XXXX

基于GPU的高速载波恢复算法设计与实现

高明，杨合天，付少忠
（西安电子科技大学通信工程学院，陕西 西安 710071）

摘　要：针对无线通信系统中载波恢复算法复杂度高、在CPU（Central Processing Unit）中运行时实时性差的问题，提出了一种基于GPU（Graphics Processing Unit）的并行高速载波恢复算法。在无线通信系统中，基于FFT变换的粗频偏估计算法结合基于判决引导锁相环（PLL，Phase Locked Loop）的细频偏估计算法是一种典型的载波恢复算法。然而，考虑到PLL中的反馈结构并不适合在GPU中并行运行，提出使用扫频算法代替判决引导的锁相环算法，通过利用GPU的强大并行运行能力提高载波恢复算法的运行速度。在此基础上，基于计算统一设备架构（CUDA，Compute Unified Device Architecture）平台，通过对扫频算法和其在CUDA中采用的并行归约算法进行优化，进一步提高了载波恢复算法的运行速度。仿真结果表明，所提出的基于GPU的并行载波恢复算法在吞吐量和算法性能方面均优于基于CPU的串行载波恢复算法，对于不同的调制方式，其吞吐量可提高20至40多倍，因此更适用于高速场景下的无线通信系统。

关键词：载波恢复；并行计算；图形处理器；计算统一设备架构
中图分类号：
文献标志码：A
doi: 10.11959/j.issn.1000−0801.

GPU-Accelerated Design and Implementation of a High-Speed Carrier Recovery Algorithm

GAO Ming, YANG Hetian, FU Shaozhong
School of Communication Engineering, Xidian University, Xi 'an 710071, China

Abstract: To address the high complexity of carrier recovery algorithms in wireless communication systems and their poor real-time performance when executed on CPUs (Central Processing Units), a parallel high-speed carrier recovery algorithm based on GPU (Graphics Processing Unit) is proposed. In wireless communication systems, the combination of a coarse frequency offset estimation algorithm based on FFT and a fine frequency offset estimation algorithm based on a decision-directed phase-locked loop (PLL) constitutes a typical carrier recovery approach. However, considering that the feedback structure inherent in the PLL is not well suited for parallel execution on GPUs, a frequency sweeping algorithm is proposed to replace the decision-directed PLL algorithm, thereby leveraging the powerful paral-

收稿日期：XXXX−XX−XX；修回日期：XXXX−XX−XX
通信作者：高明，邮箱：mgao@mail.xidian.edu.cn
基金项目：国家自然科学基金资助项目（No.62171354）
Foundation Items: The National Natural Science Foundation of China（No. 62171354）

lel processing capability of GPUs to accelerate the carrier recovery process. On this basis, within the Compute Unified Device Architecture (CUDA) platform, the execution speed of the carrier recovery algorithm is further improved by optimizing the frequency sweeping algorithm and its associated parallel reduction algorithm implemented in CUDA. Simulation results demonstrate that the proposed GPU-based parallel carrier recovery algorithm outperforms the CPU-based serial carrier recovery algorithm in terms of both throughput and algorithmic performance. For different modulation schemes, the throughput can be improved by 20 to over 40 times, making it more suitable for wireless communication systems in high-speed scenarios.

Key words: Carrier Recovery, Parallel Computing, Graphics Processing Unit, Compute Unified Device Architecture

1 引言

随着无线通信技术的不断演进，特别是在高吞吐量、低时延等新型业务需求驱动下，通信系统对解调速率与实时性的要求日益提高[1]。近年来，图形处理单元（GPU，Graphics Processing Unit）凭借其强大的并行计算能力，已被广泛应用于高性能计算领域，成为软件定义无线电（SDR，Software Defined Radio）系统的重要支撑平台[2]。目前基于NVIDIA推出的GPU通用并行计算平台CUDA（Compute Unified Device Architecture），能够很好的对通信信号处理速度进行优化，从而大幅提高软件无线电系统的运行速度，使得其更适用于需要实时处理的通信系统。

因此，为了满足实际应用对无线通信系统提出的高速率、高可靠、低延迟、多用户接入等要求，基于GPU的具备高吞吐量和强大实时处理能力的无线通信系统受到了广泛关注。文献[3]研究了一种细粒度软件无线电MIMO-OFDM通信系统算法，利用GPU构建了实时通信系统，可使系统传输速率提升10%到30%。文献[4]通过在通信系统中增加数据流基础架构和独立的GPU加速库，能够进一步提高基于GPU加速应用程序的运行速度并优化内存空间。为实现通信系统中各处理模块间吞吐率的均衡，文献[5]探索了将算法抽象为可伸缩的向量化操作。该方法充分利用GPU的向量化并行计算能力，并依据模块的实时处理负荷，动态适配向量化处理的规模，最终达成系统整体吞吐率的优化配置。文献[6]提出了一种基于GPU的CCSDS 131.2-B接收机实现方案，实验验证其在ACM 1-7模式下可稳定维持10MBaud的吞吐量。

在无线通信系统的接收机中，载波恢复模块复杂度较高，对接收机的处理速度有着重要影响，因此可使用GPU技术对其进行加速，从而提高整个系统的运行速度。在单载波无线通信系统中，在时域上主要包括：Kay频偏估计算法[7]、Fitz频偏估计算法[8]、M&M频偏估计算法[9]、L&R频偏估计算法[10]等；频域可采用FFT频偏估计算法[11]。由于上述算法的频偏估计精度有限，一般在使用这些算法后，还需要进一步使用判决引导(DD，Decision-Direct)[12]等的判决反馈算法进行载波跟踪消除剩余频偏。文献[13]针对大范围多普勒频偏处理，设计了基于FFT与PLL的频偏估计架构，用于在补偿多普勒频偏的同时，也可通过精确估计来校正信号的未知相位，该算法的思想已成为目前载波恢复的主要解决方案之一。在利用GPU对载波恢复模块进行加速的研究中，文献[14]研究了基于GPU并行化的光学相干接收机载波相位恢复算法，提出决策辅助相位解缠机制，有效解决传统Viterbi-Viterbi算法中的串行瓶颈，进一步提高了系统的吞吐量。考虑到载波恢复算法对软件无线电系统的实时性影响较大，本文在研究了典型的FFT和PLL载波恢复算法的基础上，提出了一种基于GPU加速的分阶式载波恢复算法，解决了抗大频偏和细小频偏累积

问题，并在CUDA编程平台上进行了优化，从而提升了载波恢复算法的实时性和有效性。

2　载波恢复算法原理

在无线通信系统中，载波恢复技术是接收机的关键技术之一，其主要作用是在接收到的信号中准确恢复载波相位，以便进行正确的解调。载波恢复过程通常分为粗同步和细同步两个阶段，如图1所示。

图1中的FFT载波恢复算法中，其输入输入信号可表示为：

$$r(k)=c_k e^{j(2\pi\Delta f T_s+\varphi)}+n(k) \tag{1}$$

在上式中，c_k为调制后的符号，Δf为收发载波之间的频率偏移，T_s为符号周期，$n(k)$为均值为零、方差为σ^2的加性复高斯白噪声(AWGM，Additive White Gaussian Noise)，φ为收发载波的初始相位偏移，服从0到2π的均匀分布。

在频偏估计前，首先去除基带信号中的调制信息[15]，得到只含有加性噪声的单载波信号，如下式所示：

$$r_k'=r_k e^{-j\varphi_n}=e^{j(2\pi\Delta f T_s+\theta)}+n_k' \tag{2}$$

其中，r_k'为去除调制信息的符号字节序列，n_k'为等效噪声。

设用于频偏估计的序列长度为N，则其频谱可以通过执行N点傅里叶变换得到：

$$F_k=\sum_{n=0}^{N-1} r_k' e^{-j\frac{2\pi}{N}kn},\ k=0,1,...,N-1 \tag{3}$$

为了得到功率谱密度，需要将傅里叶变换的结果归一化：

$$P_k=\frac{|F_k|^2}{N},\ k=0,1,...,N-1 \tag{4}$$

记P_k最大值对应的索引为$k_{\max}$。则通过FFT频偏估计算法得到估计的频偏为：

$$\Delta\hat{f}_{FFT}=\begin{cases}\dfrac{k_{\max}}{TN}, & 0\le k_{\max}\le \dfrac{N}{2}-1\\ -\dfrac{N-k_{\max}}{TN}, & \dfrac{N}{2}\le k_{\max}\le N-1\end{cases} \tag{5}$$

在粗同步阶段对频偏进行估计并补偿后，一般还需通过细同步完成对剩余频偏的估计与补偿，目前常使用的方法为基于锁相环路(Phase-Locked Loop, PLL)的判决反馈算法[15]，如DD算法[16]，并由此构成了FFT + PLL联合载波恢复算法[17]。对于PLL算法实现结构由乘法器、判决器(Detector)、鉴相器(Phase Detector，PD)、环路滤波器(Loop Filter，LF)和数控振荡器(Numerically Controlled Oscillator，NCO)组成，如图2所示。

尽管闭环结构的PLL算法同步精度高、跟踪能力强，但其存在两个主要缺点：一是需要一定的初始数据和较长的收敛时间才能达到精确同步，收敛速度受估计参数偏差和环路带宽制约，难以满足快速同步需求；二是由于其内部存在反馈机制，状态更新严格依赖前一时刻输出，具有强烈的时序依赖性，因此必须逐符号串行处理，这使得该算法无法并行化，从而极大限制了其在GPU并行平台上的计算效率。

为了选择更适合并行运行的载波恢复算法来代替PLL算法，本文引入扫频算法。扫频是最基本的载波恢复算法，由于其复杂度与扫频的频点密切相关，当使用CPU系统进行串行计算时，计算时延将随着频点的个数增加而快速增加，因此并不适用于CPU系统。而对于GPU系统，由于其强大的并行计算能力，可以将每个频点并行计算，从而大幅提升载波恢复算法的运行速度，使

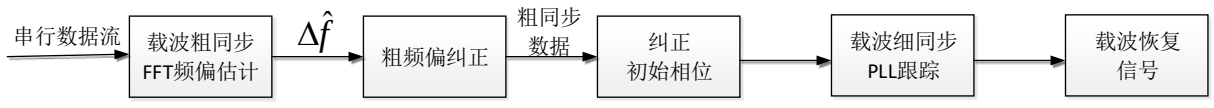

图1　载波恢复流程结构图

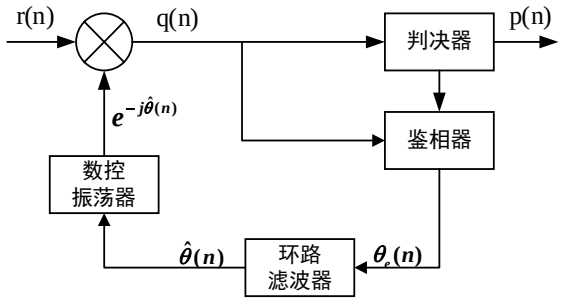

图2　PLL算法流程框图

得其可适用于GPU系统。采用扫频算法代替PLL算法后的GPU载波恢复算法如图3所示。

扫频算法的基本思想是以一定的频率间隔对信号进行频偏补偿，然后通过对所有补偿后的信号计算误差矢量幅度(EVM，Error Vector Magnitude)，选择EVM最小的频点作为最终的频偏估计值。

我们将载波粗同步后的信号序列记为$r_{coarse}[n]$，粗同步后的最大残留频偏记为$f_{\max}$，从而扫频范围为$[-f_{\max}, f_{\max}]$。记扫频步长为Δf_{step}，则候选频点的数目为：

$$m = 2\left\lfloor \frac{f_{\max}}{\Delta f_{step}} \right\rfloor + 1 \tag{6}$$

其中，$\lfloor\cdot\rfloor$表示向下取整，这意味着扫描频点数目m由扫频的最大频偏和步长精度确定。

扫频过程在规定范围内以固定步长进行，候选频点可通过如下公式表示：

$$\Delta f_k \in \left[-\left(\frac{m-1}{2}\right)\cdot\Delta f_{step}, \left(\frac{m-1}{2}\right)\cdot\Delta f_{step} \right] \tag{7}$$

其中，Δf_k为第k个候选频偏，m为扫频总点数，以$\Delta f=0$为扫频中心。

对于每个候选频点Δf_k，通过对信号进行频偏纠正和相位补偿，计算EVM以衡量信号的恢复质量，EVM计算公式如下：

$$\begin{aligned} EVM(\Delta f_k) &= \sum_{k=1}^{N} evm(k) \\ &= \sum_{k=1}^{N} \frac{\sqrt{(I(k)-I(k)_{ref})^2+(Q(k)-Q(k)_{ref})^2}}{\sqrt{I(k)_{ref}^2+Q(k)_{ref}^2}} \end{aligned} \tag{8}$$

其中，evm(k)为每个纠偏数据的计算结果，$I(k)$与$Q(k)$分别表示解调后的接收信号的同相与正交分量，$I_{ref}(k)$与$Q_{ref}(k)$分别为理想参考星座点的同相与正交分量。

在得到所有的$\left(EVM(\Delta f_k)\right)$后，选择最小EVM值对应的频点作为最佳频偏值：

$$\Delta f_{best} = \min_{\Delta f_k}\left(EVM(\Delta f_k)\right) \tag{9}$$

由此得到了信号细同步输出$r_{fine}[n]$为：

$$r_{fine}[n] = r_{coarse}[n]\cdot e^{-j2\pi\Delta f_{best}\Delta t}\cdot e^{-j\theta} \tag{10}$$

以上为本文采用的载波恢复算法的基本原理，下面对本文提出的分阶式FFT与扫频联合载波恢复算法的GPU实现与优化进行详细介绍。

3　GPU扫频算法设计及优化

3.1　载波恢复GPU算法设计

在本文采用的FFT+扫频载波恢复算法中，FFT运算可以直接调用GPU算法库函数来实现加速运算。例如，CUDA平台提供的cuFFT库可支持高性能的多维FFT并行计算。在FFT频偏估计后，可以得到一个归一化频偏值$\Delta\hat{f}_{FFT}$，然后用其对数据进行并行粗频偏纠正。完成粗频偏纠正后，需进一步对剩余频偏进行细频偏纠正，此时通过扫频算法进行。基于GPU的扫频算法流程如图4所示。

首先，需要设置扫频范围和步长精度。扫频

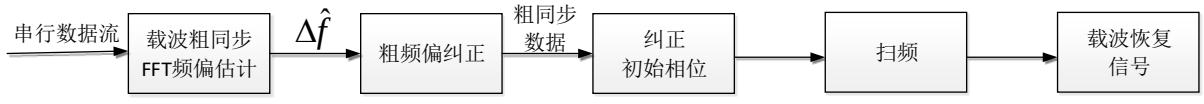

图3　基于GPU的FFT+扫频载波恢复算法流程图

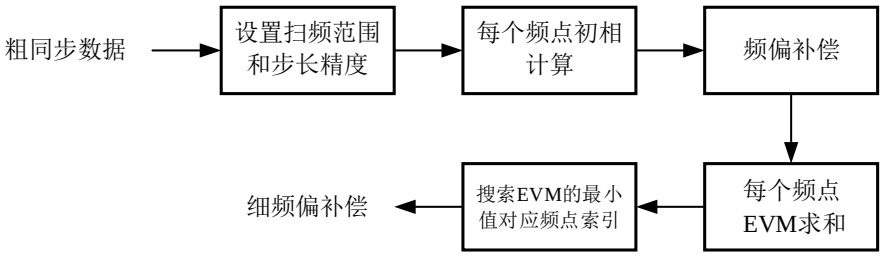

图4　基于GPU的扫频算法设计流程

范围由粗频偏估计的精度决定，步长精度则决定最终的细频偏估计精度。在扫频开始之前，需先估计初始相偏，供后续每个频点进行初相补偿时使用。其次，对于每个频点进行并行计算，完成相位和候选频点的补偿。由于每个频点下的初始相位计算与相位频偏补偿仅与当前频偏数值有关，因此在GPU实现中可以将每个频点下对应一个线程，完成各频点下的初相计算操作，具体流程如图5所示。

在上图中，每个线程处理一个扫频频点，通过这种线程级的并行设计，可以充分发挥GPU在处理大量独立、重复任务方面的优势，显著加速所有频偏候选点下的相位估计和预补偿过程。

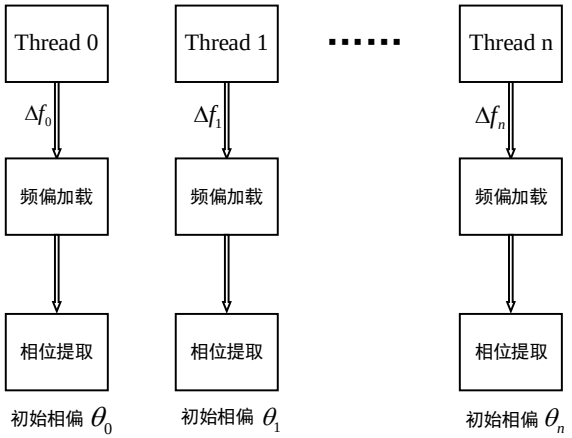

图5　初始相位并行计算线程架构

在接下来的相位频偏补偿和EVM计算模块，考虑到需要同时遍历所有候选频点和符号数据，

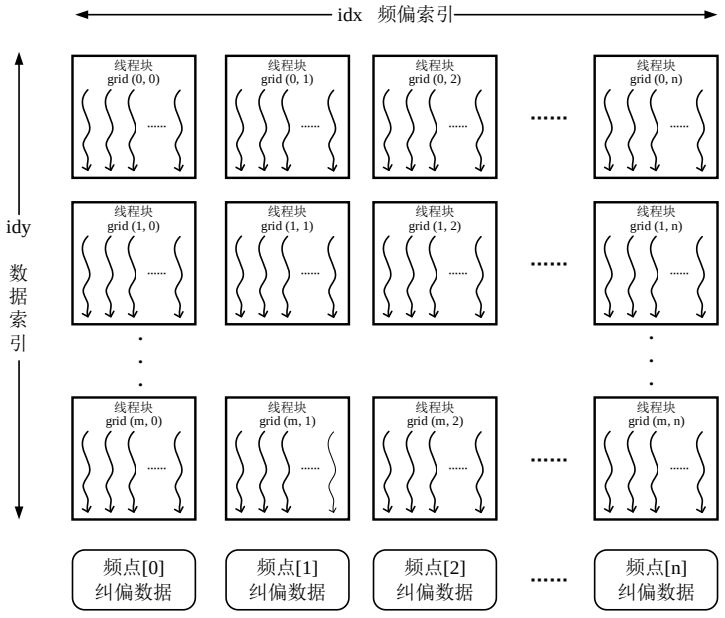

图6　频偏补偿并行计算线程排布

为提高运算效率并优化线程布局，本模块设计采用二维核函数并行计算架构，设置外层索引为频偏索引，内层索引为数据点索引。具体来说：将线程块排列设计为二维的平面模型，横向的线程块索引用于控制频偏Id，纵向的线程索引用于控制数据Id。每一列的线程块表示对应一个频点下的纠偏数据，每个线程对当前频偏下的符号数据进行补偿后，计算与理想星座点的最小欧式距离$d_{\min}$，从而得到该数据点当下的EVM，此过程的并行设计如图6所示。

在完成每个线程对单个纠偏数据的EVM计算之后，对获得的所有数据在特定频偏下的EVM进行求和，并使用并行归约算法找出最小值$\mathrm{EVM}_{\min}$，其线程索引对应的频点即为细频偏估计值Δf_{best}，使用Δf_{best}进行细频偏补偿后得到载波恢复后的信号。

3.2 GPU扫频算法优化

在GPU计算系统优化中，内存管理机制起着至关重要的作用。大量计算任务的性能瓶颈往往并非源于算力本身，而是受限于数据的加载与存储速率，因此低延迟与高带宽的内存访问成为性能优化的核心要素。以CUDA为例，其提供了多种可编程内存类型，包括：寄存器(Registers)、共享内存(Shared Memory)、本地内存(Local Memory)、全局内存(Global Memory)、常量内存(Constant Memory)、纹理内存(Texture Memory)。其中寄存器和本地内存是针对每个线程的，共享内存对一个线程块可见，全局内存、常量内存、纹理内存是全局的。通过合理利用这些内存类型和缓存，可以显著提升GPU运算的效率。首先，考虑到常量内存可以通过高速缓存机制与线程束广播机制实现对常量数据的优化访问。因此，采用常量内存存储本地星座映射数据可有效优化存取效率，并在一定程度上降低对全局内存带宽的占用，从而提升整体吞吐能力。

对于扫频算法，在计算每个扫频频点下的纠偏信号与所有星座点的之间的EVM值时，需要进行大量计算，且调制方式越高，计算量越大。为了减少此时的计算量，考虑到调制方式的星座点具有对称性，提出了一种对称规约策略，如下所述。

设扫频过程中当前频点对应的纠偏信号为$\tilde{z}=I+jQ$，调制星座下的第k个星座点可表示为：

$$s_k = I_k + jQ_k,\quad k = 1, 2, ..., M \tag{11}$$

其中，M为映射星座点总数。

定义理想星座点集合记为$\mathcal{S}$，$\mathcal{S}$中4个象限中的星座点子集分别为：

$$\mathcal{S}_{\mathrm{I}}, \mathcal{S}_{\mathrm{II}}, \mathcal{S}_{\mathrm{III}}, \mathcal{S}_{\mathrm{IV}} \tag{12}$$

EVM计算过程中寻找欧式距离的最小值需要遍历四个象限的所有点，然后取其中的最小值，即：

$$\begin{aligned} d_{\min} &= \min_{s_k \in \mathcal{S}} \left\| \tilde{z} - s_k \right\|_2 \\ &= \min\left(\min_{s_k \in \mathcal{S}_{\mathrm{I}}} \left\| \tilde{z} - s_k \right\|_2 \cup \min_{s_k \in \mathcal{S}_{\mathrm{II}}} \left\| \tilde{z} - s_k \right\|_2 \cup \min_{s_k \in \mathcal{S}_{\mathrm{III}}} \left\| \tilde{z} - s_k \right\|_2 \cup \min_{s_k \in \mathcal{S}_{\mathrm{IV}}} \left\| \tilde{z} - s_k \right\|_2 \right) \end{aligned} \tag{13}$$

由于与纠偏信号$\tilde{z}$距离最近的点一定在$\tilde{z}$所在的象限内，因此在计算时并不需要遍历其它3个象限。例如，如果$\tilde{z}$在第一象限，则式(13)变为下式：

$$d_{\min} = \min_{s_k \in \mathcal{S}_{\mathrm{I}}} \left\| \tilde{z} - s_k \right\|_2 \tag{14}$$

可以看出，此时计算量减少为原来的1/4。图7以16APSK调制方式为例进一步给出了更直观的说明。从图中可以看出，在第II、III或IV象限求$d(\tilde{z}, s_k)$的值，与将原始纠偏符号数据$\tilde{z}$映射到第一象限后求$d_{\min}$的值结果相同。因此可先将纠偏后的复数数据$\tilde{z}$统一经取模运算映射至第一象限得到$\tilde{z}_{\mathrm{mod}}$，然后在第一象限内进行最小距离

计算即可，这样可将原始的比较空间缩减为四分之一，显著减少了计算量。因此，优化后的 d_{min} 可按照下式计算：

$$d_{min}=\min_{s_k\in\mathcal{S}}\|\tilde{z}-s_k\|_2=\min_{s_k\in\mathcal{S}_I}\|\tilde{z}_{mod}-s_k\|_2 \quad (15)$$

图 8 为不同调制方式下算法优化前后的运行时间对比图。从图中可以看出，在不同调制方式下，采用常量内存的方案相比于全局内存方案执行时间的显著降低。进一步地，在常量内存基础上引入对称归约策略后，算法执行时间进一步减小，并且随着调制阶数的增加其加速效果更显著。

最后，在 FFT 算法和扫频算法设计中都使用了并行归约算法。该算法是一种利用多计算单元同时协作，将大规模数据集合合并为单一结果的并行计算算法。其核心目标是通过并行化显著加速传统串行归约(如循环累加)的过程，尤其在 GPU 并行硬件上性能提升显著。相比于 CPU 串行算法，在时间复杂度上由 $O(N)$ 降低到 $O(\log N)$，可以显著提升运算速度。为了进一步提高并行归约算法的速度，本文对其进行了优化。下面以扫频算法中计算 EVM 的和为例，对并行归约算法的优化方法进行说明。

在上面的并行归约求和算法中，将大规模求和任务分解成许多个小的子任务，通过 GPU 的海量线程并行执行。该算法高效映射于 CUDA 的层次化硬件架构，利用线程块内的共享内存实现高速数据协同与多级归约，使得当前所有活跃的线程能够同时、独立地执行相同的加法操作，相比 CPU 串行求和算法，运算速度大幅提升。

然而，观察算法的第 5 行，由于 if (tid % (2 * stride) == 0)条件语句设定了可以进行求和的活跃线程条件，从而导致 tid 为偶数的线程处于活跃状态，tid 为奇数的线程处于闲置状态。这使得同一个线程束（一个线程束有 32 个线程）内的线程利用率只有一半，产生了 CUDA 编程时常见的线程束分支(Warp Divergence)现象，导致并行效率下降。为了解决此问题，得到优化算法 1。

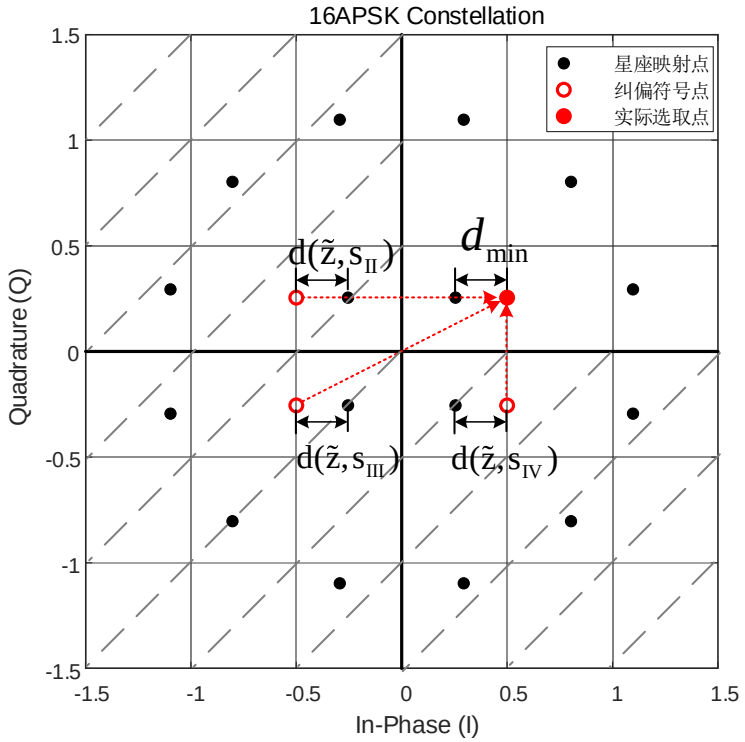

图 7　16APSK 对称性归约策略

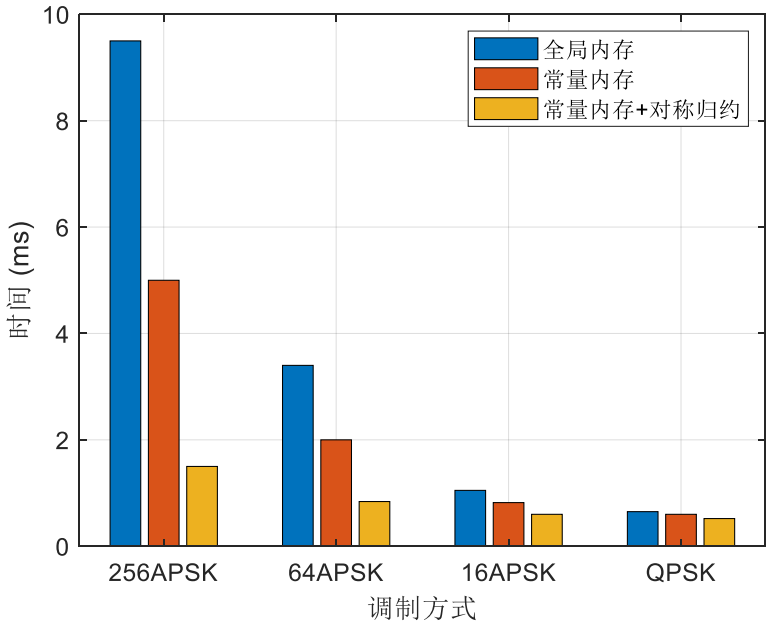

图8　不同调制模式下的优化方式时间对比

并行归约算法

```
// 对存储在 datain 数组的所有 EVM 数据求和
1: idx = blockIdx.x; tid = threadIdx.x; // 获取当前线程索引
2: dataIndex = idx * group_size + tid; // 计算全局内存数据地址
3: EVM[tid] = datain[dataIndex]; // 数据加载
4: for (int stride=1; stride < blockDim.x; stride *= 2) { // 并行归约求和
5:     if (tid % (2 * stride) == 0) {
6:         EVM [tid] += EVM [tid + stride];
7:     }
8:   } // 存储序号为 tid 的求和结果
9: if (tid == 0)
10:  EVM_sums[idx] = EVM [0]; // 保存当前数据块的求和结果
```

并行归约优化算法 1

```
// 对存储在 datain 数组的所有 EVM 数据求和
1: idx = blockIdx.x; tid = threadIdx.x; // 获取当前线程索引
2: dataIndex = idx * group_size + tid; // 计算全局内存数据地址
3: EVM[tid] = datain[dataIndex]; // 数据加载
4: for (int stride = blockDim.x / 2;stride > 0;stride /= 2) { // 并行归约求和
5:     if (tid < stride) {
6:         EVM [tid] += EVM [tid + stride];
7:     }
8:   } // 存储序号为 tid 的求和结果
9: if (tid == 0)
10:  EVM_sums[idx] = EVM [0]; // 保存当前数据块的求和结果
```

在优化算法 1 中，stride 初始值为数组长度的一半，并在每轮迭代中减半，如第 4 行所示。这样，每次求和时前一半的线程处于活跃状态，而后一半的线程处于闲置状态，使得线程束内线程要么全工作，要么全不工作，消除了线程束分支现象，从而可以提高线程的整体运行效率。

并行归约优化算法 2

```
// 对存储在 datain 数组的所有 EVM 数据求和
1: idx = blockIdx.x; tid = threadIdx.x; // 获取当前线程索引
2: dataIndex = idx * group_size + tid; // 计算全局内存数据地址
3: localSum=0.0f;
4: for (int i = 0; i < M; ++i) { // 数据加载
5:     localSum += EVM[dataIndex + i * blockDim.x];
6:   } //读入 M 个 EVM 数据并求和
7: for (int stride = blockDim.x / 2;stride > 0;stride /= 2) { // 并行归约求和
8:     if (tid < stride) {
9:         localSum [tid] += localSum [tid + stride];
10:    }
11:  } // 存储序号为 tid 的求和结果
11:  if (tid == 0)
12:  EVM_sums[idx] = localSum [0]; // 保存当前数据块的求和结果
```

在优化算法 1 中虽然消除了线程束分支现象，但是实际上每次求和运算时都只有一半的线程工作，另外一半不工作，导致线程利用率下降。为了进一步提高线程的利用率，得到优化算法 2。

优化算法2引入了一种部分和预计算策略，即在数据加载至共享内存执行并行归约之前，让每个线程连续计算M个数据的局部和（算法第4至第6行），这样所有线程都在工作，从而提高了线程的利用率。此外，数据读取阶段（第5行）以dataIndex + i * blockDim.x 的寻址方式进行，可以确保同一线程块内的线程访问连续的全局内存地址，从而实现合并内存访问，这样可以进一步提高数据读取速度。

最后，在优化算法2中，M的取值不同，对并行归约算法的整体运行速度的影响也不同。以1024个数求和为例，通过NVIDIA Nsight Compute测试工具测得不同M取值的运行时间对比如图9所示。由图可见，当M增大到一定数值时，运行时间趋于稳定，因此实际中可结合仿真确定M的取值。从运行时间对比图中可以看出，当优化算法2中的M取值为8时，即可到达最优性能，因此在本文的EVM并行归约求和算法中设置M=8。

图10所示给出了不同并行归约算法的运行时间对比图，其中Reduction_Base算法为未优化算法，Reduction_V1为采用并行归约优化算法1的算法，Reduction_V2为采用并行归约优化算法2的算法。可以看出，两种优化算法的运行时间要远快于优化前的基础算法，且优化算法2的提升效果最大。

4　仿真性能分析

在本节的仿真中，仿真平台的配置为：CPU设备型号为Intel Core i5-10400 CPU @ 2.90GHz，GPU显卡型号为NVIDIA GeForce GTX 1650，CUDA驱动程序版本为11.6。

下面先对本文采用的FFT+扫频联合载波恢复算法的性能进行分析，性能衡量指标为载波恢复输出符号$r'(n)$与发端符号$t_x(n)$之间的归一化均方误差(Normalized mean square error，NMSE)[18]，其计算公式为：

$$NMSE=\frac{\sum_{k=1}^{N}\left|r'(k)-t_x(k)\right|^2}{\sum_{k=1}^{N}\left|t_x(k)\right|^2} \tag{16}$$

上式中N为仿真次数。

若只考虑噪声的影响，其理论恢复符号的NMSE可以用以下公式计算：

$$NMSE_{theory}=\frac{\sigma_w^2}{\sigma_s^2}=\frac{1}{SNR} \tag{17}$$

其中，σ_w^2为噪声功率，σ_s^2为基带信号功率。

图13为基于CPU的FFT算法结合锁相环算法（FFT+PLL）、基于GPU的FFT算法结合扫频算法（FFT+扫频）与理论曲线之间的性能对比

Function Name	Demangled Name	Duration [us] (550.91 us)
Reduction_m2	Reduction_m2(const float..	221.41
Reduction_m4	Reduction_m4(const float..	104
Reduction_m8	Reduction_m8(const float..	74.75
Reduction_m16	Reduction_m16(const float..	74.82
Reduction_m32	Reduction_m32(const float..	74.85

图9　不同M取值的并行归约算法运行时间对比图

Function Name	Demangled Name	Duration [ms] (1.67 ms)
Reduction_Base	Reduction_Base(const float..	1.03
Reduction_V1	Reduction_V1(const float..	0.57
Reduction_V2	Reduction_V2(const float..	0.07

图10　并行归约算法设计运行时间对比

图。仿真采用QPSK调制，归一化频偏为0.02，初始相位偏移为$\pi/8$，其中扫频算法的步长分别为2.0×10^{-6}和5.0×10^{-7}。

仿真结果显示，本文提出的GPU载波恢复算法的性能在低信噪比下要优于基于CPU的FFT+PLL串行算法，并且随着信噪比的增加，与FFT+PLL串行算法性能近乎一致，逐渐贴近于理论曲线。此外，从该仿真图还可以看出，扫频步长越小，其在低信噪比下的性能越接近理论曲线，这是因为扫频算法中步长值越小，其性能越接近最优的最大似然算法，因此其性能也越好。由此可见，该算法可以通过对扫频步长的调整来实现性能与复杂度之间的折中，从而适用范围更广。

接下来，对本文提出的载波恢复算法的吞吐量性能进行仿真对比，如表3和表4所示。仿真中设置归一化频偏为0.005，每帧包含64800个比特，分别测试了从QPSK至256APSK不同的调制阶数下的执行速率，单次仿真均执行10^3帧数据，扫频算法步长精度设置为5.0×10^{-7}。

从表3中可以看出，随着调制方式的不同，优化后方案的吞吐量较优化前可以提高1到5倍，说明了本文提出的优化算法的有效性。从表4中可以看出采用GPU的扫频算法的吞吐量较CPU的PLL算法大幅提高，256APSK下可达48.1倍，说明了本文提出的扫频算法在GPU系统中的有效性。

表3　GPU扫频算法优化前后速率对比

扫频算法	优化前方案	优化后方案	提升倍数
QPSK	148.0 Msps	222.0 Msps	1.50倍
16APSK	40.2 Msps	109.0 Msps	2.71倍
64APSK	17.1 Msps	62.0 Msps	3.62倍
128APSK	11.6 Msps	49.8 Msps	4.30倍
256APSK	6.68 Msps	34.6 Msps	5.18倍

表5为采用CPU的FFT+PLL方案与采用GPU的优化后的FFT+扫频方案在不同调制模式下的吞吐量对比，其中仿真参数设置为归一化频偏为0.1，其他仿真参数与上表相同。

由上表可以看出，结合CUDA平台对FFT算法计算效率的提升，本文设计的基于GPU的载波恢复算法的吞吐量得到了进一步提升，同样调制

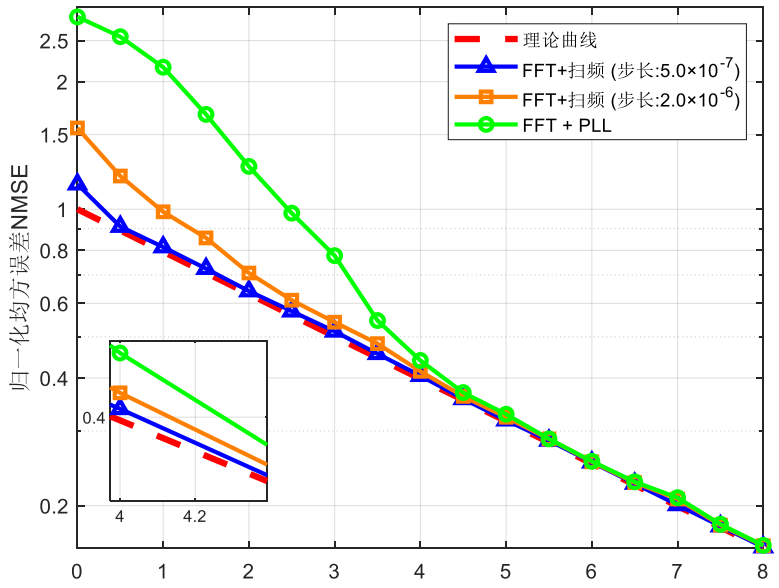

图13　两种分阶式联合算法性能对比

表4　PLL(CPU)与扫频(GPU)运行时间对比

调制模式	算法方案	吞吐量	提升倍数
QPSK	PLL(CPU)	7.35 Msps	30.2 倍
	扫频(GPU)	222.0 Msps	
16APSK	PLL(CPU)	3.92 Msps	27.8 倍
	扫频(GPU)	109.0 Msps	
64APSK	PLL(CPU)	2.18 Msps	28.4 倍
	扫频(GPU)	62.0 Msps	
128APSK	PLL(CPU)	1.18 Msps	42.2 倍
	扫频(GPU)	49.8 Msps	
256APSK	PLL(CPU)	0.72 Msps	48.1 倍
	扫频(GPU)	34.6 Msps	

表5　GPU与CPU载波恢复方案吞吐量对比

调制模式	算法方案	吞吐量	提升倍数
QPSK	FFT+PLL(CPU)	6.53 Msps	32.2 倍
	FFT+扫频(GPU)	210 Msps	
16APSK	FFT+PLL(CPU)	3.45 Msps	29.1 倍
	FFT+扫频(GPU)	100.6Msps	
64APSK	FFT+PLL(CPU)	1.93 Msps	29.5 倍
	FFT+扫频(GPU)	56.9 Msps	
128APSK	FFT+PLL(CPU)	0.94 Msps	44.5 倍
	FFT+扫频(GPU)	41.8 Msps	
256APSK	FFT+PLL(CPU)	0.61 Msps	49.5 倍
	FFT+扫频(GPU)	30.2 Msps	

阶数越高，性能提升越大，最高可达49.5倍。

5　结束语

本文围绕无线通信系统中的载波恢复问题，对基于GPU平台的高效并行化方案进行了研究，提出了一种基于FFT粗频偏估计结合扫频细频偏估计的并行载波恢复方案，并基于CUDA平台对所提出的方案进行了优化，从而进一步提高了载波恢复算法的吞吐量。本文所提出的基于GPU的载波恢复算法为面向下一代高通量无线通信系统的实时信号处理提供了高效、可扩展的解决方案，具有良好的实际应用前景。

参考文献：

[1]　Liu Y, Zhao Y, Wang W, et al. Adaptive snapshot routing strategy for software defined multi-domain satellite networks[C]// 2020 IEEE Computing, Communications and IoT Applications (ComComAp). IEEE, 2020: 1-6.

[2]　Zhang J, Ding R, Liu J, et al. QoSRA: a QoS-aware routing algorithm for software defined satellite networks[C]//2021 2nd Information Communication Technologies Conference (ICTC). IEEE, 2021: 165-171.

[3]　李荣春.基于GPU的软件无线电并行算法与系统结构关键技术研究[D].国防科学技术大学,2014.

[4]　Plishker W, Zaki G F, Bhattacharyya S S, et al. Applying graphics processor acceleration in a software defined radio prototyping environment[C]//2011 22nd IEEE International Symposium on Rapid System Prototyping. IEEE, 2011: 67-73.

[5]　Szegvari P, Hentschel C. Scalable software defined FM-radio receiver running on desktop computers[C]//2009 IEEE 13th International Symposium on Consumer Electronics. IEEE, 2009: 535-539.

[6]　Ciardi R, Giuffrida G, Bertolucci M, et al. CCSDS 131.2-B-1 Software Defined Radio receiver featuring GPU accelerators: Up to 1000x with respect to CPU implementation[C]//2022 9th International Workshop on Tracking, Telemetry and Command Systems for Space Applications (TTC). IEEE, 2022: 1-8.

[7]　Kay S. A fast and accurate single frequency estimator[J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1989, 37(12):1987-1990.

[8]　Fitz M P. Further results in the fast estimation of a single frequency[J]. IEEE Transactions on Communications, 1994, 42(234):862-864.

[9]　Mengali U, Morelli M. Data-aided frequency estimation for burst digital transmission[J]. IEEE Transactions on Communications, 1997, 45(1):23-25.

[10]　Luise M, Reggiannini R. Carrier frequency recovery in all-digital modems for burst-mode transmissions[J]. IEEE Transactions on Communications, 1995, 43(2/3/4):1169-1178.

[11]　Rife D, Boorstyn R. Single tone parameter estimation from discrete-time observations[J]. IEEE Transactions on Information Theory, 1976, 55(9):591-598.

[12]　Godard D. Self-Recovering equalization and carrier tracking in Two-Dimensional data communication systems[J]. IEEE Transactions on Communications, 1980, 28(11): 1867-1875.

[13]　Li S, He S, Cheng X, et al. A high-precision carrier synchronization algorithm based on FFT assistance[C]//2023 5th International Conference on Electronics and Communication, Network and Computer Technology (ECNCT). IEEE, 2023: 172-176.

[14]　Kim S Y, Suzuki T, Kani J I, et al. Carrier phase estimation softwarized on GPU using decision-aided phase unwrapping for

flexible optical coherent access systems[J]. Journal of Lightwave Technology, 2020, 39(6): 1706-1714.

[15] T. Jang, S. Jeong, D. Jeon, K. D. Choo, D. Sylvester and D. Blaauw, "A Noise Reconfigurable All-Digital Phase-Locked Loop Using a Switched Capacitor-Based Frequency-Locked Loop and a Noise Detector," in IEEE Journal of Solid-State Circuits, vol. 53, no. 1, pp. 50-65, Jan. 2018.

[16] Godard D. Self-Recovering equalization and carrier tracking in two-dimensional data communication systems[J]. IEEE Transactions on Communications, 1980, 28(11): 1867-1875.

[17] Wang C, Li Y, Li K. An high-precision FFT frequency offset estimation algorithm based on interpolation and binary search [C]//2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE, 2019: 437-442.

[18] 胡润江.适用于低轨卫星通信系统的盲载波恢复技术研究与FPGA实现[D].西安电子科技大学,2022.

[作者简介]

杨金天（2001–），男，山东聊城人，西安电子科技大学硕士生，主要研究方向为物理层无线通信同步技术。

高明（1979–），男，河南新乡人，博士，西安电子科技大学副教授、硕士生导师，主要研究方向为无线通信物理层传输技术，人工智能通信技术与信息安全。

付少忠（1975–），男，湖北荆门人，博士，西安电子科技大学副教授、硕士生导师，主要研究方向为无线通信物理层传输技术，无线定位技术等。