DOI:10.3969/j.issn.1005-202X.2018.12.010

医学影像物理

三维医学图像核回归算法的GPU加速研究

王玉琨[1],刘蓉[1,2],温铁祥[2],李凌[2]
1.河南理工大学计算机科学与技术学院,河南 焦作 454000;2.中国科学院深圳先进技术研究院,广东 深圳 518055

【摘 要】核回归理论被广泛应用于医学图像处理和医学图像重建领域,并取得了十分显著的效果。它包括传统核回归方法(CKR)和控制核回归方法(SKR)。三维SKR算法比三维CKR算法具有更优的去噪效果和边缘保持效果,但三维SKR算法的计算量过于庞大且复杂,使其应用领域受到限制。目前,医学图像重建使用的是基于GPU的三维CKR算法,所以基于GPU的三维SKR算法的实现是一项有研究价值且具有挑战性的工作。本文首先优化三维SKR算法的计算过程,然后利用GPU进行CUDA编程实现三维SKR并行加速算法。实验表明,基于GPU的三维SKR算法与基于CPU单线程三维SKR算法相比能获得约244.9~246.3倍的加速比,与基于CPU多线程三维SKR算法相比能获得约123.0~137.4倍的加速比。

【关键词】GPU加速;CUDA编程;三维传统核回归方法;三维控制核回归方法
【中图分类号】R318 【文献标志码】A 【文章编号】1005-202X(2018)12-1417-09

GPU acceleration for kernel regression algorithm of three-dimensional medical image

WANG Yukun[1], LIU Rong[1,2], WEN Tiexiang[2], LI Ling[2]
1. School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454000, China; 2. Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China

Abstract: Kernel regression which has been widely used in the field of medical image processing and medical image reconstruction, achieving many significant results, includes classic kernel regression (CKR) and steering kernel regression (SKR). Three-dimensional (3D) SKR has better denoising effects and edge-preserving results than 3D CKR. At present, GPU-based 3D CKR algorithm has been used on medical image reconstruction, while the computational complexity of 3D SKR algorithm limits the application of 3D SKR algorithm. The realization of GPU-based 3D CKR algorithm is a valuable research topic and remains challenging. In this experiment, the calculation process of 3D SKR algorithm is firstly optimized, and then CUDA programming is used to implement the GPU-based 3D SKR parallel acceleration algorithm. The experiments show that the speedup ratio of 3D SKR algorithm based on GPU is about 244.9-246.3 as compared with single-threaded CPU-based 3D SKR algorithm, and about 123.0-137.4 as compared with the multi-threaded CPU-based 3D SKR algorithm.

Keywords: GPU acceleration; CUDA programming; three-dimensional classical kernel regression; three-dimensional steering kernel regression

前 言

核回归理论的基本思想是通过核回归插值函数进行非参数估计。文献[1]对核回归理论进行了详细的介绍,它广泛应用于图像降噪[2]、图像融合[3]以及图像插值[4]等方面,并取得了十分显著的去噪和插值效果。Takeda等[5]在传统核回归方法(Classic Kernel Regression, CKR)理论的基础上,提出控制核回归方法(Steering Kernel Regression, SKR),并将其应用于图像去噪。与CKR相比,SKR具有更显著的降噪效果、更良好的图像边缘保持效果以及图像插值效果。

核回归理论被广泛地应用于医学图像处理和医学图像重建领域。López-rubio等[6]利用SKR理论提出基于核回归的特征提取降噪方法,并应用于核磁共振成像(Magnetic Resonance Imaging, MRI)三维体数据的降噪。在文献[7]中,三维核回归方法被应用

【收稿日期】2018-08-20
【基金项目】国家自然科学基金(61401451, 61472411, 61501444)
【作者简介】王玉琨,教授,主要研究方向:工程数字化与仿真E-mail: wyk@hpu.edu.cn;刘蓉,在读研究生,主要研究方向:医学影像处理与分析,E-mail: 13781912217@163.com;李凌,高级工程师,主要研究方向:医疗仪器开发,E-mail: ling.li@siat.ac.cn
【通信作者】温铁祥,副研究员,主要研究方向:医学影像处理与分析,E-mail: tx.wen@siat.ac.cn

于Freehand三维超声体数据的重建，该方法能够重建出清晰的图像，同时具有良好的边缘和纹理保持效果，但由于核回归算法需要分别对每个像素点运行核回归插值函数来估计出体素值，所以算法运行时间长，约6 h[8]。文献[9]提出一种基于GPU的实时三维超声增量重建方法。为了使核回归算法能够更好地应用于医学图像重建，基于GPU加速的Freehand三维超声图像的核回归重建算法被提出[10-11]，即采用基于GPU的CUDA编程技术[12]来解决核回归函数运算过程中耗时长的问题，使核回归算法能够满足医学临床的要求。基于GPU的三维核回归超声重建方法比基于CPU的核回归超声重建快200倍左右[11]。在此基础上，文献[13]提出一种基于GPU并行加速自适应核回归三维超声重建算法。因此，基于GPU的核回归算法的研究对三维图像去噪、医学重建等领域具有十分重要的意义。

Takeda等[14]对核回归理论进行了详细的阐述。核回归理论包括CKR算法和SKR算法。相比于CKR算法，SKR算法具有更好的保持边缘效果和去噪效果。SKR算法能够根据局部的结构信息来自适应地调整核回归函数对当前像素点的处理力度，局部的结构信息可通过方向梯度信息进行描述。基于GPU加速的Freehand三维超声图像的核回归重建算法[10-11]是采用基于GPU的三维CKR算法实现体素重建过程，取得了不错的重建效果。由于三维SKR算法具有较大的运算量且并行加速实现较复杂，所以基于GPU的三维SKR并行加速算法的实现仍是一项棘手的问题。

本文优化了三维SKR算法，然后利用CUDA编程实现了基于GPU的三维SKR并行加速算法，并利用不同噪声等级的脑部MRI数据进行实验测试其加速效率。基于GPU的三维SKR并行加速算法的实现为SKR算法应用于医学图像重建提供了可能。

1 三维核回归算法概述

三维核回归理论的数据观察模型如下：

$$Y_i = r(X_i) + \varepsilon_i,\ i = 1, \cdots, L \tag{1}$$

式(1)中，$r(\cdot)$是核回归函数，$X_i = (X_{i0}, X_{i1}, X_{i2})$是体素的三维坐标，$\varepsilon_i$是一个均值为0、方差为$\sigma^2$的高斯噪声，$Y_i$是待处理数据的采样值。

假设待求体素点X离已知数据采样点X_i很近，根据N阶泰勒展开式可得：

$$\begin{aligned} r(X_i) \approx & r(X) + \{\nabla r(X)\}^{\mathrm{T}}(X_i - X) + \frac{1}{2!}(X_i - X)^{\mathrm{T}} \\ & \{\boldsymbol{H} r(X)\}(X_i - X) + \cdots = \beta_0 + \boldsymbol{\beta}_1^{\mathrm{T}}(X_i - X) + \\ & \boldsymbol{\beta}_2^{\mathrm{T}} \mathrm{vech}\{(X_i - X)(X_i - X)^{\mathrm{T}}\} + \cdots \end{aligned} \tag{2}$$

式(2)中，$\nabla r(X)$是求$r(X)$的梯度算子，$\boldsymbol{H}$是求Hessian矩阵算子。

$$\boldsymbol{\beta}_1 = [G_x, G_y, G_z]^{\mathrm{T}} \tag{3}$$

式(3)中，G_x、G_y以及G_z分别是$r(X)$沿着x、y以及z方向上的梯度值。

根据最小化二乘法，对公式(2)进行优化得如下问题：

$$\min_{\{\beta_n\}_{n=0}^{N}} \sum_{i=1}^{M} [Y_i - \beta_0 - \boldsymbol{\beta}_1^{\mathrm{T}}(X_i - X) - \cdots]^2 \cdot K_H(X_i - X) \tag{4}$$

式(4)中，M是邻域窗口内已知像素的个数。

$$K_H(X_i - X) = \frac{1}{\det(\boldsymbol{H}_i)} K\left(\frac{X_i - X}{\boldsymbol{H}_i}\right) \tag{5}$$

式(5)中，$\boldsymbol{H}_i$是平滑矩阵参数；$K_H(\cdot)$是核函数，$K_H(\cdot)$通常为指数函数、高斯函数或其他函数，且需要满足下列条件：

$$\int tK(t)\mathrm{d}t = 0,\ \int t^2 K(t)\mathrm{d}t = c \tag{6}$$

不同的核函数对核回归算法计算的准确性有微小的影响，本文选择可微分且计算复杂度低的高斯核函数。

CKR的光滑矩阵为$\boldsymbol{H}_i = h\boldsymbol{I}$，$h$为全局光滑参数，$\boldsymbol{I}$为单位矩阵。自适应性的SKR的光滑矩阵为$\boldsymbol{H}_i^s = h\boldsymbol{C}_i^{-1/2}$，则其对应的核函数为：

$$K_{H_i^s}(X_i - X) = \sqrt{\frac{\det(\boldsymbol{C}_i)}{(2\pi h^2)^3}} \exp\left\{-\frac{1}{2h^2}\left\|\boldsymbol{C}_i^{1/2}(X_i - X)\right\|_2^2\right\} \tag{7}$$

式(7)中，$\left\|\boldsymbol{C}_i^{1/2}(X_i - X)\right\|_2^2 = (X_i - X)^{\mathrm{T}}\boldsymbol{C}_i(X_i - X)$，$\boldsymbol{C}_i$是对称协方差矩阵，利用图像的梯度信息矩阵$\boldsymbol{J}_i$来进行估算求解，

$$\hat{\boldsymbol{C}}_i^* = \boldsymbol{J}_i^{\mathrm{T}}\boldsymbol{J}_i \tag{8}$$

$$\boldsymbol{J}_i = \begin{bmatrix} G_x(X_1) & G_y(X_1) & G_z(X_1) \\ \vdots & \vdots & \vdots \\ G_x(X_L) & G_y(X_L) & G_z(X_L) \end{bmatrix} \tag{9}$$

根据公式(8)和公式(9)可得：

$$\hat{\boldsymbol{C}}_i^* \approx \begin{bmatrix} \sum_{X_i \in w_s} G_x(X_i)G_x(X_i) & \sum_{X_i \in w_s} G_x(X_i)G_y(X_i) & \sum_{X_i \in w_s} G_x(X_i)G_z(X_i) \\ \sum_{X_i \in w_s} G_x(X_i)G_y(X_i) & \sum_{X_i \in w_s} G_y(X_i)G_y(X_i) & \sum_{X_i \in w_s} G_y(X_i)G_z(X_i) \\ \sum_{X_i \in w_s} G_x(X_i)G_z(X_i) & \sum_{X_i \in w_s} G_y(X_i)G_z(X_i) & \sum_{X_i \in w_s} G_z(X_i)G_z(X_i) \end{bmatrix} \tag{10}$$

式(10)中，w_s是分析窗口的尺寸。

然而，利用公式(10)估算协方差矩阵$\hat{\boldsymbol{C}}_i^*$是不稳定的。一种稳定且有效的估算协方差矩阵$\hat{\boldsymbol{C}}_i$的方法是利用基于奇异值分解(Singular Value

Decomposition, SVD)的方法进行估算[15]。对图像的梯度信息矩阵 $\boldsymbol{J}_i$ 进行奇异值分解可得：

$$\boldsymbol{J}_i=\boldsymbol{U}_i\boldsymbol{S}_i\boldsymbol{V}_i^{\mathrm{T}}=\boldsymbol{U}_i\,\mathrm{diag}\left[\boldsymbol{s}_i^1,\boldsymbol{s}_i^2,\boldsymbol{s}_i^3\right]\left[\boldsymbol{v}_i^1,\boldsymbol{v}_i^2,\boldsymbol{v}_i^3\right]^{\mathrm{T}} \tag{11}$$

式(11)中，$\boldsymbol{S}_i=\mathrm{diag}[\boldsymbol{s}_i^1,\boldsymbol{s}_i^2,\boldsymbol{s}_i^3]$ 为奇异值对角阵，其对角线上的元素为奇异值，$\boldsymbol{V}_i=[\boldsymbol{v}_i^1,\boldsymbol{v}_i^2,\boldsymbol{v}_i^3]$ 为奇异向量矩阵。根据奇异值和奇异向量可求得 $\hat{\boldsymbol{C}}_i$，即

$$\hat{\boldsymbol{C}}_i=\boldsymbol{\mu}_i\cdot\left[\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^1\left(\boldsymbol{v}_i^1\right)^{\mathrm{T}}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^2\left(\boldsymbol{v}_i^2\right)^{\mathrm{T}}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^3\left(\boldsymbol{v}_i^3\right)^{\mathrm{T}}\right] \tag{12}$$

式(12)中，

$$\boldsymbol{\mu}_i=\left(\frac{s_i^1s_i^2s_i^3+\lambda_1}{L}\right)^{\alpha} \tag{13}$$

$$\boldsymbol{\varphi}_i^1=\frac{s_i^1+\lambda_2}{\sqrt{s_i^2s_i^3}+\lambda_2};\boldsymbol{\varphi}_i^2=\frac{s_i^2+\lambda_2}{\sqrt{s_i^1s_i^3}+\lambda_2};\boldsymbol{\varphi}_i^3=\frac{s_i^3+\lambda_2}{\sqrt{s_i^1s_i^2}+\lambda_2} \tag{14}$$

式(13)和(14)中，λ_1 和 λ_2 是规则化参数，能够削减噪声的影响，同时也可预防公式 $\boldsymbol{\varphi}_i$ 计算失效情况的发生。

公式(4)可用矩阵表述为：

$$\hat{\boldsymbol{B}}=\min_{\{b\}}\left\|\boldsymbol{Y}-\boldsymbol{X}_F\boldsymbol{B}\right\|_{\boldsymbol{W}}^2=\min_{\{b\}}(\boldsymbol{Y}-\boldsymbol{X}_F\boldsymbol{B})^{\mathrm{T}}\boldsymbol{W}(\boldsymbol{Y}-\boldsymbol{X}_F\boldsymbol{B}) \tag{15}$$

$$\boldsymbol{Y}=\left[\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots,\boldsymbol{Y}_{\mathrm{M}}\right]^{\mathrm{T}} \tag{16}$$

$$\boldsymbol{B}=\left[\boldsymbol{\beta}_0,\boldsymbol{\beta}_1^{\mathrm{T}},\cdots,\boldsymbol{\beta}_{\mathrm{N}}^{\mathrm{T}}\right]^{\mathrm{T}} \tag{17}$$

$$\boldsymbol{W}=\mathrm{diag}\left[K_H(X_0-X),K_H(X_1-X),\cdots,K_H(X_M-X)\right] \tag{18}$$

式(15)中，$\boldsymbol{Y}$ 是所有体素值组成的向量集，$\boldsymbol{B}$ 是所有 $\boldsymbol{\beta}_i$ 参数组成的向量集，$\boldsymbol{W}$ 是对角矩阵，对角线上的元素为利用CKR核函数 $K_H(\cdot)$ 或SKR核函数 $K_{H_i'}(\cdot)$ 求解的值。

根据最小二乘法，可求得方程(15)的解：

$$\hat{r}(X)=\hat{\beta}_0=\boldsymbol{e}_1^{\mathrm{T}}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y} \tag{19}$$

式(19)中，$\hat{r}(X)$ 为最终计算的体素值，$\boldsymbol{e}_1^{\mathrm{T}}=[1,0,\cdots,0]$，且：

$$\boldsymbol{X}_F=\begin{bmatrix}1 & (\boldsymbol{X}_0-\boldsymbol{X}_i)^{\mathrm{T}} & \mathrm{vech}^{\mathrm{T}}\{(\boldsymbol{X}_0-\boldsymbol{X}_i)(\boldsymbol{X}_0-\boldsymbol{X}_i)^{\mathrm{T}}\} & \cdots\\ 1 & (\boldsymbol{X}_1-\boldsymbol{X}_i)^{\mathrm{T}} & \mathrm{vech}^{\mathrm{T}}\{(\boldsymbol{X}_1-\boldsymbol{X}_i)(\boldsymbol{X}_1-\boldsymbol{X}_i)^{\mathrm{T}}\} & \cdots\\ \vdots & \vdots & \vdots & \cdots\\ & & & \cdots\\ 1 & (\boldsymbol{X}_M-\boldsymbol{X}_i)^{\mathrm{T}} & \mathrm{vech}^{\mathrm{T}}\{(\boldsymbol{X}_M-\boldsymbol{X}_i)(\boldsymbol{X}_M-\boldsymbol{X}_i)^{\mathrm{T}}\} & \cdots\end{bmatrix} \tag{20}$$

式(20)中，vech(·)算子是对称矩阵进行半向量化操作，即：

$$\mathrm{vech}\left(\begin{bmatrix}a & b & c\\ b & d & e\\ c & e & f\end{bmatrix}\right)=\begin{bmatrix}a & b & c & d & e & f\end{bmatrix}^{\mathrm{T}} \tag{21}$$

同理，可得 $\hat{\boldsymbol{\beta}}_1$ 为：

$$\hat{\boldsymbol{\beta}}_1=\begin{bmatrix}\boldsymbol{e}_2^{\mathrm{T}}\\ \boldsymbol{e}_3^{\mathrm{T}}\\ \boldsymbol{e}_4^{\mathrm{T}}\end{bmatrix}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y} \tag{22}$$

式(22)中，$\boldsymbol{e}_2^{\mathrm{T}}=[0,1,\cdots,0]$，$\boldsymbol{e}_3^{\mathrm{T}}=[0,0,1,\cdots,0]$，$\boldsymbol{e}_4^{\mathrm{T}}=[0,0,0,1,\cdots,0]$。

2 基于GPU的SKR并行加速算法的优化

文献[10-11]和文献[13]是利用基于GPU的CUDA编程实现三维CKR并行加速重建算法，并取得了不错的重建效果。然而，三维SKR算法的计算更加复杂且运算量更加庞大，所以实现基于GPU加速的三维SKR算法仍是一项具有挑战性的工作。

三维SKR算法之所以计算复杂且运算量庞大，是由于每个像素点都需要根据它的方向梯度信息来计算一个3×3的协方差矩阵 $\hat{\boldsymbol{C}}_i$，其过程如下：根据公式(11)，对梯度信息矩阵 $\boldsymbol{J}_i$ 进行奇异值分解得到对应的奇异值和奇异值向量；然后，根据公式(13)、公式(14)以及对应的奇异值，计算变量 $\boldsymbol{\varphi}_i^1$、$\boldsymbol{\varphi}_i^2$、$\boldsymbol{\varphi}_i^3$ 以及 $\boldsymbol{\mu}_i$，并利用奇异向量分别求解 $\boldsymbol{v}_i^1\cdot(\boldsymbol{v}_i^1)^{\mathrm{T}}$、$\boldsymbol{v}_i^2\cdot(\boldsymbol{v}_i^2)^{\mathrm{T}}$ 以及 $\boldsymbol{v}_i^3\cdot(\boldsymbol{v}_i^3)^{\mathrm{T}}$ 得到3个3×3的矩阵；最后，利用公式(12)求解协方差矩阵 $\hat{\boldsymbol{C}}_i$。

所有像素点 X_i 都需要计算对应的协方差矩阵 $\hat{\boldsymbol{C}}_i$(3×3)来组成一个庞大的总协方差矩阵 $\hat{\boldsymbol{C}}$，即 $\hat{\boldsymbol{C}}$ 的每一个元素都为一个3×3的 $\hat{\boldsymbol{C}}_i$。所以，利用CPU来实现三维SKR算法需要很长的时间。因此，优化协方差矩阵求解过程是一个亟需解决的问题。

假设第 i 个像素点对应的奇异向量所组成的矩阵 $\boldsymbol{V}_i$ 为：

$$\boldsymbol{V}_i=\begin{bmatrix}\boldsymbol{v}_i^{11} & \boldsymbol{v}_i^{21} & \boldsymbol{v}_i^{31}\\ \boldsymbol{v}_i^{12} & \boldsymbol{v}_i^{22} & \boldsymbol{v}_i^{32}\\ \boldsymbol{v}_i^{13} & \boldsymbol{v}_i^{23} & \boldsymbol{v}_i^{33}\end{bmatrix} \tag{23}$$

为了减少中间矩阵变量的计算量，同时解决总协方差矩阵 $\hat{\boldsymbol{C}}$ 的存储问题，基于GPU的三维SKR算法只求取每个像素点所对应的协方差矩阵 $\hat{\boldsymbol{C}}_i$ 的上三角部分，并将上三角部分的6个元素分别存储在6个总协方差矩阵中，即 $\hat{\boldsymbol{C}}_{11}$、$\hat{\boldsymbol{C}}_{12}$、$\hat{\boldsymbol{C}}_{13}$、$\hat{\boldsymbol{C}}_{22}$、$\hat{\boldsymbol{C}}_{23}$、$\hat{\boldsymbol{C}}_{33}$，例如，$\hat{\boldsymbol{C}}_{23}$ 只存储所有像素点的协方差矩阵的第二行、第三列所对应的元素。因此，根据奇异值向量矩阵 $\boldsymbol{V}_i$、公式(13)以及公式(14)，可将公式(12)分解存储在6个总协方差矩阵中，其计算公式如下：

$$\begin{aligned}
\hat{\boldsymbol{C}}_i^{11}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{11}\boldsymbol{v}_i^{11}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{21}\boldsymbol{v}_i^{21}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{31}\boldsymbol{v}_i^{31})\cdot\boldsymbol{\mu}_i\\
\hat{\boldsymbol{C}}_i^{12}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{11}\boldsymbol{v}_i^{12}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{21}\boldsymbol{v}_i^{22}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{31}\boldsymbol{v}_i^{32})\cdot\boldsymbol{\mu}_i\\
\hat{\boldsymbol{C}}_i^{13}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{11}\boldsymbol{v}_i^{13}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{21}\boldsymbol{v}_i^{23}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{31}\boldsymbol{v}_i^{33})\cdot\boldsymbol{\mu}_i\\
\hat{\boldsymbol{C}}_i^{22}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{12}\boldsymbol{v}_i^{12}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{22}\boldsymbol{v}_i^{22}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{32}\boldsymbol{v}_i^{32})\cdot\boldsymbol{\mu}_i\\
\hat{\boldsymbol{C}}_i^{23}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{12}\boldsymbol{v}_i^{13}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{22}\boldsymbol{v}_i^{23}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{32}\boldsymbol{v}_i^{33})\cdot\boldsymbol{\mu}_i\\
\hat{\boldsymbol{C}}_i^{33}&=(\boldsymbol{\varphi}_i^1\boldsymbol{v}_i^{13}\boldsymbol{v}_i^{13}+\boldsymbol{\varphi}_i^2\boldsymbol{v}_i^{23}\boldsymbol{v}_i^{23}+\boldsymbol{\varphi}_i^3\boldsymbol{v}_i^{33}\boldsymbol{v}_i^{33})\cdot\boldsymbol{\mu}_i
\end{aligned} \tag{24}$$

3 基于GPU的三维SKR算法的实现

基于GPU的CUDA并行加速编程[16-17]被广泛应用于医学图像的处理和三维重建，例如文献[18-19]。因此，基于GPU的CUDA并行加速编程实现三

维SKR算法，不仅能够实现部分体素的并行估计，而且大幅度提高算法的计算效率。与基于GPU的三维CKR算法不同的是需要将基于GPU的三维SKR算法分为两个内核函数（在GPU上运行的CUDA并行计算函数），一个内核函数是计算6个总协方差矩阵，即 $\hat{\boldsymbol{C}}_{11}$、$\hat{\boldsymbol{C}}_{12}$、$\hat{\boldsymbol{C}}_{13}$、$\hat{\boldsymbol{C}}_{22}$、$\hat{\boldsymbol{C}}_{23}$、$\hat{\boldsymbol{C}}_{33}$，另一个是利用总协方差矩阵来估计体素值的内核函数。

针对协方差矩阵内核函数covkernel<<<blocks, thread>>>，定义并存储关键的数组和变量：G_x、G_y以及G_z分别为图像沿x、y以及z方向的方向梯度信息；L为方向梯度信息矩阵的长度；H为方向梯度信息矩阵的高度；D为方向梯度信息矩阵的深度；iSize为输入数据的大小；oSize为输出数据的大小，其定义如下：

$$\begin{aligned}&\text{int iSize} = L \times H \times D \times \text{sizeof(unsignedchar)}\\&\text{int oSize} = L \times H \times D \times \text{sizeof(unsignedchar)}\end{aligned} \tag{25}$$

在协方差矩阵内核函数运行前，需要利用CUDA库中的cudaMalloc(·)函数来为输入数组、输出数组分配内存，例如：

$$\begin{aligned}&\text{cudaMalloc((void**)\&}G_x\text{, iSize)}\\&\text{cudaMalloc((void**)\&}\hat{\boldsymbol{C}}_{11}\text{, oSize)}\end{aligned} \tag{26}$$

然后，利用CUDA库中的cudaMemcpy(·)将输入数据从主机复制到设备中存储。在协方差矩阵内核函数运行结束后，需要将输出数据从设备复制到主机中存储，例如：

$$\begin{aligned}&\text{cudaMemcpy}(G_x, G_x^*, \text{iSize, cudaMemcpyHostToDevice})\\&\text{cudaMemcpy}(\hat{\boldsymbol{C}}_{11}^*, \hat{\boldsymbol{C}}_{11}, \text{oSize, cudaMemcpyDeviceToHost})\end{aligned} \tag{27}$$

式(27)中，G_x^*是主机端沿x方向的梯度信息，$\hat{\boldsymbol{C}}_{11}^*$是$\hat{\boldsymbol{C}}_{11}$所对应主机端的总协方差矩阵，则计算协方差矩阵的核函数伪代码，如表1所示。表1描述的是协方差矩阵内核函数的伪代码。在表1中，第1~6行中所示的代码实现了将线性索引转换成数据阵列的三维逻辑索引；第7行代码是利用方向梯度信息构成梯度信息矩阵；第8行代码是利用辅助函数Extract(J, m, n, k, w_s)获取以(m, n, k)为中心的分析窗口内的数据数组；第9行代码是利用辅助函数SVD($\boldsymbol{J}_{out}$)来求取奇异值和奇异向量；第10~15行代码是利用奇异值和奇异向量求取6个总协方差矩阵；第16行代码是重复执行主循环中循环变量的计算。

针对体素值估计内核函数voxkernel<<<blocks, thread>>>，定义并存储关键的数组和变量：$\boldsymbol{R}_i$为输入数据数组；$\boldsymbol{R}_o$为输出数据数组；L^*为三维重建体的长度；H^*为方向梯度信息矩阵的高度；D^*为方向梯度信息矩阵的深度；iSize*为输入数据的大小；oSize*为输出数据的大小，其定义如下：

$$\begin{aligned}&\text{int iSize}^* = L^* \times H^* \times D^* \times \text{sizeof(unsignedchar)}\\&\text{int oSize}^* = L^* \times H^* \times D^* \times \text{sizeof(unsignedchar)}\end{aligned} \tag{28}$$

表1 协方差矩阵核函数的伪代码

Tab.1 Pseudo code of covariance matrix kernel function

Algorithm1: covkernel<<<blocks,thread>>>

Input: the gradient information matrix: $\boldsymbol{G}_x$, $\boldsymbol{G}_y$, $\boldsymbol{G}_z$, the analysis window size: w_s, the length of $\boldsymbol{G}_x$: L, the height of $\boldsymbol{G}_x$: H, the depth of $\boldsymbol{G}_x$: D;

Output: The total covariance matrixes $\hat{\boldsymbol{C}}_i^{11}$、$\hat{\boldsymbol{C}}_i^{12}$、$\hat{\boldsymbol{C}}_i^{13}$、$\hat{\boldsymbol{C}}_i^{22}$、$\hat{\boldsymbol{C}}_i^{23}$、$\hat{\boldsymbol{C}}_i^{33}$;

Auxiliary function: The function for obtaining the local data array M centered at (m, n, k) with the analysis window size w_s:

$\boldsymbol{M}_{out} = \text{Extract}(\boldsymbol{M}, m, n, k, w_s)$;

The function for the singular value decomposition: $[s,v] = \text{SVD}(\boldsymbol{J}_{out})$;

do {

1. int x = threadIdx.x + blockIdx.x × blockDim.x;
2. int y = threadIdx.y + blockIdx.y × blockDim.y;
3. int offset = $x + y \times \text{blockDim}.x \times \text{gridDim}.x$;
4. $k = \text{offset}/(L \times H)$;
5. $n = (\text{offset} - k \times (L \times H))/L$;
6. $m = \text{offset} - n \times L - k \times (L \times H)$;
7. $G_{xout} = \text{Extract}(G_x, m, n, k, w_s)$;
8. $G_{yout} = \text{Extract}(G_y, m, n, k, w_s)$;
9. $G_{zout} = \text{Extract}(G_z, m, n, k, w_s)$;
10. $\boldsymbol{J}_{out} = [G_{xout}, G_{yout}, G_{zout}]$;
11. $[\boldsymbol{S}_i, \boldsymbol{V}_i] = \text{SVD}(\boldsymbol{J}_{out})$;
12. $\hat{\boldsymbol{C}}_i^{11} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{11} \boldsymbol{v}_i^{11} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{21} \boldsymbol{v}_i^{21} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{31} \boldsymbol{v}_i^{31}\right) \cdot \boldsymbol{\mu}_i$;
13. $\hat{\boldsymbol{C}}_i^{12} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{11} \boldsymbol{v}_i^{12} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{21} \boldsymbol{v}_i^{22} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{31} \boldsymbol{v}_i^{32}\right) \cdot \boldsymbol{\mu}_i$;
14. $\hat{\boldsymbol{C}}_i^{13} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{11} \boldsymbol{v}_i^{13} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{21} \boldsymbol{v}_i^{23} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{31} \boldsymbol{v}_i^{33}\right) \cdot \boldsymbol{\mu}_i$;
15. $\hat{\boldsymbol{C}}_i^{22} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{12} \boldsymbol{v}_i^{12} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{22} \boldsymbol{v}_i^{22} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{32} \boldsymbol{v}_i^{32}\right) \cdot \boldsymbol{\mu}_i$;
16. $\hat{\boldsymbol{C}}_i^{23} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{12} \boldsymbol{v}_i^{13} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{22} \boldsymbol{v}_i^{23} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{32} \boldsymbol{v}_i^{33}\right) \cdot \boldsymbol{\mu}_i$;
17. $\hat{\boldsymbol{C}}_i^{33} = \left(\boldsymbol{\varphi}_i^1 \boldsymbol{v}_i^{13} \boldsymbol{v}_i^{13} + \boldsymbol{\varphi}_i^2 \boldsymbol{v}_i^{23} \boldsymbol{v}_i^{23} + \boldsymbol{\varphi}_i^3 \boldsymbol{v}_i^{33} \boldsymbol{v}_i^{33}\right) \cdot \boldsymbol{\mu}_i$;
18. offset += $\left(\text{blockDim}.x \times \text{gridDim}.x + \text{blockDim}.y \times \text{gridDim}.y\right)$;

} while (offset < $L \times H \times D$)

在体素值估计内核函数运行前，需要利用CUDA库中的cudaMalloc(·)函数来为输入数组、输出数组分配内存，例如：

$$\begin{aligned}&\text{cudaMalloc((void**)}\boldsymbol{R}_i\text{, iSize}^*)\\&\text{cudaMalloc((void**)}\boldsymbol{R}_o\text{, oSize}^*)\\&\text{cudaMalloc((void**)}\hat{\boldsymbol{C}}_{11}\text{, iSize}^*)\end{aligned} \tag{29}$$

然后利用CUDA库中的cudaMemcpy(·)将输入数据从主机复制到设备中存储。在体素估计内核函数运行结束后，需要将输出数据从设备复制到主机中存储，例如：

cudaMemcpy($\hat{\boldsymbol{C}}_{11}$, $\hat{\boldsymbol{C}}_{11}^{*}$, iSize*, cudaMemcpyHostToDevice)
cudaMemcpy($\boldsymbol{R}_i$, $\boldsymbol{R}_i^{*}$, iSize*, cudaMemcpyHostToDevice)
cudaMemcpy($\boldsymbol{R}_o^{*}$, $\boldsymbol{R}_o$, oSize*, cudaMemcpyDeviceToHost)　　(30)

式(30)中，$\hat{\boldsymbol{C}}_{11}^{*}$ 是主机端的一个总协方差矩阵，$\boldsymbol{R}_i^{*}$ 是主机端的输入数据，$\boldsymbol{R}_o^{*}$ 是主机端的输出数据。

表2中，描述的算法2为体素估计核函数的伪代码。第7行代码是利用辅助函数 Extract($\boldsymbol{R}_i$, m, n, k, k_s) 获取以(m, n, k)为中心的SKR核函数内的数据数组；第8行代码是利用6个总协方差矩阵和辅助函数 $K_{H_i^{*}}(\cdot)$ 来求取权重矩阵 $\boldsymbol{W}$；第9行代码是根据公式(19)求取估计的体素值；第10~13行代码是根据公式(22)求取新的梯度信息；第14行代码是重复执行主循环中循环变量的计算。

表2 体素值估计核函数的伪代码

Tab.2 Pseudo code of voxel kernel function

Algorithm 2: voxkernel<<<blocks,thread>>>

Input: the input matrix $\boldsymbol{R}_i$, the feature matrix $\boldsymbol{X}_F$, the SKR kernel size k_s, the length of volume L^{*}, the height of volume H^{*}, the depth of volume D^{*};

Output: the output matrix $\boldsymbol{R}_o$;

Auxiliary functions: The function for obtaining the local data array centered at (m, n, k) with the SKR kernel size k_s:

$\boldsymbol{R}_{out}$ = Extract($\boldsymbol{R}_i$, m, n, k, k_s);

The function for SKR function: $K_{H_i^{*}}(\cdot)$, and the relative parameters are $\hat{\boldsymbol{C}}_i^{11}$、$\hat{\boldsymbol{C}}_i^{12}$、$\hat{\boldsymbol{C}}_i^{13}$、$\hat{\boldsymbol{C}}_i^{22}$、$\hat{\boldsymbol{C}}_i^{23}$、$\hat{\boldsymbol{C}}_i^{33}$;

do {

1. int x = threadIdx.x + blockIdx.x × blockDim.x;
2. int y = threadIdx.y + blockIdx.y × blockDim.y;
3. int offset = x + y × blockDim.x × gridDim.x;
4. k = offset/(L^{*} × H^{*});
5. n = (offset − k × (L^{*} × H^{*}))/L^{*};
6. m = offset − n × L^{*} − k × (L^{*} × H^{*});
7. $\boldsymbol{R}_{out}$ = Extract($\boldsymbol{R}_i$, m, n, k, k_s);
8. $\boldsymbol{W}$ = diag$\left[K_{H_1^{*}}(X_1 - X), \cdots, K_{H_M^{*}}(X_M - X)\right]$;
9. $\boldsymbol{R}_o = \boldsymbol{e}_1^{\mathrm{T}}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y}$;
10. $G_x = \boldsymbol{e}_2^{\mathrm{T}}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y}$;
11. $G_y = \boldsymbol{e}_3^{\mathrm{T}}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y}$;
12. $G_z = \boldsymbol{e}_4^{\mathrm{T}}\left(\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_F\right)^{-1}\boldsymbol{X}_F^{\mathrm{T}}\boldsymbol{W}\boldsymbol{Y}$;
13. $\hat{\boldsymbol{\beta}}_1 = [G_x, G_y, G_z]^{\mathrm{T}}$;
14. offset += (blockDim.x × gridDim.x + blockDim.y × gridDim.y);

} while (offset < L^{*} × H^{*} × D^{*})

4 三维SKR算法CPU与GPU实现效率对比

为了验证基于GPU的三维CKR算法和三维SKR算法的计算效率，实验数据采用从Brainweb database网站下载不同噪声等级的T1w 7%、T1w 9%、T2w 7%、T2w 9%、PD 7%以及PD 9% 6组脑部MRI数据(181×217×181)。如图1所示为实验的T1w、T2w和PD的0%、7%以及9%噪声等级的MRI数据图。

实验运行环境为NVIDIA GeForce 920A、Intel (R) Core(TM)i7- 6500U CPU、Visual Studio 2012和CUDA 8.0库。对上述6组不同噪声级别的脑部MRI数据进行实验，包括基于CPU的单线程的三维CKR算法、基于CPU的多线程(四线程)三维CKR算法、基于GPU的三维CKR算法、基于CPU的单线程三维SKR算法、基于CPU的多线程(四线程)三维SKR算法以及基于GPU的三维SKR算法的实验。

图2为基于GPU的三维CKR算法对6组实验数据的去噪结果，其中图2a和图2d分别是对T1w 7%和T1w 9%噪声等级的CKR去噪结果图；图2b和图2e分别是对T2w 7%和T2w 9%噪声等级的CKR去噪结果图；图2c和图2f分别是对PD 7%和PD 9%噪声等级的CKR去噪结果图。

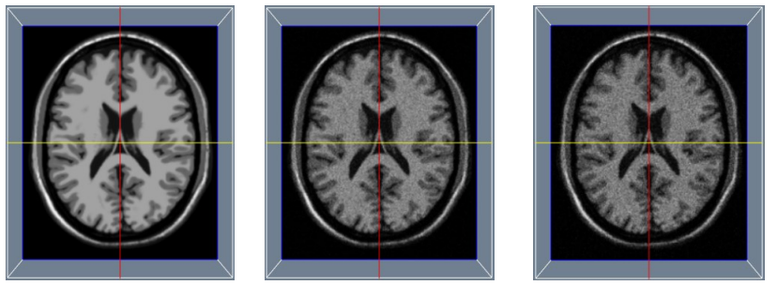
图3为基于GPU的三维SKR算法对6组实验数据的去噪结果，其中图3a和图3d分别是对T1w 7%和T1w 9%噪声等级的SKR去噪结果图；图3b和图3e分别是对T2w 7%和T2w 9%噪声等级的SKR去噪结果图；图3c和图3f分别是对PD 7%和PD 9%噪声等级的SKR去噪结果图。从图2和图3对比可得，基于GPU的三维SKR算法具有更好的去噪效果和边缘保持效果。

6组噪声数据的三维CKR算法的运行时间、加速比统计结果，如表3所示。

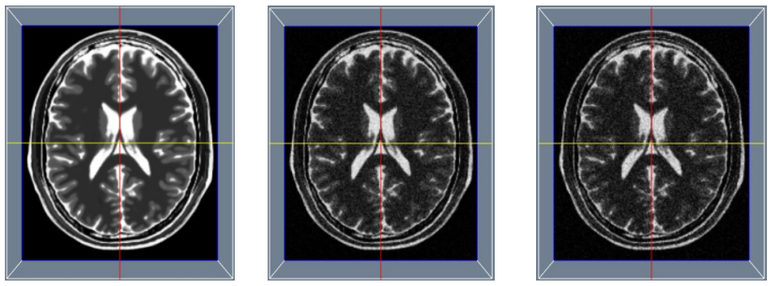
表3给出了6组实验数据在不同处理器条件下的三维CKR算法的实际运行时间。基于GPU的三维CKR算法的处理时间与单线程CPU的三维CKR算法相比加速了约31.3~31.9倍；基于GPU的三维CKR算法的处理时间与多单线程CPU的三维CKR算法相比加速了约26.6~27.9倍。

6组噪声数据的三维SKR算法的运行时间、加速比统计结果，如表4所示。

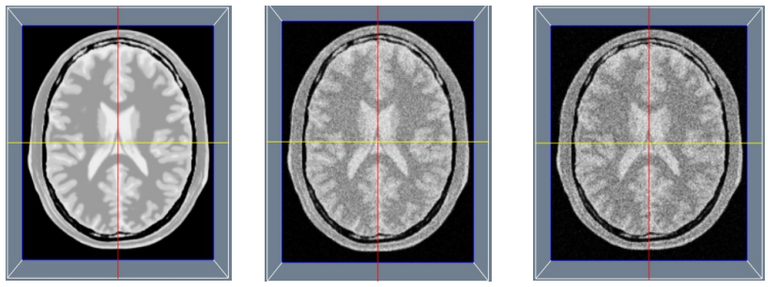
表4给出了6组实验数据在不同处理器条件下的三维SKR算法的实际运行时间，其包括计算协方差矩阵的时间和体素值估计的时间。相比单线程



a: T1w 0% b: T1w 7% c: T1w 9%



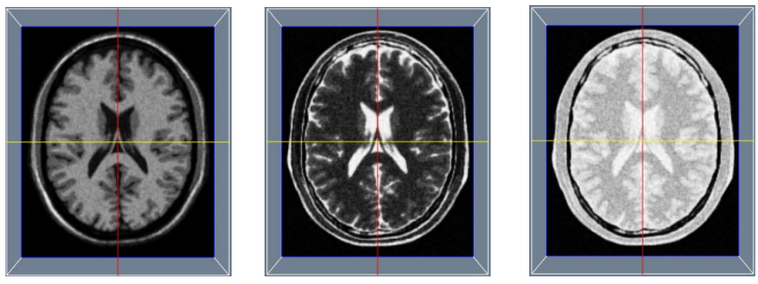
d: T2w 0% e: T2w 7% f: T2w 9%



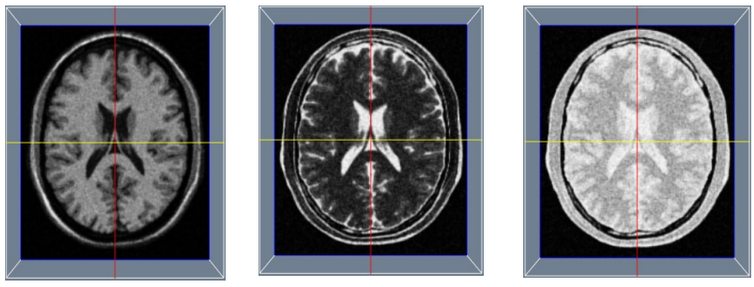
g: PD 0% h: PD 7% i: PD 9%

图1 T1w、T2w、PD不同噪声等级的MRI数据

Fig.1 Magnetic resonance imaging (MRI) data with different noise ratios for T1w, T2w, and PD



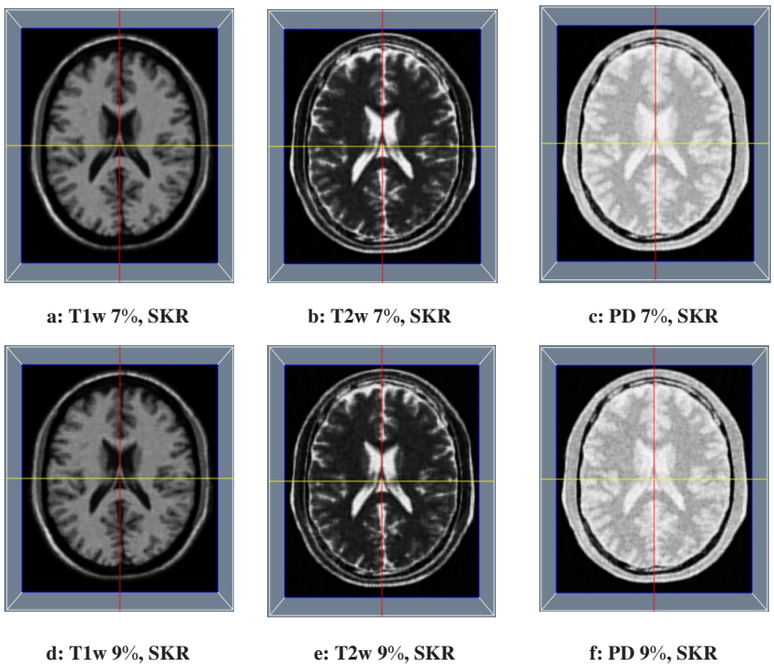
a: T1w 7%, CKR b: T2w 7%, CKR c: PD 7%, CKR



d: T1w 9%, CKR e: T2w 9%, CKR f: PD 9%, CKR

图2 T1w,T2w、PD不同噪声等级的CKR去噪结果

Fig.2 T1w, T2w and PD MRI denoising results by classical kernel regression (CKR) algorithm for different noise ratios



a: T1w 7%, SKR　b: T2w 7%, SKR　c: PD 7%, SKR

d: T1w 9%, SKR　e: T2w 9%, SKR　f: PD 9%, SKR

图3 T1w, T2w, PD不同噪声等级的SKR去噪结果

Fig.3 T1w, T2w and PD MRI de−noising results by steering kernel regression (SKR) algorithm for different noise ratios

表3 基于CPU和GPU的CKR算法运行时间比较

Tab.3 Comparison of running time of CKR algorithms based on CPU and GPU, respectively

Processor	Single-threaded CPU/s	Multi-threaded CPU/s	GPU/s	Single-threaded CPU *vs* GPU	Multi-threaded CPU *vs* GPU
T1w 7%	962.035	813.542	30.555	31.5	26.6
T1w 9%	954.674	845.940	30.469	31.3	27.8
T2w 7%	959.573	832.095	30.332	31.6	27.4
T2w 9%	962.043	825.939	30.144	31.9	27.4
PD 7%	961.306	808.412	30.344	31.7	26.6
PD 9%	960.458	844.640	30.274	31.7	27.9

CPU三维SKR算法，基于GPU的三维SKR算法能获得约244.9~246.3倍的加速比，其中协方差计算模块能获得约283.8~285.4倍的加速比，体素估计模块能获得约236.4~237.9倍的加速比；与多线程CPU三维SKR算法相比，基于GPU的三维SKR算法能获得约123.0~137.4倍的加速比，其中协方差计算模块能获得约205.8~236.2倍的加速比，体素估计模块能获得约104.6~118.5倍的加速比。

5 总结与展望

虽然三维SKR算法比三维CKR算法具有更显著的去噪效果，但基于CPU的三维SKR算法计算时间过于冗长，以至于影响三维SKR算法在三维医学图像处理等领域的应用。本文主要针对三维的SKR算法进行GPU并行加速。本文首先优化了三维SKR算法的计算过程，然后CUDA编程实现了基于GPU的三维SKR算法，从而使计算复杂且运算量大的三维SKR算法可以应用于医学图像处理等领域。实验表明，基于GPU的三维SKR算法能够在几十秒内完成算法的执行，例如，针对T1w、T2w以及PD不同噪声级别的脑部数据，基于GPU的三维SKR算法比基于CPU单线程的三维SKR算法的处理时间减少了约99.59%；基于GPU的三维SKR算法比基于CPU多线程的三维SKR算法的处理时间减少约99.24%。今后将基于GPU的三维SKR算法用于三维Freehand超声重建方面进行深入研究。

表4 基于CPU和GPU的SKR算法运行时间比较

Tab.4 Comparison of running time of SKR algorithms based on CPU and GPU, respectively

Processor	Parameter	Single-threaded CPU/s	Multi-threaded CPU/s	GPU/s	Single-threaded CPU *vs* GPU	Multi-threaded CPU *vs* GPU
T1w 7%	Covariance matrix	2 077.487	1 497.883	7.279	285.4	205.8
	Voxel estimation	7 905.758	3 822.796	33.290	237.5	114.8
	Total time	9 983.245	5 320.679	40.569	246.1	131.2
T1w 9%	Covariance matrix	2 072.164	1 597.965	7.284	284.5	219.4
	Voxel estimation	7 933.067	3 573.158	33.345	237.9	107.2
	Total time	10 005.231	5 171.123	40.629	246.3	127.3
T2w 7%	Covariance matrix	2 070.544	1 511.657	7.284	284.3	207.5
	Voxel estimation	7 887.228	3 487.064	33.346	236.5	104.6
	Total time	9 957.772	4 998.721	40.630	245.1	123.0
T2w 9%	Covariance matrix	2 072.713	1 722.280	7.292	284.2	236.2
	Voxel estimation	7 921.928	3 869.507	33.413	237.1	115.8
	Total time	9 994.641	5 591.787	40.705	245.5	137.4
PD 7%	Covariance matrix	2 066.850	1 616.132	7.282	283.8	221.9
	Voxel estimation	7 901.816	3 952.415	33.355	236.9	118.5
	Total time	9 968.666	5 568.547	40.637	245.3	137.0
PD 9%	Covariance matrix	2 068.846	1 710.717	7.284	284.0	234.9
	Voxel estimation	7 902.312	3 603.154	33.431	236.4	107.8
	Total time	9 971.158	5 313.871	40.715	244.9	130.5

【参考文献】

[1] TAKEDA H, FARSIU S, MILANFAR P. Kernel regression for image processing and reconstruction [J]. IEEE Trans Image Process, 2007, 16(2): 349-366.

[2] WANG J F, CHEN Y P, LI T, et al. A residual-based kernel regression method for image denoising[J]. Math Probl Eng, 2016: 1-13.

[3] 卓国浩, 吴波, 朱欣然. 一种自适应三维核回归的遥感时空融合方法[J]. 武汉大学学报(信息科学版), 2018, 43(4): 563-570.
ZHUO G H, WU B, ZHU X R. Spatio-temporal reflectance fusion based on 3D steering kernel regression techniques[J]. Geomatics and Information Science of Wuhan University, 2018, 43(4): 563-570.

[4] 张峥嵘, 刘红毅, 韦志辉. 边缘保持的核回归图像插值方法[J]. 计算机工程, 2011, 37(19): 194-197.
ZHANG Z R, LIU H Y, WEI Z H. Edge-preserved kernel regression image interpolation method[J]. Computer Engineering, 2011, 37(19): 194-197.

[5] TAKEDA H, FARSIU S, MILANFAR P. Deblurring using regularized locally adaptive kernel regression [J]. IEEE Trans Image Process, 2008, 17(4): 550-563.

[6] LÓPEZ-RUBIO E, FLORENTÍN-NÚÑEZ M N. Kernel regression based feature extraction for 3D MR image denoising [J]. Med Image Anal, 2011, 15(4): 498-513.

[7] 范阳阳, 倪倩, 温铁祥, 等. Freehand三维超声体数据重建的最新研究进展[J]. 中国生物医学工程学报, 2017, 36(1): 92-102.
FAN Y Y, NI Q, WEN T X, et al. The latest research progress of Freehand 3D US volume reconstruction[J]. Chinese Journal of Biomedical Engineering, 2017, 36(1): 92-102.

[8] CHEN X K, WEN T X, LI X M, et al. Reconstruction of freehand 3D ultrasound based on kernel regression[J]. Biomed Eng Online, 2014, 13(1): 124-138.

[9] DAI Y, TIAN J, DONG D, et al. Real-time visualized freehand 3D ultrasound reconstruction based on GPU [J]. IEEE Trans Inf Technol Biomed, 2010, 14(6): 1338-1345.

[10] WEN T X, LI L, ZHU Q S, et al. GPU- accelerated kernel regression reconstruction for freehand 3D ultrasound imaging[J]. Ultrason Imaging, 2017, 39(4): 240-259.

[11] WEN T X, LIU R, LIU L, et al. GPU-based volume reconstruction for freehand 3D ultrasound imaging[J]. Conf Proc IEEE Eng Med Biol Soc, 2017: 3700-3703.

[12] SANDERS J. GPU高性能编程CUDA实战[M]. 北京: 机械工业出版社, 2011.
SANDERS J. CUDA by example: an introduction to general-purpose GPU programming[M]. Beijing: China Machine Press, 2011.

[13] WEN T X, YANG F, GU J. An adaptive kernel regression method for 3D ultrasound reconstruction using speckle prior and parallel GPU implementation[J]. Neurocomputing, 2017, 275: 208-223.

[14] TAKEDA H, MILANFAR P, PROTTER M, et al. Super-resolution without explicit subpixel motion estimation[J]. IEEE Trans Image Process, 2009, 18(9): 1958-1975.

［15］FENG X G, MILANFAR P. Multiscale principal components analysis for image local orientation estimation［C］//Signals, Systems and Computers, Conference Record of the Thirty-Sixth Asilomar Conference on. IEEE, 2002: 478-482.

［16］王泽寰, 王鹏. GPU并行计算编程技术介绍［J］. 科研信息化技术与应用, 2013, 4(1): 81-87.
WANG Z H, WANG P. Introduction to GPU parallel programming technology［J］. E-science Technology & Application, 2013, 4(1): 81-87.

［17］张澳博. GPU并行计算分析［J］. 数字通信世界, 2017(9): 39-40.
ZHANG A B. GPU parallel computing analysis［J］. Digital Communication World, 2017(9): 39-40.

［18］翟争峯, 蒲立新, 曲建明, 等. 基于GPU的医学图像三维重建体绘制技术综述［J］. 中国数字医学, 2015, 10(4): 11-13.
ZHAI Z F, PU L X, QU J M, et al. Overview of volume rendering technology of 3D reconstruction of medical image based on GPU［J］. China Digital Medicine, 2015, 10(4): 11-13.

［19］ZHAO X, HU J J, YANG T. GPU based iterative cone-beam CT reconstruction using empty space skipping technique［J］. J X-ray Sci Technol, 2013, 21(1): 53-69.

（编辑：薛泽玲）