的灰度值在整个灰度级范围内取值概率相等。这样一来，明文图像的统计特性被完全打破，明密文之间的相关性将大大降低。

图4.3是Lena图像（512×512）加密前后的直方图，从图中可以看出CSIM和CBLP算法均能使加密后的直方图灰度值分布更均匀，表明了两种算法都能有效地掩盖原始图像的统计特征，从而使得密图的灰度直方图不能提供任何有价值的信息。

4.4 信息熵分析

信息熵是衡量一个密码体制安全性的重要指标，信息熵的概念是由香农博士在1949年首先提出的。设s为信源，则s的熵$H(s)$定义为：

$$H(s) = -\sum_{i=0}^{2^{N_b}-1} p(s_i) \log_2 p(s_i) \tag{4.2}$$

式(4.2)中，N_b是表示一个像素所需要的比特个数，$p(s_i)$是二进制位s_i出现的概率。对于能发送2^8个符号的理想随机信源来说，$s=\{s_1, s_2, \ldots, s_{2^8}\}$，由式(4.2)可知其熵为8。所以一幅灰度级为256的密图的信息熵应该非常接近于8，否则说明密图还不够混乱，密图中还存在一定程度上的可预测性，大幅削弱了加密算法的安全性。表4.5~4.7给出了几种测试图像加密前后的信息熵，从表中可以看出本章所提出的算法使得加密后的图像信息熵非常接近于8，这说明了明文图像已被充分混乱，想从密图中获取任何有关明文的信息都是不可能的，所以CSIM和CBLP算法对抵抗基于信息熵的攻击具有足够的安全性。

表 4.5　几种测试图像的信息熵（CSIM 算法，图像尺寸：512×512）

原图	加密前	加密后	原图	加密前	加密后
Jetplane	6.7025	7.9996	Mandrill	7.2925	7.9998
Peppers	7.5937	7.9999	Pirate	7.2367	7.9991
Lena	7.4451	7.9991	Cameraman	7.0480	7.9999

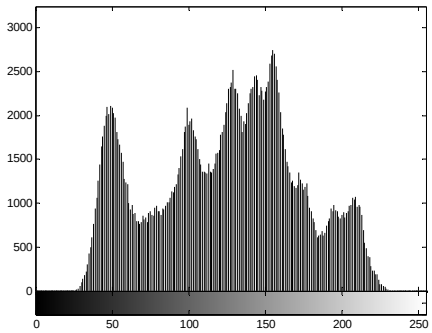
表 4.6　几种测试图像的信息熵（CBLP 算法，图像尺寸：512×512）

原图	加密前	加密后	原图	加密前	加密后
Jetplane	6.7025	7.9999	Mandrill	7.2925	7.9999
Peppers	7.5937	7.9999	Pirate	7.2367	7.9999
Lena	7.4451	7.9998	Cameraman	7.0480	7.9999

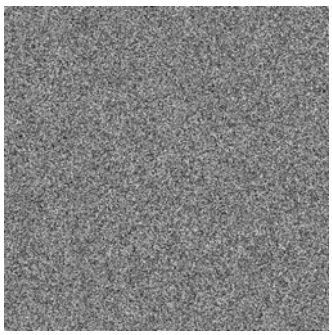
表 4.7　几种测试图像的信息熵（Vihari 算法，图像尺寸：512×512）

原图	加密前	加密后	原图	加密前	加密后
Jetplane	6.7025	7.9987	Mandrill	7.2925	7.9898
Peppers	7.5937	7.9901	Pirate	7.2367	7.9791
Lena	7.4451	7.9911	Cameraman	7.0480	7.9973

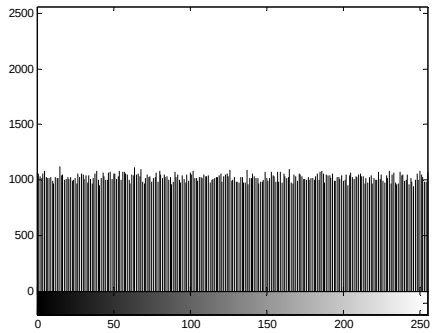
(a) 原始图像



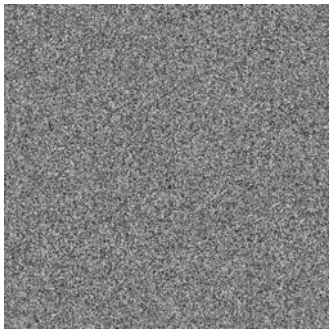
(b) 原始图像直方图



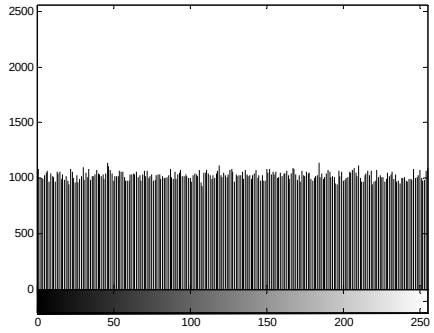
(c) CSIM 算法加密



(d) CSIM 算法直方图



(e) CBLP 算法加密



(f) CBLP 算法直方图

图4.3 CSIM和CBLP算法的灰度直方图分析

4.5 相邻像素相关性分析

相邻像素的相关性是评价加密算法优劣的又一个重要统计指标，它反映了图像置乱后的置乱效果。如果加密后图像的相邻像素相关性越小，则表明加密的置乱效果越好，反之则越差。相邻像素相关性的计算公式如下：

$$r_{xy} = \mathrm{cov}(x, y) / \sqrt{D(x)D(y)} \tag{4.3}$$

$$\mathrm{cov}(x, y) = \frac{1}{N}\sum_{i=1}^{N}[x_i - E(x)][y_i - E(y)] \tag{4.4}$$

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}x_i,\ D(x) = \frac{1}{N}\sum_{i=1}^{N}[x_i - E(x)]^2 \tag{4.5}$$

其中x, y表示图像中两相邻像素点的灰度值，$D(x)$表示所选像素灰度值的方差，$E(x)$表示所选像素灰度值的均值，$\mathrm{cov}(x, y)$和r_{xy}分别表示两相邻像素点的相关函数和相关系数。

通常原始明文图像的相邻像素是高度相关的，即相关系数接近于1。而比较理想的加密算法使得密图的相邻像素的相关系数接近于0，即相邻像素已基本不相关，说明明文的统计特征已被算法扩散到随机的密文中。

为了检验CSIM和CBLP算法的有效性，现随机从图像中选取3000对相邻像素（水平、垂直或对角方向），然后使用上述公式定量计算相邻像素的相关性。表4.8~4.10给出了Pirate、Peppers和Lena等三幅标准测试图像的明密文相邻像素的相关系数，图4.4给出了Lena图像在加密前后的相邻像素灰度值的分布情况。实验结果表明几种测试图像的相邻像素相关系数接近于1，而经过CSIM和CBLP算法加密后的它们的相邻像素相关系数已接近于0，这说明密图的相邻像素已基本不相关。另外，通过与Vihari 算法的对比可知，本文提出的两种算法均具有更低的相邻像素相关性。

表 4.8　Pirate 图像的相邻像素相关系数（图像尺寸：512×512）

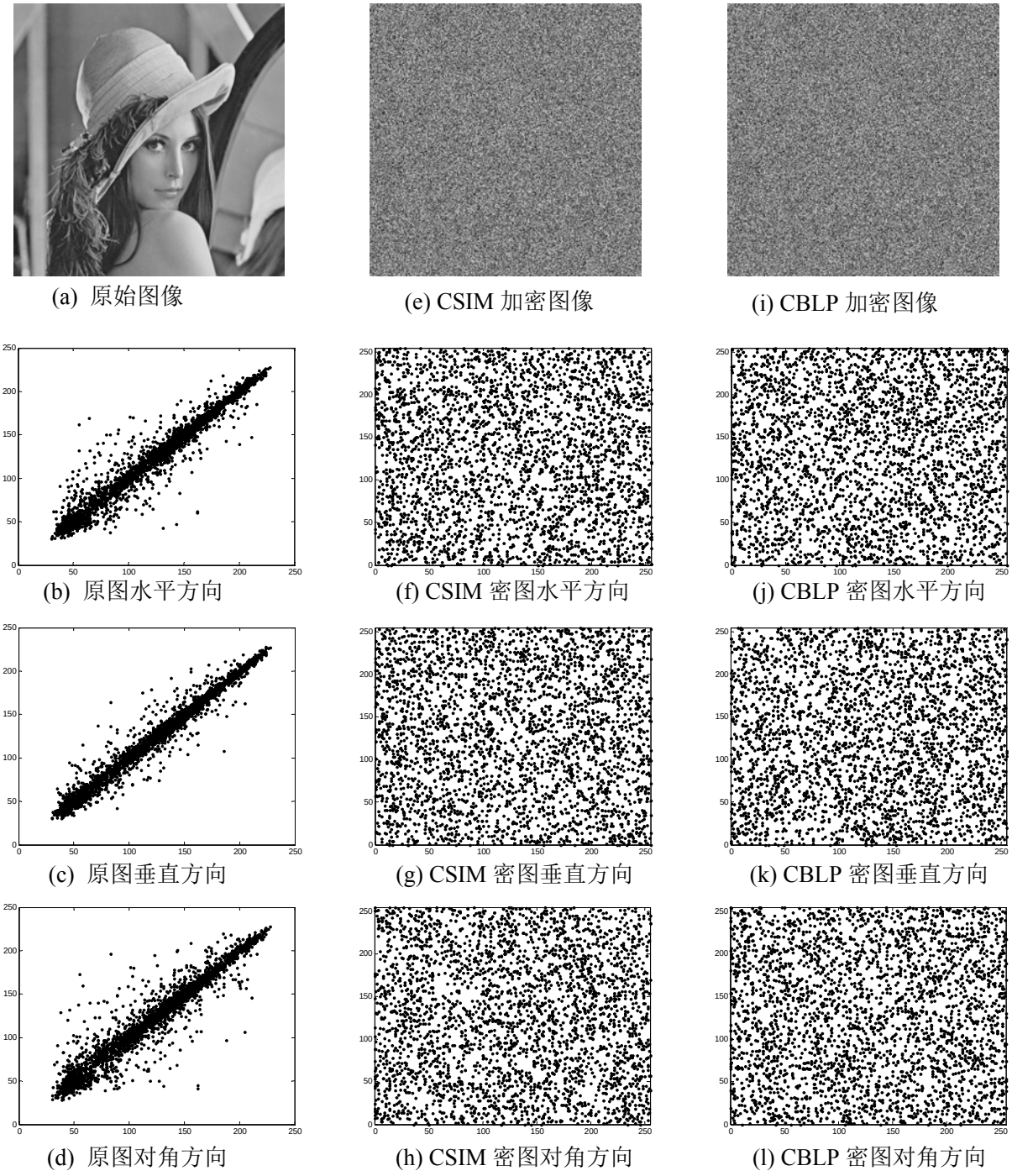
类型	原始图像	CSIM	CBLP	Vihari 算法
水平	0.9591	-0.0104	-0.0126	0.0227
垂直	-0.9651	0.0104	-0.0094	-0.0148
对角	-0.9403	-0.0119	-0.0108	-0.0198

表 4.9　Peppers 图像的相邻像素相关系数（图像尺寸：512×512）

类型	原始图像	CSIM	CBLP	Vihari 算法
水平	-0.9780	-0.0091	-0.0165	0.0400
垂直	0.9783	-0.0105	-0.0107	-0.0408
对角	0.9642	0.0096	0.0269	-0.0197

表 4.10　Lena 图像的相邻像素相关系数（图像尺寸：512×512）

类型	原始图像	CSIM	CBLP	Vihari 算法
水平	0.9716	0.0056	0.0156	-0.0170
垂直	0.9854	-0.0104	-0.0155	0.0203
对角	0.9590	0.0101	-0.0109	0.0510



(a) 原始图像　(e) CSIM 加密图像　(i) CBLP 加密图像

(b) 原图水平方向　(f) CSIM 密图水平方向　(j) CBLP 密图水平方向

(c) 原图垂直方向　(g) CSIM 密图垂直方向　(k) CBLP 密图垂直方向

(d) 原图对角方向　(h) CSIM 密图对角方向　(l) CBLP 密图对角方向

图4.4 Lena图像明密文相邻像素相关性分析

4.6 明文敏感性分析

在数字图像加密中，一般用平均强度变化率（Unified Average Changing Intensity, UACI）和像素数目改变率（Number of Pixels Change Rate, NPCR）来测试当明文或密钥发生微小改变时对密文的影响。UACI用来衡量算法抵抗差分攻击的能力，该指标的理想值为33.333…%，当UACI越接近理想值时，加密算法抵抗差分攻击的能力就越强；NPCR的理想值为100%，越接近这个值，表明密文对于明文变化越敏感，则加密算法抵抗已知明文攻击和选择明文攻击的能力越强。UACI和NPCR可以通过下面的公式计算：

$$\mathrm{UACI}=\frac{1}{MN}\left\{\sum_{ij}\frac{C_1(i,j)-C_2(i,j)}{255}\right\}\times 100\% \tag{4.6}$$

$$\mathrm{NPCR}=\frac{\sum_{ij}D(i,j)}{MN}\times 100\% \tag{4.7}$$

其中，M和N表示图像的行数和列数，C_1和C_2分别表示在明文图的一个像素改变前后的密文图，$C_1(i,j)$和$C_2(i,j)$分别表示C_1，C_2在(i,j)处的像素值。$D(i,j)$的值由下式决定：

$$D(i,j)=\begin{cases}1 & C_1(i,j)\neq C_2(i,j)\\ 0 & otherwise\end{cases} \tag{4.8}$$

使用CSIM和CBLP算法对几种不同尺寸的图像加密后的UACI和NPCR分数如表4.11所示。作为比较，我们分别给出了Vihari和Zhang等[36]的算法（Zhang的算法2）的UACI和NPCR分数。从表中可以看出，对明文的微小改变，CSIM和CBLP算法就能导致99%以上的密文发生变化，且密文的平均变化强度非常接近于理想值，这也说明了明文图像的信息已经较好地扩散到了密文图像中，即CSIM和CBLP算法均具有良好的明文敏感性，能够有效抵抗差分攻击。而在Vihari的算法中，由于算法的扩散操作只是独立地作用于每个像素，使得任意位置上的像素值的变化不能影响其他像素，所以算法的明文敏感性很差，从表中可以看出CSIM和CBLP算法的明文敏感性明显优于Vihari的算法，但和Zhang等人的算法相近。

表4.11　密文图像的UACI和NPCR测试

原图	UACI (%)				NPCR (%)			
	CSIM	CBLP	Vihari	Zhang	CSIM	CBLP	Vihari	Zhang
Jetplane	33.6199	33.4301	3.5×10^{-6}	33.5331	99.6378	99.6199	3.0×10^{-6}	99.1346
Peppers	33.4978	33.5024	9.1×10^{-6}	33.1604	99.6399	99.8474	1.1×10^{-6}	99.3541
Lena	33.4570	33.2814	1.9×10^{-6}	33.3824	99.5956	99.6534	2.1×10^{-6}	99.5661
Mandrill	33.4547	33.4836	3.2×10^{-6}	33.5816	99.6174	99.8956	1.2×10^{-5}	99.4812
Pirate	33.4498	33.2540	3.4×10^{-6}	33.8640	99.5977	99.7355	3.4×10^{-5}	99.6865
Cameraman	33.4636	33.3814	1.0×10^{-5}	33.1234	99.6245	99.7500	1.0×10^{-6}	99.3010

4.7 本章小结

为了分析本文提出的两个图像加密算法 CSIM 和 CBLP 的安全性和加密效率，本章进行了大量的仿真实验，并在加密效率、密钥空间及敏感性、直方图、信息熵、相邻像素相关性和明文敏感性等方面对两种并行图像加密算法进行了性能分析。分析结果表明，CSIM 和 CBLP 算法在加密性能上都是安全的，都可以有效抵抗穷举攻击、统计攻击、差分攻击以及已知/选择明文攻击等。另外，由于 CSIM 和 CBLP 算法所具有的高度并行性使得它们非常适合使用 GPGPU 进行加速，我们在 NVIDIA GeForce GTX580 GPU 上使用 OpenCL 技术实现了这两种算法，结果表明 OpenCL 能大幅提升算法的执行速度。因此，本文所提出的两个并行图像加密方案都是高效、安全的加密系统，都可以有效地应用于实际加密当中，具有较好的应用价值。

第 5 章 总结

图像加密是保护图像数据安全的一种直接而有效的重要手段，同时也是信息隐藏和数字水印等应用中不可或缺的一项重要技术。近年来，人们在图像加密方面进行了一些研究工作，取得了大量的研究成果。随着混沌理论的引入，数字图像加密技术得到了空前的蓬勃发展，各种各样的加密算法层出不穷。这些算法的出现极大的丰富了图像加密技术的内容，在保护数字多媒体安全方面体现了极大的应用价值。但是，当前的绝大多数图像加密算法都是基于 CPU 开发的，也就是说，这些算法没有考虑到图像加密的可并行性，只能在 CPU 上串行执行。这样导致的直接后果就是虽然开发出来的图像加密算法远比一些传统基于文本的加密算法要快，但是仍然不能满足人们在实时图像加密中对加密速度的要求。GPGPU 计算的出现为加速数值计算开辟了新的途径，利用 GPGPU 实现图像的实时加密已得到了研究者们的广泛关注。本文从混沌理论和 GPGPU 计算的应用出发，研究了基于 OpenCL 和混沌理论的图像加密技术，取得了一些研究成果。现将本文的主要研究工作总结如下。

(1) 介绍了图像加密算法的设计原则以及使用 OpenCL 作为实现 GPGPU 计算的意义。提出了将 OpenCL 技术应用于图像加密的思路和方法。

(2) 提出了两种新的基于 OpenCL 的并行图像加密算法——CSIM 和 CBLP。其中，CSIM 加密算法包含像素位置置换和图像矩阵乘法运算，位置置换用于完成图像的置乱，矩阵乘法用于完成图像的扩散。CBLP 算法包含像素位置置换、比特旋转和异或运算，位置置换用于置乱图像的像素位置，比特旋转和异或用于扩散图像的像素值。高度的并行性与安全性是 CSIM 和 CBLP 算法的共同特点，也是它们的最大特色。

(3) 通过大量的仿真实验，本文对所提出的两种并行图像加密算法在加密效率、密钥空间及敏感性、直方图、信息熵、相邻像素相关性和明文敏感性等方面进行了性能分析，验证了两种算法的安全性与加密效率。实验结果表明，CSIM 和 CBLP 算法在加密性能上都是安全的，都可以有效抵抗穷举攻击、统计攻击、差分攻击以及已知/选择明文攻击等。而且 CSIM 和 CBLP 算法所具有的高度并行性使得它们非常适合使用 GPGPU 进行加速，所以两种算法均能达到很高的加密效率。因此，本文所提出的两个图像加密方案都是高效、安全的加密系统，都可以有效地应用于实际加密当中，具有较好的应用价值。

到此，本文的研究工作也将告一段落，但随着互联网和多媒体技术的进一步发展，人们对实时图像加密的要求也越来越高，因此如何设计更安全、更快速的图像加密算法仍有许多工作要做。本文提出的两个基于 OpenCL 的并行图像加密算法为解决实时图像加密提供了一种可行的解决方案，同时我们也希望本文的研究工作能为 GPGPU 在数字图像加密技术领域的应用起到一定的推进作用。

致　谢

行文至此，我的这篇论文已接近尾声。在此我想要衷心的感谢在学习上曾给予我很多帮助的老师们。

首先，我要衷心的感谢我的导师游林教授，没有游老师的悉心指导，我不可能顺利完成研究生期间的科研任务和后续科研工作。游老师学识渊博、学术洞察力敏锐，论文的选题和思想都源于游老师。在完成论文的过程中，游老师更是在百忙之中牺牲很多休息时间，一遍又一遍地阅读我的论文并不厌其烦的修改我论文中的不足，提出了许多宝贵的意见和建议。这些都在学术上给予我莫大的帮助。游老师待人真诚、乐于奉献、信守承诺对学生就像对自己孩子一样关心和爱护，除了学术上悉心教导外在生活上游老师也给了我很大的帮助，他总是能设身处地的为我们的利益着想，想法设法解决我们生活上遇到的问题和困难。游老师人生阅历丰富，他的提携和指点让我明白很多为人处事的道理使我受益匪浅，游老师的这些优秀品格更会让我终生受益。在此我要再一次向游老师表示我最诚挚的谢意和最深的敬意。

同时，我要感谢通信学院的李光球教授、赵泽茂教授、方昕副教授等老师，他们的悉心授课和指导，使我学到了扎实的理论知识，为科研工作打下了坚实的基础。感谢本文参考文献中所有作者，他们的研究成果为本文提供了理论依据和丰富的素材。

再一次向导师和所有关心帮助过我的人表达我最诚挚的谢意！

参考文献

[1] 廖晓峰，肖迪, 陈勇, 向涛. 混沌密码学原理及其应用[M]. 北京: 科学出版社, 2009: 4-7.

[2] 郝柏林, 陈式刚, 郑伟谋. 从抛物线谈起:混沌动力学引论[M]. 上海: 上海科技教育出版社,1993: 1-15.

[3] 舒斯特. 混沌学引论[M]. 成都: 四川教育, 1994: 45-49.

[4] 陈式刚. 映像与混沌[M]. 南京: 南京大学出版杜, 1992: 990.

[5] Li T Y, Yorke J A. Period Three Implies Chaos[M]. New York: The Theory of Chaotic Attractors, 2004: 77-84.

[6] Luebke D, Harris M, Govindaraju N, et al. GPGPU: general-purpose computation on graphics hardware[C]. Proceedings of the 2006 ACM/IEEE conference on Supercomputing. ACM, 2006: 208-209.

[7] Eskicioglu A M, Delp E J. An overview of multimedia content protection in consumer electronics devices[J]. Signal Processing: Image Communication, 2001, 16(7): 681-699.

[8] 谭丽. 多媒体信息的混沌加密算法研究[D]. 长沙：长沙理工大学, 2011: 1-7.

[9] Devaney R L. An introduction to chaotic dynamical systems[M]. Reading: Addison-Wesley, 1989: 10-25.

[10] 齐东旭, 邹建成, 韩效宥. 一类新的置乱变换及其在图像信息隐蔽中的应用[J]. 中国科学: E 辑, 2000, 30(5): 440-447.

[11] 马在光, 丘水生. 基于广义猫映射的一种图像加密系统[J]. 通信学报, 2003, 24(2): 51-57.

[12] Chen G, Mao Y, Chui C K. A symmetric image encryption scheme based on 3D chaotic cat maps [J]. Chaos, Solitons & Fractals, 2004, 21(3): 749-761.

[13] Wang X, Jin C. Image encryption using Game of Life permutation and PWLCM chaotic system[J]. Optics Communications, 2012, 285(4): 412-417.

[14] Parberry I. An efficient algorithm for the knight's tour problem[J]. Discrete Applied Mathematics, 1997, 73(3): 251-260.

[15] Delei J, Sen B, Wenming D. An Image Encryption Algorithm Based on Knight's Tour and Slip Encryption-filter[C].Computer Science and Software Engineering, 2008 International Conference on. IEEE, 2008, 1: 251-255.

[16] Thanikaiselvan V, Arulmozhivarman P, Amirtharajan R, et al. Horse riding & hiding in image for data guarding[J]. Procedia Engineering, 2012, 30(Complete): 36-44.

[17] 林雪辉, 蔡利栋. 基于 Hilbert 曲线的数字图像置乱方法研究 [J]. 中国体视学与图像分析, 2004, 9(4): 224-227.

[18] Fridrich J. Symmetric ciphers based on two-dimensional chaotic maps[J]. International Journal of Bifurcation and Chaos, 1998, 8(06): 1259-1284.

[19] Lian S, Sun J, Wang Z. A block cipher based on a suitable use of the chaotic standard map[J]. Chaos,

Solitons & Fractals, 2005, 26(1): 117-129.

[20] Wong K W, Kwok B S H, Law W S. A fast image encryption scheme based on chaotic standard map[J]. Physics Letters A, 2008, 372(15): 2645-2652.

[21] Tong X, Cui M. Image encryption with compound chaotic sequence cipher shifting dynamically[J]. Image and Vision Computing, 2008, 26(6): 843-850.

[22] Behnia S, Akhshani A, Mahmodi H, et al. A novel algorithm for image encryption based on mixture of chaotic maps[J]. Chaos, Solitons & Fractals, 2008, 35(2): 408-419.

[23] Behnia S, Akhshani A, Akhavan A, et al. Applications of tripled chaotic maps in cryptography [J]. Chaos, Solitons & Fractals, 2009, 40(1): 505-519.

[24] Behnia S, Akhshani A, Ahadpour S, et al. A fast chaotic encryption scheme based on piecewise nonlinear chaotic maps [J]. Physics Letters A, 2007, 366(4): 391-396.

[25] Wang Y, Wong K W, Liao X, et al. A block cipher with dynamic S-boxes based on tent map[J]. Communications in Nonlinear Science and Numerical Simulation, 2009, 14(7): 3089-3099.

[26] Gao T, Chen Z, Yuan Z, et al. A hyperchaos generated from Chen's system[J]. International Journal of Modern Physics C, 2006, 17(04): 471-478.

[27] Kwok H S, Tang W K S. A fast image encryption system based on chaotic maps with finite precision representation[J]. Chaos, solitons & fractals, 2007, 32(4): 1518-1529.

[28] Gao T, Chen Z. Image encryption based on a new total shuffling algorithm[J]. Chaos, Solitons & Fractals, 2008, 38(1): 213-220.

[29] Gao T, Chen Z. A new image encryption algorithm based on hyper-chaos[J]. Physics Letters A, 2008, 372(4): 394-400.

[30] Arroyo D, Li C, Li S, et al. Cryptanalysis of an image encryption scheme based on a new total shuffling algorithm [J]. Chaos, Solitons & Fractals, 2009, 41(5): 2613-2616.

[31] 许克兵,黄文培.一种基于 Baker 映射的二维混沌图像加密[J].微计算机信息,2008,12(1): 59-61.

[32] Sun F Y, Liu S T, et al. Image encryption using high-dimension chaotic system[J]. Chinese Physics, 2007, 16(12):3616-3623.

[33] 赵亮, 廖晓峰, 向涛等. 对高维混沌系统的图像加密算法安全性和效率的改进[J]. 计算机应用, 2009, 27(7): 1775-1781.

[34] 许冰, 孙永维, 李洋. 基于高维混沌系统的图像加密改进算法[J]. 吉林大学学报:信息科学版, 2012, 30(1): 12-17.

[35] Wu Y, Yang G, Jin H, et al. Image encryption using the two-dimensional logistic chaotic map[J]. Journal of Electronic Imaging, 2012, 21(1): 013014-1-013014-15.

[36] Zhang W, Wong K, Yu H, et al. An image encryption scheme using reverse 2-dimensional chaotic map and dependent diffusion[J]. Communications in Nonlinear Science and Numerical Simulation, 2013, 18(8): 2066-2080.

[37] Proudfoot K, Mark W R, Tzvetkov S, et al. A real-time procedural shading system for programmable graphics hardware[C]. Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, 2001: 159-170.

[38] Buck I, Foley T, Horn D, et al. Brook for GPUs: stream computing on graphics hardware[C]. ACM Transactions on Graphics (TOG). ACM, 2004: 777-786.

[39] Rost R J. OpenGL: Shading Language[M]. Reading: Addison-Wesley Professional, 2004:1-100.

[40] Peeper C, Mitchell J L. Introduction to the DirectX® 9 high level shading language[J]. ShaderX2: Introduction and Tutorials with DirectX, 2003, 9:1.

[41] Mark W R, Glanville R S, Akeley K, et al. Cg: a system for programming graphics hardware in a C-like language[C]. ACM Transactions on Graphics (TOG). ACM, 2003: 896-907.

[42] CUDA, http://www.nvidia.com/object/cuda_home_new.html, 2013-10-01.

[43] OpenCL, http://www.khronos.org/opencl, 2013-12-25.

[44] DirectCompute, http://www. msdn.microsoft.com/zh-cn/directx, 2012-05-15.

[45] Munshi A, Gaster B, Mattson T G, et al. OpenCL programming guide [M]. Boston: Pearson Education, 2011.

[46] 石林. GPU 通用计算虚拟化方法研究[D]. 长沙: 湖南大学, 2012: 1-7.

[47] 石飞. 基于 OpenCL 的 OpenSSL-AES 加密算法优化[D]. 吉林: 吉林大学, 2012:1-3.

[48] Wang W, Wang X, Song D. A parallel chaotic cryptosystem for dual-core processor[C]. Information Science and Engineering (ICISE), 2010 2nd International Conference on. IEEE, 2010: 920-923.

[49] 孙冰. 基于 GPGPU 的高效 AES 彩色数字图像加密技术的研究与实现[D]. 成都: 电子科技大学, 2010: 1-4.

[50] Vihari P L V, Mishra M. Chaotic image encryption on GPU[C]. Proceedings of the CUBE International Information Technology Conference. ACM, 2012: 753-758.

[51] Du P, Weber R, Luszczek P, et al. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming[J]. Parallel Computing, 2012, 38(8): 391-407.

[52] Acharya B, Patra S K, Panda G. Image encryption by novel cryptosystem using matrix transformation[C]. Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on. IEEE, 2008: 77-81.

[53] USC-SIPI image database, http://www.sipi.usc.edu/database/, 2013-12-25.

[54] Shannon C E. Communication theory of secrecy systems[J]. Bell system technical journal, 1949, 28(4): 656-715.

[55] 李辉. 超宽带混沌通信技术的研究[D]. 南京: 南京理工大学, 2008: 1-3.

[56] 王育民,刘建伟. 通信网的安全: 理论与技术[M]. 西安: 西安电子科技大学出版社, 1999:1-10.

[57] Gaster B, Howes L, Kaeli D R, et al. Heterogeneous Computing with OpenCL: Revised OpenCL 1[M]. Boston: Newnes, 2012: 1-40.

[58] 孙冰. 基于 GPGPU 的高效 AES 彩色数字图像加密技术的研究与实现[D]. 成都: 电子科技大学,

2010:1-10.

[59] 刘家胜. 基于混沌的图像加密技术研究 [D]. 合肥: 安徽大学, 2007:1-6.

[60] 黄金. 安全混沌图像加密算法的研究[D]. 长沙: 长沙理工大学, 2010:1-2.

[61] 任洪娥, 戴琳琳, 张健. 基于位平面变换的数字图像加密算法[J]. 计算机工程, 2013, 39(6): 185-189.

[62] 张云鹏, 左飞, 翟正军. 基于混沌的数字图像加密综述[J]. 计算机工程与设计, 2011, 32(2): 463-466.

[63] Stone J E, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems[J]. Computing in science & engineering, 2010, 12(3): 66.

[64] Fang J, Varbanescu A L, Sips H. A comprehensive performance comparison of CUDA and OpenCL[C]. Parallel Processing (ICPP), 2011 International Conference on. IEEE, 2011: 216-225.

附　录

作者在读期间发表的学术论文及参加的科研项目

发表的学术论文：

[1] You L, Yang Y, Wen W. Formulations of some bit switching functions in DES[J]. Wuhan University Journal of Natural Sciences, 2013, 18(5): 402-406.

[2] 温万里，游林. 基于混沌和比特级置乱的并行图像加密算法[J].信息网络安全（已录用）.

参加的科研项目：

[1] 基于指纹特征及亏格 2 超椭圆曲线上的密码技术. 钱江人才计划.　2013.07-2015.06.

[2] 几类非椭圆代数曲线密码体制的关键运算研究. 信息安全国家重点实验室开放课题. 2012.01-2013.12.