六边形稀疏网格上的 FFT 算法①

任晓波

(中国科学院 软件研究所, 北京 100190)

摘 要: 稀疏网格是一种具有特殊分层插值性质的非均匀网格形式, 稀疏网格上的离散傅立叶变换算法称为 Hyperbolic Cross FFT 算法. 这一算法能够有效降低采样点数量, 并将指数时间复杂度的 d 维 DFT 算法降低到 $O(N\log^d N)$ [10]. 六边形网格是另一种具有特殊性质的网格, 具有在采样点数量较少和采样效率较高等优势. 本文的研究工作主要集中在将六边形网格和稀疏网格相结合, 构造六边形稀疏网格上的FFT算法. 通过定义六边形和方形网格下标之间的转换, 实现了六边形稀疏网格上的 FFT 算法, 并通过数值实验证明了这一算法的有效性.

关键词: 离散傅里叶变换; 稀疏网格; 六边形网格

FFT Algorithm on Hexagonal Sparse Grid

REN Xiao-Bo

(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Sparse grid is one of Non-uniform grid forms with special properties of hierarchical interpolation. These is an algorithm of discrete Fourier transform on sparse grid, which is called Hyperbolic Cross FFT. This algorithm effectively reduce the number of sampling points, andminimize exponential time complexity of d-dimensions DFT algorithms to $O(N\log^d N)$. Hexagonal lattice is another kind of grid which has some special propertys such as number of sampling points less and higher sampling efficiency etc. This research focus on combining hexagonal lattice with sparse grid to design FFT algorithm on hexagonal sparse grid. We successfully achieve FFT algorithm on hexagonal sparse grid by defining ahexagon and square lattice conversion between subccript. And the new algorithm is proved to be effective by numerical experiments.

Key words: discrete Fourier transform; sparse grid; hexagon lattice

1 导言

快速傅里叶变换(Fast Fourier Transform, FFT)是一种应用十分广泛的数值算法. FFT 算法广泛地应用于信号处理, 图像处理, 偏微分方程求解, 大数乘法等领域[2,12,13]. 在对高维离散傅立叶变换的研究过程中, 人们发现, 随着维度的升高, 快速傅里叶变换的时间复杂度与维度成指数关系. 稀疏网格上的的快速傅里叶变换因为能够有效解决上述问题, 逐渐引起了人们的重视. 稀疏网格是一种具有特殊分层插值性质的非均匀网格形式. 已有在稀疏网格上进行快速傅立叶变换的 Hyperbolic Cross FFT 算法. 这一算法能够有效降低采样点数量, 并将指数时间复杂度的 DFT 算法降低到$O(N\log^d N)$.

六边形网格是另一种具有特殊性质的网格. 除了矩形之外, 六边形也是自然界普遍存在的一种基本图形, 具有良好的对称性和自镶嵌性以及其他独特的性质. 在信号处理等领域, 当对各向同性的函数进行采样时, 六边形网格比正方形网格更为高效[9,11]. 因此, 六边形网格往往能够处理矩形网格并不擅长的问题.

我们发现, Hyperbolic Cross FFT 算法不仅能够与正方形稀疏网格结合, 构造任意维度的高效 FFT 算法, 并且还能部分应用到六边形网格上. 通过对均匀六边

① 收稿时间:2015-04-20;收到修改稿时间:2015-06-03

形网格进行改造, 有可能得到一种兼具六边形网格和稀疏网格特点的网格形式, 并构造简单高效的 FFT 算法.

2 稀疏网格上的快速傅里叶变换算法原理

稀疏网格的节点是分层构建的, 在一个稀疏网格上, 分布着多层的均匀网格, 每层网格的节点步长逐层减半, 网格逐步加密. 在每一层的均匀网格上进行插值算法, 每增加一层, 就在上一层的插值结构基础上再进行插值计算, 使得结果越来越精确. 根据这种分层插值特性, 可以在稀疏网格上构建相应的算法[4].

在稀疏网格上构造快速傅立叶变换算法, 首先要设计分层插值函数, 使这种函数能够满足分层插值的性质, 从而分层计算逐步逼近精确结果. 其次, 需要将得到的分层插值结果重新变换到最终的结果即离散傅里叶系数上. 因为理论的推导稍显复杂, 且在其他文献中有详细的介绍[8], 本文只简要介绍稀疏网格上 FFT 算法(又称为 Hyperbolic Cross FFT 算法)的原理.

2.1 分层基函数和非分层基函数

众所周知, 傅里叶变换是一类从时域(或空域)到频域的积分变换, 而离散傅立叶变换则是将傅里叶变换在时域和频域上离散化得到的. 对于一元函数, 若其满足傅里叶展开的条件, 则很容易得到该函数的离散傅里叶展开:

$$f(x)=\sum_{k\in\mathbf{Z}}\hat{f}_k\omega_k(x),$$

其中 $\omega_k(x)=e^{ikx}$ 为频率 k 对应的基函数, $\hat{f}_k$ 称为傅里叶系数. 对 $\omega_k(x)$ 的下标做一个简单的变换, 可以得到一种新的基函数.

$$\sigma:\boldsymbol{N}_0\mapsto\mathbf{Z}:j\mapsto\begin{cases}-j/2 & \text{if } j \text{ is even},\\ (j+1)/2 & \text{if } j \text{ is odd}.\end{cases}$$

并定义: $B_l:=\{\phi_j\}_{0\le j\le 2^l-1}$, $\phi_j:=\omega_{\sigma(j)}$. 我们将 ϕ_j 称为非分层基函数, B_l 为非分层基函数集, l 为层数, $N=2^l$ 为 1 维空间/频域点数目.

在离散傅里叶变换中, 目标是得到函数在对应频域空间中的傅里叶系数. 因此, 可以定义: $j\in G_l, G_l:=\{0,\dots,2^l-1\}$, G_l 为指标(下标)集合. 以及采样点空间: $S_l:=\{2m\pi/2^l:m=0,\dots,2^l-1\}$. 通过上述定义并根据离散傅立叶变换的计算式, 可以得到非分层基函数对应的系数(简称非分层基系数):

$$\hat{f}_j^{(l)}:=2^{-l}\sum_{x\in S_l}f(x)\phi_j^*(x),\quad j\in G_l,\tag{1}$$

$\phi_j^*(x)$ 为 $\phi_j(x)$ 的共轭函数.

对于 d 维空间上的离散傅立叶变换, 各个维度相互独立, 可以通过空间上张量积复合得到. 简单地说明如下:

$\boldsymbol{l}:=(l_1,l_2,\dots,l_d)$ 为 d 维下标, $\boldsymbol{G_l}:=G_{l_1}\times G_{l_2}\times\dots\times G_{l_d}$ 为 d 维下标的集合, $\boldsymbol{S_l}:=S_{l_1}\times S_{l_2}\times\dots\times S_{l_d}$ 为 d 维均匀网格, $\phi_{\boldsymbol{j}}:=\omega_{\sigma(j_1)}\cdot\omega_{\sigma(j_2)}\dots\omega_{\sigma(j_d)}$ 为相应的非分层基函数, 以及在 d 维上的非分层基系数:

$$\hat{f}_{\boldsymbol{j}}^{(\boldsymbol{l})}:=2^{-|\boldsymbol{l}|_1}\sum_{\boldsymbol{x}\in\boldsymbol{S_l}}f(\boldsymbol{x})\phi_{\boldsymbol{j}}^*(\boldsymbol{x}),\quad \boldsymbol{j}\in\boldsymbol{G_l}.$$

通过以上的讨论, 我们获得了非分层基系数的在 d 维上的定义. 为了在稀疏网格上应用分层插值算法, 还需要定义分层基函数和相应的分层基系数.

首先在 1 维建立分层基函数, 定义如下基函数:

$$\varphi_j:=\begin{cases}\phi_0 & j=0,\\ \phi_j-\phi_{2^l-1-j} & 2^{l-1}\le j\le 2^l-1, l\ge 1,\end{cases}$$

称 $2^{l-1}\le j\le 2^l-1$ 为第 l 层. 此外有: $B_l^{\boldsymbol{h}}:=\{\varphi_j\}_{0\le j\le 2^l-1}$. 则函数 u 可以得到两种基函数空间下的等价表述, 即:

$$u=\sum_{0\le j\le 2^l-1}\hat{f}_j^{(l)}\phi_j=\sum_{0\le j\le 2^l-1}\check{f}_j^{(l)}\varphi_j.$$

其中, $\hat{f}_j^{(l)}$ 为非分层基系数, $\check{f}_j^{(l)}$ 为分层基系数. 这样, 我们可以在非分层基系数和分层基系数之间建立联系, 实现两者之间的转换.

再次, 定义分层基上的指标集:

$$G_v^h:=\begin{cases}\{0\} & v=0,\\ \{2^{v-1},\dots,2^v-1\} & v\ge 1,\end{cases}$$

以及插值点空间: $S_l^h:=S_l\setminus S_{l-1}, S_{-1}=\varnothing$, 也即: $S_l^h:=\{2m\pi/2^l:m=2^{l-1},\dots,2^l-1\}$.

最后, 把分层基系数推广到 d 维. 定义 L 为 d 维网格的总层数, 有: $\Gamma_L=\{\boldsymbol{l}:|\boldsymbol{l}|=l_1+l_2+\dots+l_d\le L\}$. 对满足 Γ_L 约束的 d 维稀疏网格为(图 1 为 2 维时的图像):

$$\boldsymbol{S}_{\boldsymbol{l}}^{\boldsymbol{h}}=S_{l_1}^h\times S_{l_2}^h\times\dots\times S_{l_d}^h,\ \boldsymbol{l}\in\boldsymbol{\Gamma_L}.$$

对频域空间, 同样类似的定义. 1 维频域空间为 $H_l=\{1-2^{L-1},\dots,2^{L-1}\}$, d 维傅里叶频域空间 $\boldsymbol{H_l}=\otimes_{l_1+\dots+l_d\le L}H_{l_i}$, 据此, 作出它的 2 维时的图像(图 2).

其他定义的 d 维形式在这里省略.

因为频域空间的图像类似与双曲线与坐标轴所围成的截面, 故称为 Hyperbolic Cross 区域. 我们也将算法称为 Hyperbolic Cross FFT 算法(简称 HCFFT 算法).

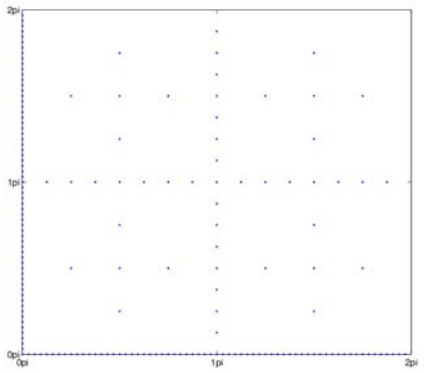

图 1 2 维稀疏网格 S_5

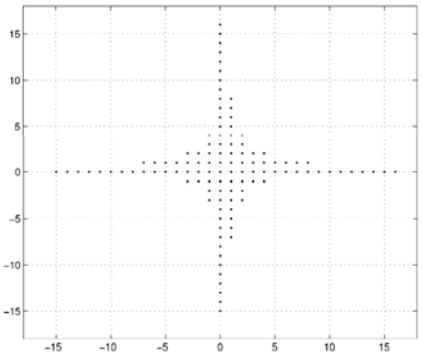

图 2 2 维 Hyperbolic Cross 空间 H_5

2.1 2 维 HCFFT 算法的实现

在这里我们定义了两种傅里叶变换的基函数, 第一种非分层基函数和它的非分层基系数直接对应频域点, 第二种分层基函数主要是为在稀疏网格上构建分层插值算法. 通过两种基函数的系数之间的转换, 就能实现稀疏网格上的 FFT 算法. 在一个维度上, 可以通过如图3所示的流程, 获得分层基系数. 那么, 对于对于d维稀疏网格空间, 对每一个维度分别执行该流程, 就能够得到所有维度上的分层基系数. 再将分层基系数, 通过反变换得到非分层基系数, 就可以得到最终的傅里叶系数. 这就是d维HCFFT算法的基本原理. 这个过程的离散傅立叶变换, 是从 $\boldsymbol{S}_l^h$ 到 $\boldsymbol{H}_l^h$ 的变换.

在实现HCFFT算法时, 简单的1维FFT算法可以通过调用 FFT 专用的数学库中的标准接口得到. 这里我们使用的 FFTW3 作为底层 FFT 实现, 版本号为FFTW3.3.4. FFTW3[4]是一种计算1维或多维DFT, DST, DCT 等的 C 语言程序库, 它经过长期的优化, 具有接口广泛, 效率较高, 底层支持并行化等优点.

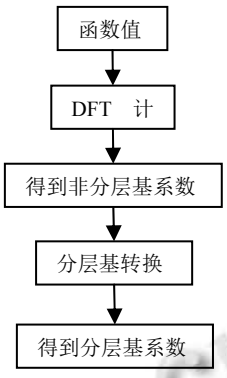

图 3 1 维上获得分层基系数的流程

为实现 Hyperbolic Cross FFT 算法, 首先要定义计算和存储的数据结构. 为了使稀疏网格上的原始数据存储方式和 Hyperbolic Cross 区域上的存储方式统一到一起, 定义一种正整数到实数的映射:

$$\begin{aligned}&\rho_k := \rho(k) = \pi(\alpha_0 2^0 + \alpha_1 2^{-1} + \alpha_2 2^{-2} + \ldots + \alpha_n 2^{-n}),\\ &k = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + \ldots + \alpha_n 2^n.\end{aligned} \tag{2}$$

在这一映射下, 设 A 为 $N \times N$ 的存储矩阵 ($N = 2^n, n$ 为层数), 各个维度独立, 有: $\mathrm{A}[k] = f(\rho_k)$ 为初始的输入数据, $\mathrm{A}[k] = \hat{f}_{\sigma(k)}$ 为输出的傅里叶系数. 可以验证, $\mathrm{A}[k] = f(\rho_k)$ 涵盖了所有的数据点坐标. 即 $\rho_k \in \{2i\pi / 2^l, i = 0, \ldots, 2^l - 1\}$, l 为当地的层数, 而 $\mathrm{A}[k] = \hat{f}_{\sigma(k)}$ 则将 $k \in \{0, \ldots, 2^l - 1\}$ 通过 $\sigma(k)$ 映射到了 $\{1 - 2^{l-1}, \ldots, 2^{l-1}\}$ 上. 因为将 $\mathrm{A}[k] = f(\rho_k)$ 设为存储数据, 导致了原始数据在进入 DFT 计算前次序不符合 FFTW 库的接口的要求, 所以在进入DFT计算前, 要对数据按 ρ_k 的大小从小到大排列. 另外, 在DFT计算后, 数据仍要重新按 k 的大小进行组织, 因此, 在 DFT 前后, 必须插入两次排序操作. 据此, 给出如下的算法实现[6]:

算法 1 *2 维 HCFFT 算法*

输入: set A[k1,k2]= $f(\rho_{k1}, \rho_{k2})$
输出: A[k1,k2]= $\hat{f}_{\sigma(k1),\sigma(k2)}$

```
1.  for  k2 = 0  to  2^n − 1
2.        m = n − ⌈log2(k2 + 1)⌉
3.        sort_by_rho(A[ 0...2^m − 1, k2 ])
4.        DFT1d(A[ 0...2^m − 1, k2], m)
5.        sort_by_k(A[ 0...2^m − 1, k2 ])
6.        hira(A[ 0...2^m − 1, k2], m)
7.  end for
8.  for  k1 = 0  to  2^n − 1
9.        m = n − ⌈log2(k1 + 1)⌉
10.       sort_by_rho(A[k1, 0...2^m − 1 ])
```

11.　　DFT1d(A[k1, 0...2^m-1],m)
12.　　sort_by_k(A[k1, 0...2^m-1])
13. end for
14. for　k2 = 0　to　2^n-1
15.　　$m = n - \lceil \log_2(\text{k2}+1) \rceil$
16.　　dehira(A[0...2^m-1,k2],m)
17. end for

上述算法中, n 为总层数, $m = n - \lceil \log_2(\text{k2}+1) \rceil$ 求得本地层数 m, $\lceil \bullet \rceil$ 为向上取整操作. 而 sort_by_rho()和 sort_by_k()是在 DFT 计算前后调整元素顺序的操作. hira()函数的功能是将非分层基系数转换为分层基系数, dehira()则是将分层基系数转换为非分层基系数, 以获得最终结果. 值得注意的是, 算法 1 只在第一个循环时(第一个维度)执行非分层基系数向分层基系数进行转换, 这是由于hira()和dehira()是互逆操作, 即先后执行一次相当于不改变结果, 在第二个循环(第二个维度)上执行 hira(), 意味着最后还需要增加一次循环执行 dehira(), 这样反而降低了程序的效率.

最后, 参照文献[6], 我们不加证明地给出 hira()和 dehira()算法.

算法 2　*1 维分层基到非分层基算法 dehira(A,n)*

输入: A[0...2^n-1]中存储了 1 维分层基系数
输出: A[0...2^n-1]中存储 1 维非分层基系数
1. for j=1 to n
2.　　for　$k = 2^{j-1}$ to 2^j-1
3.　　A[2^j-1-k] = A[2^j-1-k] − A[k]
4.　　end for
5. end for

算法 3　*1 维非分层基到分层基算法 hira(A,n)*

输入: A[0...2^n-1]中存储了 1 维非分层基系数
输出: A[0...2^n-1]中存储了 1 维分层基系数
1. for j=n downto 1
2.　　for　$k = 2^j-1$ downto 2^{j-1}
3.　　A[2^j-1-k] = A[2^j-1-k] + A[k]
4.　　end for
5. end for

2.3 算法复杂度

根据论文[8]的成果, 若稀疏网格的层数为 L, $N = 2^L$ 为一个方向上网格点数最大值, 则 d 维 HCFFT 算法的时间复杂度应为: $O(2^L \cdot L^d)$, 也就是 $O(N \cdot \log^d N)$. 在空间复杂度上, 存储数据点需要:

$$\sum_{l_1+l_2+\ldots+l_d \le L} 2^{l_1} \times 2^{l_2} \times \ldots \times 2^{l_d}$$

个基本数据单元, 约为$O(L^d \cdot 2^L)$. 可见, 相对均匀网格的$O(N^2)$, 稀疏网格的存储空间大大减少了.

3 六边形稀疏网格网格上的DFT算法原理

在六边形网格上构造 DFT 算法, 主要的目标是将六边形网格上的计算转换到四边形网格上, 并用四边形网格上的已有算法来实现. 因此, 通过寻找六边形网格上的 DFT 算法到四边形网格上的 DFT 算法的转换, 并对两者的坐标进行变换, 就能够构造出六边形网格上的算法. 再在四边形网格 DFT 上, 应用 HCFFT 算法. 这就是本文的主要思路.

3.1 六边形网格与周期性

在六边形上, 普通的二维直角坐标系不易操作, 这里主要采用齐次坐标系[3]. 选取 3 个互成 120 度, 轮换对称的坐标轴 , 使与直角坐标系的原点重合, 并使得 v_2 轴与 y 轴重合, 如图 4. 这一坐标系有如下的性质: 任意 (v_1, v_2, v_3) 是齐次坐标系上的坐标, 则满足 $v_1 + v_2 + v_3 = 0$. 显然, 这 3 个坐标不是互相独立的. 如此, 我们可以简单定义六边形网格上的离散傅里叶变换指标集:

设 $\boldsymbol{j} = (j_1, j_2, j_3), \boldsymbol{j} \in \Lambda_N$, 且 Λ_N 定义为六边形区域, $\Lambda_N := \{\boldsymbol{j} \in \Lambda_N : -N \le j_1, j_2, -j_3 < N\}$.

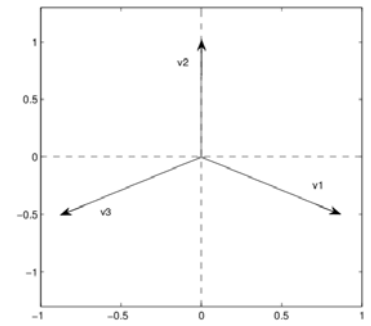

图 4　本文使用的齐次坐标系

$\boldsymbol{j}$ 与六边形上的采样点的关系为:

$$\boldsymbol{t}_j = \boldsymbol{t}_j^N = (\frac{j_1}{N}, \frac{j_2}{N}, \frac{j_3}{N}).$$

因为离散傅立叶变换中往往认为计算区域为一个完整的周期, 有必要定义六边形网格上的周期性行为. 则可以认为, 满足关系

$$\boldsymbol{u}_{\boldsymbol{j}'+\boldsymbol{Nm}}=\boldsymbol{u}_{\boldsymbol{j}'},$$

对 $\forall \boldsymbol{j}',\boldsymbol{m}\in\boldsymbol{Z}^2$ 成立，$\boldsymbol{N}$ 是周期矩阵，$\boldsymbol{j}'=(j_1,j_2)$ 是 $\boldsymbol{j}=(j_1,j_2,j_3)$ 的部分向量. 根据文献[7]，一个可行的周期矩阵为:

$$\boldsymbol{N}=\begin{pmatrix}2N & -N\\ -N & 2N\end{pmatrix}$$

在本文中，对六边形网格的计算是基于子区域分解的，把六边形区域按照图 5 划分成 3 个区域，并依次编号为 1, 2, 3，这三个区域的坐标约束为:

$$1:\begin{cases}-N\le k_1\le -1,\\ 0\le k_2\le N-1,\end{cases}$$

$$2:\begin{cases}0\le k_1\le N-1,\\ 1-N\le k_3\le 0,\end{cases}$$

$$3:\begin{cases}-N\le k_2\le -1,\\ 1\le k_3\le N.\end{cases}$$

可见，为满足周期性的要求，图 5 中虚线部分是不被取到的.

3.2 基于“多色排序”的六边形网格离散傅里叶变换

以 $\hat{u}_{\boldsymbol{k}},\boldsymbol{k}\in\Lambda_N$ 表示六边形上的傅里叶系数，根据文献[7]，得到六边形上的 DFT 计算式为:

$$\hat{u}_{\boldsymbol{k}}=\sum_{\boldsymbol{j}\in\Lambda_N}u_{\boldsymbol{j}}\cdot\boldsymbol{e}^{-i\frac{2\pi}{3N}\boldsymbol{k}\cdot\boldsymbol{j}}. \tag{3}$$

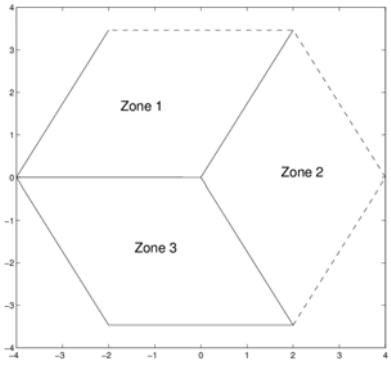

图 5 六边形网格的一种区域分解方式

对于 $\boldsymbol{k}\cdot\boldsymbol{j}$，利用齐次坐标的特点，可以得到一系列等价式:

$$\begin{aligned}\boldsymbol{k}\cdot\boldsymbol{j}&=k_1\cdot j_1+k_2\cdot j_2+k_3\cdot j_3\\ &=k_1\cdot j_1+k_2\cdot j_2-(k_1+k_2)\cdot(-j_1-j_2)\\ &=k_1(j_1-j_3)+k_2(j_2-j_3)\\ &=k_1(j_1-j_2)+k_3(j_3-j_2)\\ &=k_2(j_2-j_1)+k_3(j_3-j_1).\end{aligned}$$

观察到 j_1-j_2,j_2-j_3,j_3-j_1 具有模 3 同余的性质，也即:

$$j_1-j_2\equiv j_2-j_3\equiv j_3-j_1 \mod 3.$$

我们可以将其划分为 3 种可能的情况，分别为 $s=0,1,-1$，则有:

$$\begin{cases}j_1-j_2=3p+s,\\ j_2-j_3=3q+s,\\ j_3-j_1=-3p-3q-2s.\end{cases} \tag{4}$$

这里 p, q 是除 3 后得到的商，为整数，令 s 表示余数. 这样，可以根据 s 的值把六边形网格上的点分为三类即 Λ_N^s，它们之间互不相交且 $\Lambda_{\boldsymbol{N}}=\Lambda_N^0\cup\Lambda_N^1\cup\Lambda_N^{-1}$，并把式(3)化简为:

$$\begin{aligned}\hat{\boldsymbol{u}}_{\boldsymbol{k}}=&\sum_{\boldsymbol{j}\in\Lambda_N^0}\boldsymbol{u}_{\boldsymbol{j}}\cdot\boldsymbol{e}^{-i\frac{2\pi}{N}[p\boldsymbol{k}_1+q(-\boldsymbol{k}_3)]}\\ &+\sum_{\boldsymbol{j}\in\Lambda_N^1}\boldsymbol{u}_{\boldsymbol{j}}\cdot\boldsymbol{e}^{-i\frac{2\pi}{3N}[p\boldsymbol{k}_1+q(-\boldsymbol{k}_3)]}\cdot\boldsymbol{e}^{-i\frac{2\pi}{3N}(\boldsymbol{k}_1-\boldsymbol{k}_3)}\\ &+\sum_{\boldsymbol{j}\in\Lambda_N^{-1}}\boldsymbol{u}_{\boldsymbol{j}}\cdot\boldsymbol{e}^{-i\frac{2\pi}{3N}[p\boldsymbol{k}_1+q(-\boldsymbol{k}_3)]}\cdot\boldsymbol{e}^{-i\frac{2\pi}{3N}(\boldsymbol{k}_3-\boldsymbol{k}_1)}\end{aligned} \tag{5}$$

上式中，右边的三项分别为一个标准的四边形 DFT 乘以一个所谓“旋转因子”的系数. 因此，我们把六边形上的计算，化简到了四边形上. 我们把 Λ_N，按照 $j_1-j_2 \mod 3$ 的模来划分，类似对网格中的点进行着色，所以称这种方法为“多色排序”.

由于式(5)右边三项的 DFT 计算部分是一致的，我们把这三项统一到频域空间在第一象限的模式进行计算. 从而对 $\boldsymbol{k}\in\Lambda_N$，需要按照区域的不同调整到第一象限中. 这一从六边形的频域空间，到四边形的频域空间的变换，称之为正向频域变换.

3.3 二维四边形 Hyperbolic Cross 区域到六边形频域空间的变换

要利用 HCFFT 算法，频域空间必须在 Hyperbolic Cross 区域，同时，(p,q)必须在四边形稀疏网格指标集上. 那么为使这一条件达成，需要将数据从稀疏网格再变换到六边形网格上，同时，频域空间也要从 Hyperbolic Cross 区域变换到六边形上，称这一变换为反向频域变换. 对空间区域的变换，每一个 s 值，都将相应的稀疏网格变换到对应的六边形网格点，将它们叠加到一起，得到图 6 及图 7，称这样的网格点为六边形稀疏网格点. 图 8 和图 9 反应了频域点上的反向变换过程，我们把得到的六边形网格上的频域空间称为六边形 Hyperbolic Cross 区域. 六边形 Hyperbolic Cross 就是算法最终得到的频域空间.

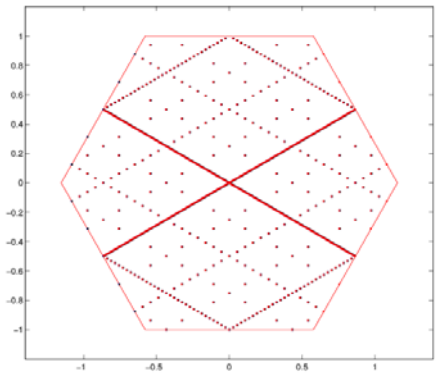

图 6　(j_1,j_2,j_3)平移到六边形区域, s=-1,0,1,层数为 9

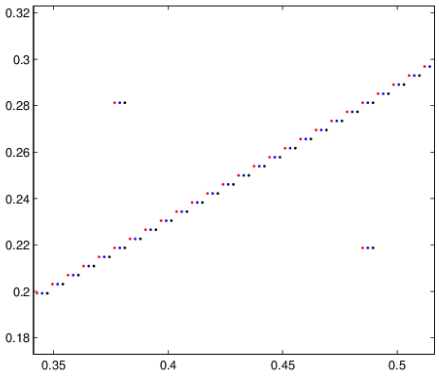

图 7　(j_1,j_2,j_3)平移到六边形区域, s=-1,0,1,层数为 9

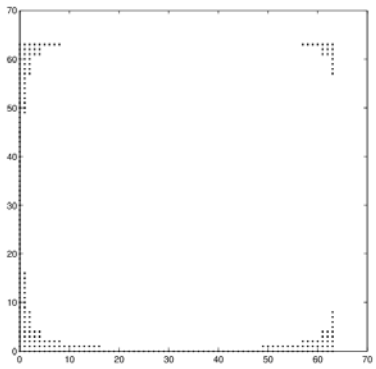

图 8　通过周期平移调整到第一象限上的 Hyperbolic Cross 点集, 层数 6

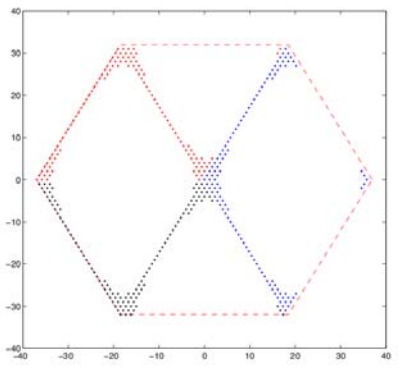

图 9　六边形上的频域空间, 层数 5

4　算法实现及数值实验

4.1　六边形稀疏网格上的 FFT 算法实现

经过以上的讨论, 六边形稀疏网格上的快速傅里叶算法的过程可以较为清晰地得到. 根据式(5), 首先生成六边形稀疏网格点集, 将数据装订到HCFFT算法的输入, 再通过 HCFFT 算法得到四边形网格上的 Hyperbolic Cross 区域, 最后利用反向频域变换, 得到式(5)右边的三项, 乘以旋转因子, 得到六边形 Hyperbolic Cross 上的频域点数值, 即傅里叶系数.

对六边形系数网格进行 FFT 计算, 高效的稀疏网格的存储方式是必不可少的. 针对所进行的研究工作, 需要存储矩阵具有如下的特点:

①较高的压缩率. 因为稀疏网格上点的数量要远远地小于均匀网格, 如果存储方式不当, 将浪费较大的内存空间, 甚至导致无法进行大规模的数据计算.

②较高的随机查找和写入效率. 六边形稀疏网格上的 DFT 计算在 2 维 HCDFT2d 前需要至少要读取每个元素 1 次, 写入 3 次. 因此, 查找和写入的效率对性能的影响尤为关键.

③不需要插入, 删除, 移动数据. 这一性质为选择存储方案提供了很大的方便.

权衡以上各个方面, 本文选择了 CSR 存储格式.

CSR(行优先压缩存储, Compressed Sparse Row)存储格式是一种应用广泛的通用稀疏矩阵存储格式. CSR 格式通常需要三个数组来存储. 设稀疏矩阵有 N_r 行 N_c 列, 共有 $N_v \le N_r \times N_c$ 个非零元素. 第一个数组为正整数类型的行偏移数组 R[$0...N_r$], 第二个数组为列下标数组 C[$0...N_v-1$], 第三个数组为数据类型的数据存储数组 V[$0...N_v-1$]. R 数组长度为 N_r+1, 记录每行第一个元素的列下标在列下标数组 C 中的偏移, 最后一个元素记录指向数据数组 V 的尾部, 以防止越界. C 数组长度为 N_v, 按从行从地位到高位, 列从低位到高位的顺序记录每个元素在矩阵中的列下标, 每个元素和它对应的数据相对位置一致, 即 C[i]存储了 V[i]数据的列下标. 给定下标(i,j)查找数据元素时, 首先获得 R[i], 在 C 数组中从 C[R[i]]个元素开始, 到 C[R[i+1]]结束, 查找是否存在 C[t]==j. 若找到, 则 V[t]为元素值. 否则, 元素值为 0. 详见算法 5.

算法 4 给出 CSR 格式的顺序存储算法. CSR 格式的随机插入元素代价高昂, 且在本文的应用中并不是必须的, 故不需要相应的算法.

算法 4 CSR 矩阵的顺序存储算法

```
1.  offset = 0
2.  for j1 从低到高
3.      R[j1] = offset
4.      for  j2 从低到高
5.          if M[j1, j2] != 0
6.              C[offset] = j2
7.              V[offset] = M[j1, j2]
8.              offset = offset + 1
9.          end if
10.     end for
11. end for
```

再给出 CSR 矩阵的查找算法:

算法 5 CSR 矩阵的随机查找算法 CSR_find(j_1 j_2)

```
1.  find R[j1]
2.  for t= R[j1] to R[j1+1]-1
3.      if C[t] == j2
4.          return V[t]
5.      end if
6. end for
7. return 0
```

具体的六边形稀疏网格上的 FFT 算法如下:

算法 6 六边形稀疏网格上的 FFT 算法

输入: 存储了六边形稀疏网格采样点的原始数据文件, 层数 L

输出: CSR 格式稀疏矩阵 Fhex, 存储了六边形 Hyperbolic Cross 频域点的值.

1. for s in {-1,0,1}
2. for $(p,q)\in[0,N-1],\lceil\log(p+1)\rceil+\lceil\log(q+1)\rceil\le L$
3. 根据式(4)得到(j_1,j_2,j_3), 并将之存入J_{set}
4. end for
5. end for
6. 按照J_{set}从数据文件读入数据, 存入 CSR 格式稀疏矩阵 M, 完成初始化.
7. for s in {-1,0,1}
8. for all $(l_1,l_2)\in[0,N-1]$, $\lceil\log(l_1+1)\rceil+\lceil\log(l_2+1)\rceil\le L$
9. $\rho_{l_1}=\rho(l_1),\rho_{l_2}=\rho(l_2)$//式(2)
10. $p=\rho_{l_1}\cdot N/2\pi, q=\rho_{l_2}\cdot N/2\pi$
11. 根据式(4)从(p,q)得到(j_1,j_2,j_3)
12. (j_1,j_2,j_3)周期性地调整到六边形区域, 得到(j_1^*,j_2^*,j_3^*)
13. $A[l_1,l_2]=M(j_1^*,j_2^*,j_3^*)$

// $M(j_1^*,j_2^*,j_3^*)$为(j_1^*,j_2^*,j_3^*)所确定的函数值

//这样, 使得: $A[l_1,l_2]=u(j_1^*/N,j_2^*/N,j_3^*/N)$

14. end for
15. 数据准备完毕, 对 $A[0\ldots N-1,0\ldots N-1]$ 进行 HCFFT 算法. 即:

$$\hat{u}^{(s)}_{l_1,l_2}=\sum_{j\in\Lambda_N^s}u_j e^{-i\frac{2\pi}{N}(pl_1+ql_2)}$$

16. end for
17. 将频域反变换到六边形区域, 即从(l_1,l_2)到(k_1,k_2,k_3)
18. 执行式(5)的加和
19. 存储结果到 Fhex(k_1,k_2,k_3)

4.2 性能测试

本文的主要测试在并行软件与计算科学实验室的开发测试机上进行, 该平台采用的 CPU 型号为 Intel Xeon X5550, 主频 2.67GHz, 64 位 8 核心, 内存容量为 16G. 操作系统为 Ubuntu 11.04, 编译器版本为 gcc 4.5.2.

我们选择并行软件与计算科学实验室孙家昶老师, 李会元老师等开发的六边形网格傅里叶变换的软件包 FFTH[1]. FFTH 实现了 2 维和 3 维六边形均匀网格上的离散傅里叶变换, 这里使用 2 维的 FFTH 计算程序进行比较研究. 因为本文的算法和 FFTH 算法都调用了底层 FFTW3 库, 而通过编译选项的调节, 我们使底层 FFTW3 库均采用串行库, 以获得公平的结果.

我们把 FFT 计算分成两部分, 一部分为启动时间, 即装载数据, 制定 fft_plan 等操作的时间; 另一部分为计算时间, 即数据准备完成后开始 DFT 计算的时间. 统计两种程序的这两部分程序, 可以发现他们各自的一些特点.

表 1 六边形稀疏网格 FFT 实测结果

层数	数据规模	平均启动时间(s)	平均计算时间(s)
5	336	0.00	0.00
6	768	0.00	0.00
7	1728	0.00	0.01
8	3840	0.01	0.03
9	8448	0.02	0.07
10	18432	0.06	0.35
11	39936	0.15	1.30
12	86016	0.43	4.93
13	184320	1.31	19.26
14	393216	4.33	75.52

六边形稀疏网格 FFT 算法当层数达到 15 时, 数据点数量为 835584, 超过测试平台的内存容量.

表 2 六边形均匀网格 FFTH 实测结果

层数	数据规模	平均启动时间(s)	平均计算时间(s)
5	3072	0.1	0.00
6	12288	0.31	0.00
7	49152	0.94	0.00
8	196608	2.23	0.01
9	786432	8.12	0.05
10	3145728	78.31	0.23
11	12582912	146.18	1.10
12	28311552	332.48	4.56

首先比较算法的启动时间, 六边形稀疏网格上的 FFT 算法在同样的层数时, 数据规模要远小于 FFTH 算法. 因为大规模数据的初始化是 I/O 密集型操作, 往往比 FFT 算法本身更耗费时间, 所以, 新算法在启动速度上相对来说有数量级地提高. 实际情况中, 因为数据点数量减少, 采样时间也将大大减少. 此外, 在计算到第 12 层时, FFTH 算法已经十分缓慢, 以至于难以接受, 而六边形稀疏网格 FFT 算法直到第 15 层才因为内存不足而无法进行计算. 因此, 稀疏网格带来的好处是降低了整体的数据量.

在具体的 FFT 计算部分, 六边形稀疏网格 FFT 算法的计算速度不如 FFTH 算法, 但仍然较快. 这里的原因主要有两方面, 第一是 2 维 HCFFT 算法的理论时间复杂度相对均匀网格 FFT 算法优势并不明显, 第二则可能是由于本算法采用的 FFTW3 优化策略为 FFTW_ESTIMATE, 相对优化效果的不如 FFTH 算法采用的 FFTW_PATIENT. 尽管如此, 新算法的还是足够快速有效的.

此外, 相对于普通的方形网格 FFT 算法, 六边形网格具有独特的适应性, 便于应用在一些特殊的场合.

5 结语

本文在基于稀疏网格的 Hyperbolic Cross FFT 算法的基础上, 提出了将稀疏网格与六边形网格相结合的设想. 我们基于六边形区域划分, 得到将六边形上 DFT 转化为方形网格上 DFT 的计算式. 然后通过六边形网格点与稀疏网格点、六边形 Hyperbolic Cross 点与方形 Hyperbolic Cross 点之间的变换, 成功地将六边形网格与稀疏网格上 FFT 算法的结合. 通过设计合适的存储数据结构, 分别实现了四边形稀疏网格 HCFFT 算法和六边形稀疏网格上的 FFT 算法. 经过性能实验的比较分析, 验证了本算法具有较快的启动速度, 以及良好的运算性能.

参考文献

1 陈家杰,李会元,张先轶.六边形区域快速傅里叶变换的 CUDA-MPI 算法及其实现.数值计算与计算机应用,2012, 33(1):59–72.

2 李会元,乔海军.六边形 Fourier 谱方法.应用数学与计算数学学报,2013,27(1):147–162.

3 孙家昶,姚继锋.平行六边形区域上的快速离散傅立叶变换. 计算数学,2004,26(3):351–366.

4 Frigo M, Johnson SG. The design and implementation of FFTW3. Proc IEEE, 2005, 93(2): 216–231.

5 Griebel M, Oswald P, Schiekofer T. Sparse grids for boundary integral equations. Numerische Mathematik, 1993, 83(2): 279–312.

6 Hallatschek K. Fourier transformation auf dünnen Gittern mit hierarchischen Basen. Numerische Mathematik, 1992, 63(1): 83–97.

7 Sun JC, Li HY. Fast Fourier transform on hexagons. In: Wu Zhang, et al. eds. Current Trends in High Performance Computing and its Applications, Springer Berlin Heidelberg, 2005: 357–362.

8 Griebel M, Hamaekers J. Fast discrete Fourier transform on generalized sparse grids. Lecture Notes in Computational Science & Engineering. 2014. 75–108.

9 Murphy P, Gallagher N. Hexagonal sampling techniques applied to Fourier and Fresnel digital holograms. J. Opt. Soc. Am. 1982, 72(7): 929–937.

10 Staunton RC, Storey N. A comparison between square and hexagonal sampling methods for pipeline image processing. Advances in Intelligent Robotics Systems Conference. International Society for Optics and Photonics. 1990.

11 Zhao X, Wang H, Cong F, et al. High-resolution restoration of image based on fourier transform. Electronics Optics & Control, 2010, 09: 25–36.

12 Sziklas EA, Siegman AE. Diffraction calculations using fast Fourier transform methods. Proc. of the IEEE, 1974, 62(3): 410–412.