基于GPU的压缩感知重构算法的设计与实现

张 静[1] 熊承义[1] 高志荣[2]
(中南民族大学电子信息工程学院 武汉 430074)[1] (中南民族大学计算机科学学院 武汉 430074)[2]

摘 要 针对大尺度压缩感知重构算法实时性应用的需要,探讨了基于图形处理器(GPU)的正交匹配追踪算法(OMP)的加速方法及实现。为降低中央处理器与GPU之间传输的高延迟,将整个OMP算法的迭代过程转移到GPU上并行执行。其中,在GPU端根据全局存储器的访问特点,改进CUDA程序使存储访问满足合并访问条件,降低访问延迟。同时,根据流多处理器(SM)的资源条件,增加SM中共享存储器的分配,通过改进线程访问算法来降低bank conflict,提高访存速度。在NVIDIA Tesla K20Xm GPU和Intel(R) E5-2650 CPU上进行了测试,结果表明,算法中耗时长的投影模块、更新权值模块分别可获得32和46倍的加速比,算法整体可获得34倍的加速比。

关键词 压缩感知重构,正交匹配追踪,图形处理器,并行执行,加速
中图法分类号 TP391.41 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.8.065

Implementation for Compressed Sensing Reconstruction Algorithm Based on GPU

ZHANG Jing[1] XIONG Cheng-yi[1] GAO Zhi-rong[2]
(College of Electronic and Information Engineering, South-Central University for Nationalities, Wuhan 430074, China)[1]
(College of Computer Science, South-Central University for Nationalities, Wuhan 430074, China)[2]

Abstract Aiming at the need of real-time application of large scale compressed sensing reconstruction algorithm, the acceleration method and implementation of the Orthogonal Matching Pursuit (OMP) algorithm based on Graphic Processing Unit (GPU) was discussed. In order to reduce the high latency of transmission between the central processing unit and GPU, the iterative process of the whole OMP algorithm is transferred to the GPU for parallel execution. According to the access characteristics of global memory, the CUDA program is improved in graphic processing unit, which makes the storage access meet the combined access conditions, and reduces the access delay. At the same time, according to the resource conditions of the Streaming Multiprocessor (SM), the allocation of shared memory is increased in SM. In addition, the bank conflict is reduced by improving the threads access algorithm to increase the memory access speed. Tests on the NVIDIA Tesla K20Xm GPU and Intel (R) E5-2650 CPU show that the time consuming projection module and the updated weight module can get 32 and 46 times of speedup ratio respectively, and the whole algorithm can achieve 34 times of speedup.

Keywords Compressed sensing reconstruction, Orthogonal matching pursuit, Graphic processing unit, Parallel execution, Acceleration

1 引言

近年来,压缩感知(Compressed Sensing,CS)理论[1]在无线通信、阵列信号处理、雷达成像、生物传感等领域得到广泛应用,它利用信号的稀疏特性,在远小于奈奎斯特采样率的条件下,用随机采样获取信号的离散样本,然后通过非线性重建算法恢复出原始信号[2]。压缩感知理论包含3个关键要素:稀疏表示、压缩测量和非线性优化重建[3],其中,信号准确重构是该理论的核心问题。压缩感知在采样编码端容易实现,但却以重构解码端的复杂度为代价。凸优化算法(BP算法、迭代硬阈值法)和贪婪算法(正交匹配追踪算法、子空间追踪算法)[4]作为目前两类主流的重构算法,在进行二维、三维图像重构时,其计算量都十分巨大,特别在进行大规模信号重构时,会存在实时性差的问题,限制了压缩感知的广泛应用。因此,在重构算法的大尺度实时应用上,加速研究具有重要意义。

随着通用图形处理器(General Purpose GPU,GPGPU)的推行,最初用于图像加速、处理图形转换、渲染等工作的GPU在高性能计算方面得到大量的应用,在对复杂算法的处理上,GPU平台比FPGA表现更优。近年来,国内外许多研究者在GPU与各领域结合方面展开了广泛的研究,并取得了良好的进展。其中,在医疗成像、图像处理和压缩传感领域

到稿日期:2015-07-10 返修日期:2015-10-15 本文受国家自然科学基金资助项目(61471400,61201268),湖北省自然科学基金资助项目(2013CFC118),中央高校基本科研业务费专项(CZW14018)资助。

张 静(1989-),男,硕士生,主要研究方向为图像处理、压缩感知,E-mail:13554522280@163.com;熊承义(1969-),男,教授,硕士生导师,主要研究方向为图像压缩、小波变换、压缩感知、人脸识别,E-mail:xiongcy@mail.scuec.edu.cn;高志荣(1972-),女,博士生,副教授,主要研究方向为模式识别、图像处理。

的GPU加速研究引起了国内外学者的密切关注。在文献[5]中，俄亥俄州立大学的Klaus Mueller和Roni Yagel为了满足医学上临床应用的需要，最早使用图形硬件加速代数重建算法，通过对算法数据量预测和图像投影两个主要操作的分析，在两种量分解模式(体素和切片)下使用图形硬件对该算法进行加速，并取得了显著的加速比。在此之后，GPU强劲的并行处理能力成为了学术界和工业界的研究热点。国防科技大学的卢凯等人[6]在计算机图形处理方面，针对显著区域检测算法计算复杂度高的问题，采用GPU加速原串行算法，并获得了较好的加速比。随着压缩感知理论的发展，GPU与压缩感知的结合很快成为了国内外学者研究的课题。针对压缩感知OMP重构算法耗时长的问题，文献[7]提出了基于GPU的并行重构算法，其利用GPU的强大运算能力，在对现有算法进行优化的同时，将重构算法中复杂的矩阵操作模块转移到GPU上并行执行，与原算法相比，得到了3.5倍左右的加速比。此外，在基于压缩感知理论的稀疏磁共振图像(Magnetic Resonance Imaging，MRI)重构算法的基础上，文献[8]在NIVIDIA CUDA的框架上对正交匹配追踪算法进行了并行化的设计与实现，结果表明，基于GPU实现的算法具有较高的重构速度，1024^2大小的磁共振图像的重构时间对比CPU获得了24倍的加速比。对于国外的相关工作，美国马里兰大学巴尔的摩县分校的Amey Kulkarni和Tinoosh Mohsenin[9]在基于压缩感知的OMP重构算法上，提出了一种高效的并行架构，并在7种不同的平台(FPGA、CPU、通用GPU等)上对该架构进行了仿真实验，结果表明，该架构较好地提高了OMP重构效率。

在以往研究的基础上，本文进一步探讨了基于GPU的正交匹配追踪算法的加速方法，并结合CUDA编程特点，对GPU优化实现进行了研究。为降低中央处理器与GPU之间传输的高延迟，将整个OMP算法的迭代过程转移到GPU上并行执行。其中，在GPU端，根据全局存储器的访问特点，改进CUDA程序，使存储访问满足合并访问条件，降低访问延迟。同时，根据流多处理器(SM)的资源条件，增加SM中共享存储器的分配，通过改进线程访问算法来降低bank conflict，提高访存速度。在NVIDIA Tesla K20Xm GPU和Intel(R)E5-2650 CPU上进行了测试，结果表明，算法中耗时长的投影模块、更新权值模块分别可获得32和46倍的加速比，算法整体可获得34倍的加速比。

2 压缩感知及OMP重构

设$X\in R^n$是一维离散信号，$\Psi\in R^{n\times n}$为正交变换矩阵，其中R代表实数集。假设信号X满足$X=\Psi\times Z$并且Z只包含$k\ll n$个非零元素，那么认为信号X在Ψ域是K-稀疏的。用测量矩阵$\Phi\in R^{m\times n}(k<m<n)$对信号$X$采样可得测量向量：

$$Y=\Phi X=AZ\in R^m \tag{1}$$

其中，称$A=\Phi\Psi$为感知矩阵。

如果测量矩阵Φ服从有限等距特性(RIP)并且与变换矩阵Ψ具有较低的相关性，那么就可以从低维度的测量向量Y中有效地重构出高维度的稀疏向量Z，从而得到原始信号X[10,11]。设矩阵$A=(a_1,a_2,\cdots,a_n)$，$A_t=(a_{m1},a_{m2},\cdots,a_{mt})$，权值向量$U=(u_1,u_2,\cdots,u_t)$，其中$a_i\in R^m$表示感知矩阵$A$的第$i$列，$a_{mt}\in R^m$表示算法进行第$t$次迭代时在感知矩阵$A$中选择的第$t$个列向量，$u_1,u_2,\cdots,u_t$表示算法在第$t$次迭代时更新的$t$个加权值，$R$代表实数集。OMP算法的迭代过程可分解为如下4个模块，以下对每个模块的时间复杂度进行分析。

I)投影：计算$\frac{\langle r_{t-1},a_i\rangle}{\|a_i\|_2}$。本模块负责计算残差向量$r$($r$的初始值为$Y$)在原子$a_i$上的投影，投影值作为下个模块的输入。其中$\langle r,a_i\rangle$代表残差向量$r$和原子向量$a_i$的内积，$\|\cdot\|_2$表示向量的$\ell_2$范数。本模块主要耗时部分是$(n\times m)\times(m\times 1)$的矩阵向量乘法操作，因此，本模块的复杂度为$O(mn)$。

II)选择最佳匹配原子：从模块I中得到的n个投影值中确定绝对值最大的投影值的索引号，通过索引号选择感知矩阵A中的原子(即为最佳匹配原子a_{mt})，并且记录索引号，选择出的最佳匹配原子和索引号作为下个模块的输入。由于要从n个原子中选择一个最佳匹配原子，因此，本模块的复杂度大约为$O(n)$。

III)更新权值：由最小二乘法得$U_t=\arg\min_U\|Y-A_tU\|_2$，该步骤可简化为$U_t=(A_t^TA_t)^{-1}A_t^TY$。本模块负责计算权值向量$U_t$的值，向量$U_t$的解作为下个模块的输入，当迭代完成时，权值向量$U_k$即为稀疏向量$Z$的最优近似解。本模块需要计算4个操作，$A_t^T\times A_t$操作的复杂为$O(t^2m^2)$，$A_t^T\times Y$操作的复杂度为$O(tm^2)$，$(A_t^TA_t)^{-1}$操作的复杂度为$O(t^3)$，$(A_t^TA_t)^{-1}\times A_t^TY$操作的复杂度为$O(t^2)$。

IV)更新残差：计算残差向量$r_t=Y-A_tU_t$。其中A_t可由模块II获得，残差向量r_t为下次迭代中的模块I做准备。本模块的主要耗时部分是$(m\times t)\times(t\times 1)$的矩阵向量乘法操作。因此，本模块的复杂度为$O(mt)$。

通过以上分析可知，当k(代表稀疏向量Z中非零元素的个数)较小时，OMP算法的复杂度主要在投影模块，即$O(mn)$。然而，在k比较大的情况下，OMP算法的复杂度主要在更新权值中的矩阵求逆操作，即$O(k^3)$。较容易看出，OMP算法的瓶颈在于矩阵求逆操作和矩阵向量乘法操作，因此，下文将主要介绍矩阵求逆和矩阵向量乘在GPU上的加速设计。

3 基于GPU的加速算法设计

3.1 GPU并行模型

CUDA(Compute Unified Device Architecture)是NVIDIA公司在2007年推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题[12]。CUDA编程模型将CPU作为主机(Host)，GPU作为协处理器(co-processor)或者设备(Device)，CPU与GPU协同工作，CPU负责进行逻辑性强的事务处理和串行计算，GPU则专注于执行高度线程化的并行处理任务。运行在GPU上的程序称为kernel(内核函数)，kernel以线程网格(Grid)的形式组织，每个线程网格由若干个线程块(block)组成[13]。实质上，kernel是以block为单位执行的，一个kernel函数中存在两个层次的并行，即Grid中的不同block间存在不需要通信的粗粒度并行和block中的thread间形成允许通信的细粒度并行[14]。如图1所示，一个完整的CUDA程序是由一系列的设备端kernel函数的并行处理步骤和主机端的串行处理步骤共同组成的。

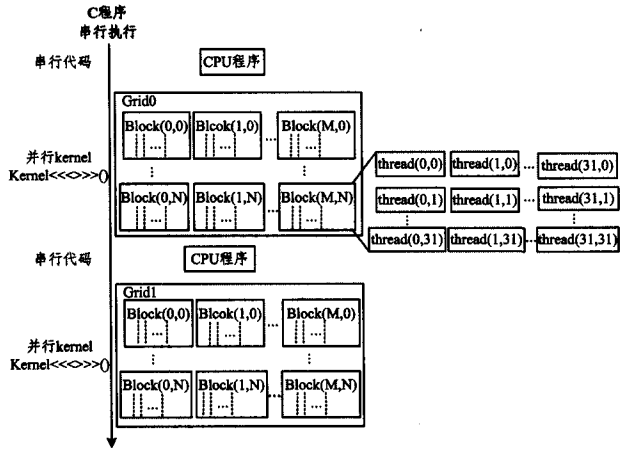

图 1　CUDA 编程模型

除了编程模型和执行模型，CUDA 也规定了存储器模型：每个线程拥有自己的私有寄存器和局部存储器；每一个线程块拥有一块共享存储器（shared memory）；grid 中所有的线程都可以访问同一块全局存储器（global memory）。寄存器和共享存储器是 GPU 片上高速存储器，其访问速度远高于全局存储器。为了能够在并行访问时获得高带宽，共享存储器被划分为大小相等且能被同时访问的存储器模块，该模块被称为 bank。

3.2　OMP 并行设计

由上文可知，OMP 算法的主要耗时部分在投影模块中的矩阵向量乘操作和更新权值模块中的矩阵求逆操作，因此，本文主要针对矩阵向量乘操作和矩阵逆操作进行加速设计。考虑到 CPU 与 GPU 数据传输的高延迟性，为了减少主机内存与设备内存之间的数据传输，将算法的 4 个计算模块全部放在 GPU 上执行，如图 2 所示。

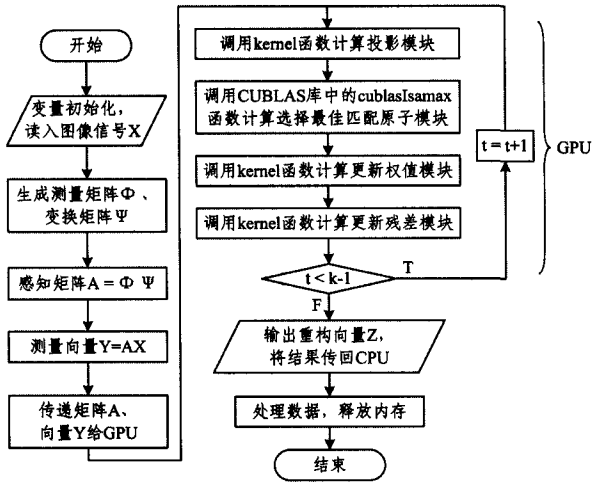

图 2　OMP 加速算法流程图

下面将阐述每个模块的具体设计方案。

3.2.1　投影模块

在执行 $A^{\mathrm{T}}\times r$ 操作时，向量 r 可看作与矩阵 A^{T} 中的每一行进行内积运算，利用这个特点，通过把向量 r 的数据存放在流多处理器（Streaming Multiprocessors，SM）的共享内存中来降低数据传输延迟，达到提高运算效率的目的。但是，本实验显卡 NVIDIA Tesla K20Xm 的每个 SM 上只有 48kB 的共享存储器空间，这使得空间容量满足不了残差向量 r 全部数据的存储，于是本文设计出一种块并行算法。特别地，线程在访问共享存储器时，half-warp 请求访问的多个地址位于同一个 bank 中，就会出现 bank conflict，导致访问效率降低。因此，在线程访问设计部分，针对 bank conflict 的问题，让同一个 warp 中的线程访问不同的 bank，减少 bank conflict，达到提高访存速度的目的。

设向量 $q=A^{\mathrm{T}}\times r$，为了并行化求解向量 q 的值，本文采取用每个线程块计算向量 q 中 32 个元素的方法，并且每个线程块分配 32×32 个线程，那么矩阵 A^{T} 可被分成许多个 32×1024 大小的矩阵块，对应地，残差向量 r 会被切割成许多长度为 1024 的段，如图 3 所示，i 表示线程块的序号数，j 表示第 i 个线程块进行的第 j 次循环。然后，在 SM 上的共享内存中分配两个大小为 4kB（1024 个单精度浮点数）的内存空间，一个用于从全局内存中读取向量 r 的每个段，另一个用于存储中间结果。该并行算法的执行过程为：每个线程执行 $m/1024$ 次循环，每一次循环中，每个线程负责从向量 r 中读取 1 个元素到共享内存并从矩阵 A^{T} 中读取 32 个元素到寄存器；然后每个线程负责累加向量 r 段中元素和矩阵块中元素的乘积并将中间结果存放在共享内存中；最后，对中间结果作 CUDA 归约求和运算并获得向量 q 的值。

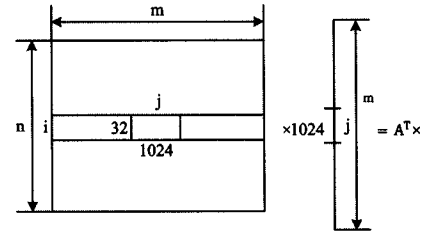

图 3　矩阵向量乘的块并行算法

3.2.2　选择最佳匹配原子模块

由于 CPU 与 GPU 之间的数据传输存在高延迟性，为了减少主机与设备间的数据传输，通过模块 Ⅰ 中得到的投影值，直接调用 CUDA 内部的 CUBLAS 库中的 cublasIsamax 函数来完成并行设计，函数返回的最佳匹配原子的索引号将作为下一模块的输入。

3.2.3　更新权值模块

分析可知，本模块由矩阵逆和矩阵向量乘积两个操作组成，而矩阵向量乘积运算的并行设计可以参考模块 I 的设计思路。接下来，针对矩阵逆 $(A_t^{\mathrm{T}}A_t)^{-1}$ 的特点，采用矩阵逆更新算法取代并行性差的 LU 分解方法，完成矩阵逆的并行设计。特别地，在 CUDA 并行设计中，本模块对全局存储器的访问频繁，于是，全局存储器的访问优化对本模块的加速效果起到了重要的作用。另外，在编写 CUDA 程序时，对全局内存的访问是否满足合并访问条件是对 CUDA 程序性能影响最明显的因素之一。因此，在设计线程访问算法时，尽量让一个 warp 中的线程访问一个连续的全局内存块，实现合并访问，有效地降低访问延迟。

考虑到在 OMP 算法的每次迭代中感知矩阵 A 中的一列被增加到增量矩阵 A_t 中，根据 Sherman-Morrison-Woodbury[15] 公式，$(A_t^{\mathrm{T}}A_t)^{-1}$ 的计算可以采用迭代的思想进行且可并行化实现，其中 $A_t=(a_{m1},a_{m2},\cdots,a_{mt})$，$a_{mt}$ 表示算法第 t 次迭代时在感知矩阵 A 中选择的第 t 个列向量。在第 $t-1$ 次迭代中，A_{t-1} 表示大小为 $m\times(t-1)$ 的矩阵，并假设矩阵 $B_{t-1}=(A_{t-1}^{\mathrm{T}}A_{t-1})^{-1}$ 已经被求得，那么在第 t 次迭代中，感知矩阵 A

中的一列 a_{mt} 会被增加到矩阵 A_{t-1} 中，即 $A_t=(A_{t-1}\ \ a_{mt})$。在给出矩阵 B_{t-1} 的条件下，采用矩阵逆更新算法[16]，矩阵 $B_t=(A_t^TA_t)^{-1}$ 的值可由式(2)较容易地得到：

$$B_t=\begin{pmatrix} F & -dB_{t-1}A_{t-1}^Ta_{mt} \\ -da_{mt}^TA_{t-1}B_{t-1}^T & d \end{pmatrix} \tag{2}$$

其中，$F=B_{t-1}+dB_{t-1}A_{t-1}^Ta_{mt}a_{mt}^TA_{t-1}B_{t-1}^T$，$d=1/(\|a_{mt}\|_2^2-a_{mt}^TA_{t-1}B_{t-1}A_{t-1}^Ta_{mt})$。

令 $u_1=A_{t-1}^Ta_{mt}$，$u_2=B_{t-1}u_1=B_{t-1}A_{t-1}^Ta_{mt}$，那么 $F=B_{t-1}+du_2u_2^T$，$d=1/(\|a_{mt}\|_2^2-u_1^Tu_2)$，矩阵 B_t 的求解可以简化为：

$$B_t=\begin{pmatrix} F & -du_2 \\ -du_2^T & d \end{pmatrix} \tag{3}$$

由式(3)可知，矩阵逆的计算可以简化为多个矩阵向量乘运算和矩阵求和运算，其中矩阵向量乘法操作同样可以参照模块Ⅰ的设计思路，而矩阵求和操作较容易实现并行化。在GPU上，为了计算矩阵 $B_t=(A_t^TA_t)^{-1}$，本文的CUDA程序调用了8个kernel函数。第1个kernel函数计算向量 u_1，第2个kernel函数计算向量 u_2，第3个kernel函数计算 $u_1^Tu_2$ 并返回 d 的值，接下来用2个kernel函数分别计算 $du_2u_2^T$ 和 $B_{t-1}+du_2u_2^T$ 并返回矩阵 F，其它3个kernel函数用来得到矩阵 B_t。

3.2.4 *更新残差模块*

本模块计算的复杂度主要在于矩阵向量乘法运算，该运算的并行设计可以用到模块Ⅰ的方法。为了在GPU上执行本模块的运算，只需要调用一个kernel函数实现矩阵向量乘操作和向量求差操作，最后将求得的残差作为下一次迭代中模块Ⅰ的输入。

4 并行实现与结果

4.1 OMP算法的并行实现

为了在GPU上执行OMP加速算法，首先应在CPU内存上分配变量空间并初始化输入、输出变量，相应地，在GPU上分配输入变量 Y(长度为 m 的随机测量向量)、输入变量 A^T(大小为 $m\times n$ 的感知矩阵 A 的转置)、输出变量 T(长度为 n 的向量)、辅助变量 I(长度为 k 的向量，用来记录最佳匹配原子的索引号)和其它中间变量。在执行OMP并行算法之前，矩阵 A^T 和向量 Y 从CPU上的主机内存中传输到GPU上的全局内存中。OMP并行算法执行完成后，输出变量 T 从GPU的全局内存传回到主机内存空间。

如图4所示，OMP算法的并行实现包含 k 次迭代。每次迭代的具体过程如下：首先，调用一个kernel函数计算 $A^T\times r$，部分kernel代码见图5；其次，为了在感知矩阵 A 中确定最佳匹配原子，可调用CUDA自身的CUBLAS库中的cublasIsamax函数；再次，可以由上文给出的矩阵逆更新并行算法调用8个kernel函数计算矩阵 $B_t=(A_t^TA_t)^{-1}$，另外，$A_t^T\times Y$ 运算可以认为 $A^T\times Y$ 是运算的子运算，由于向量 Y 和矩阵 A^T 在算法的每次迭代中都是不变的，因此 $A^T\times Y$ 的值可以在算法执行的第一次迭代时计算出来并存放在GPU上的中间变量中，$A_t^T\times Y$ 的值就可以容易地从该变量中获得；最后，调用一个kernel函数实现矩阵 B_t 和向量 A_t^TY 的乘积运算，调用另一个kernel函数更新残差向量 r 的值。

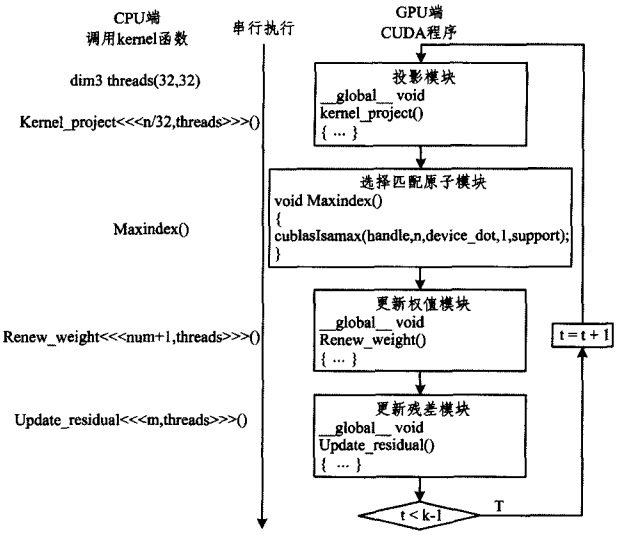

图4 迭代过程并行设计模型

```
__global__ void kernel_project(…)
{
    …
    线程配置
    …
    for(int i=0;i<m/1024;i++)
    {
        …
        for(int k=0;k<32;k++)
        {
        ps[tdy][tdx]+=device_subA[bid * 32 * m+tdy * m+tdx+
                        32 * k+i * 1024] * xs[tdx+32 * k]/device_
                        l2[bid * 32+tdy];
        }
    }
      …
      归约求和
    …
    if(tdx==0)
      device_dot[bid * 32+tdy]=ps[tdy][0];
}
```

图5 投影模块kernel函数

4.2 测试及结果分析

为了测试本文实现的系统性能，在选取不同信号维度和稀疏度的条件下，对系统的运行时间和加速效率两方面性能进行了测试和比较。实验硬件如下：主机端采用Intel(R) E5-2650 CPU和4GB主机内存，设备端采用NVIDIA Tesla K20Xm GPU和4GB全局内存。另外，为了与文献[7]的结果进行对照比较，本次实验选取2幅不同尺寸(128 * 128、128 * 64)的图像信号。测量矩阵 Φ 采用大小为 $m\times n$ 的高斯随机矩阵，稀疏矩阵 Ψ 是大小为 $n\times n$ 的离散余弦变换矩阵，那么感知矩阵 $A=\Phi\Psi$ 的大小为 $m\times n$。其中，n 为信号的长度，m 是测量向量的维度，k 代表稀疏向量 Z 中非零元素的个数。对OMP算法中耗时长的投影模块，表1给出了该模块在CPU和GPU上执行的性能比较。对更新权值模块中的矩阵求逆部分，由于影响该求逆运算性能的关键因素是 k 的值，因此，为了测试求逆运算的加速效率，在 $n=8192$，$m=1024$ 的条件下选取8组不同的 k 值在CPU和GPU上对矩阵逆更新算法的性能进行测试，GPU对比CPU的加速效果如图6所示。图6表明，在稀疏度较高的条件下，矩阵求逆并行算法

对比串行算法可以获得较高的加速比。对于OMP算法的整体加速效果，表2给出了测试结果，并将数据结果与文献[7]进行对照；表3在重建时间和加速比上与文献[8]进行了3组数据的比较。由于无法直接与文献[9]进行对比，只给出了与文献[7]、文献[8]比较的结果。另外，由于文献[8]是基于MATLAB平台的GPU加速，而本文是在Microsoft Visual Studio平台上进行的GPU加速设计，因此表3中文献[8]的执行时间会比本文短，但对于加速性能，本文表现出了更突出的优势。表1、表2和表3的结果表明，信号长度n、测量向量维度m和k值是影响加速效率的3个关键因素，当信号长度n越大、测量向量维度m越长、k值越高时，GPU对于CPU的加速效果越明显。本次实验结果证实了所提算法的有效性，其大幅度地提高了图像信号的重构效率。

表1 投影模块在CPU和GPU上的性能比较

n	m	k	CUP执行时间(s)	GPU执行时间(s)	加速比
16384	4096	512	47.12	1.45	32.50
16384	2048	256	11.84	0.37	32.00
16384	1024	128	2.96	0.10	29.60
8192	2048	256	5.80	0.20	29.00
8192	1024	128	1.44	0.05	28.80
8192	512	64	0.33	0.03	11.00

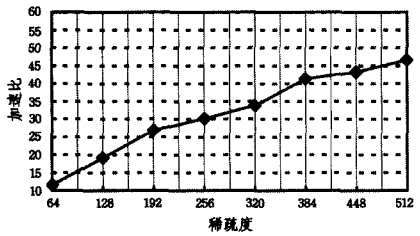

图6 更新算法的GPU加速效率

表2 OMP整体加速结果对比

n	m	k	CPU执行时间(s)	GPU执行时间(s)	文献[7]加速比	本文加速比
16384	4096	512	79.03	2.31	3.50	34.21
16384	2048	256	15.81	0.62	3.52	25.50
16384	1024	128	3.91	0.30	3.59	13.03
8192	2048	256	9.54	0.44	3.45	21.68
8192	1024	128	2.05	0.24	3.41	8.54
8192	512	64	0.49	0.14	3.31	3.50

表3 加速结果对比

图像大小	CPU重建时间(s)		GPU重建时间(s)		加速比	
	文献[8]	本文	文献[8]	本文	文献[8]	本文
32^2	0.052	0.103	0.270	0.152	0.19	0.68
64^2	0.145	0.997	0.263	0.231	0.55	4.32
128^2	0.416	79.026	0.266	2.312	1.56	34.17

结束语 针对大规模信号重构时存在的实时性差的问题，探讨了基于图形处理器的正交匹配追踪算法的加速方法及实现。在GPU优化部分，通过改进CUDA程序，使存储访问满足合并访问条件，提高了访存速度。同时，根据流多处理器的资源条件，通过改进线程访问算法来降低内存冲突，提高了OMP算法的性能。实验结果说明，对于密集度高的并行数据，GPU能获得可观的加速效果。

参考文献

[1] Donoho D L. Compressed sensing [J]. IEEE Transactions on Information Theory (S0018-9448), 2006, 52(4): 1289-1306

[2] Shao Wen-ze, Wei Zhi-hui. Advances and perspectives on compressed sensing theory [J]. Journal of Image and Graphics, 2012, 17(1): 1-12(in Chinese)
邵文泽，韦志辉. 压缩感知基本理论：回顾与展望[J]. 中国图象图形学报，2012，17(1)：1-12

[3] Jiao Li-cheng, Yang Shu-yuan, Liu Fang, et al. Development and Prospect of Compressive Sensing [J]. Acta Electronica Sinica, 2011, 39(7): 1651-1662(in Chinese)
焦李成，杨淑媛，刘芳，等. 压缩感知回顾与展望[J]. 电子学报，2011，39(7)：1651-1662

[4] Dai Qiong-hai, Fu Chang-jun, Ji Xiang-yang. Research on Compressed Sensing [J]. Chinese Journal of Computers, 2011, 34(3): 425-434(in Chinese)
戴琼海，付长军，季向阳. 压缩感知研究[J]. 计算机学报，2011，34(3)：425-434

[5] Mueller K, Yagel R. On the Use of Graphics Hardware to Accelerate Algebraic Reconstruction Methods [C] // Proceedings of SPIE Medical Imaging Conference 1999. San Diego, America, 1999

[6] Xiong Z, Chi W Q, Lu K, et al. GPU Acceleration of Saliency Detection Algorithm [C] // Proceedings of the 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science. IEEE, Guangxi, 2012: 48-51

[7] Guo Rui-ran, Song Jian-xin. Research and Implementation of Parallel Signal Reconstruction about Image Compressed Sensing Based on GPU [J]. Video Engineering, 2014, 38(11): 15-19(in Chinese)
郭睿冉，宋建新. 图象压缩感知的基于GPU的并行重构算法研究[J]. 电视技术，2014，38(11)：15-19

[8] Li Guo-yan, Hou Xiang-dan, Gu Jun-hua, et al. A GPU-Based Parallel Design and Implementation of Sparse MRI Reconstruction Algorithm [J]. Computer Applications and Software, 2013, 30(9): 163-166(in Chinese)
李国燕，侯向丹，顾军华，等. 稀疏磁共振图像重建算法的GPU并行设计与实现[J]. 计算机应用与软件，2013，30(9)：163-166

[9] Kulkarni A, Mohsenin T. Accelerating Compressive Sensing Reconstruction OMP Algorithm with CPU, GPU, FPGA and Domain Specific Many-Core [C] // IEEE International Symposium on Circuits and Systems (ISCAS 2015). Lisbon, Portugal, 2015: 970-973

[10] Fowler J E, Mun S, Tramel E W. Block-Based Compressed Sensing of Images and Video [J]. Foundations Trends in Signal Processing, 2012, 4: 297-416

[11] Lee H, Oh H, Lee S, et al. Visually Weighted Compressive Sensing: Measurement and Reconstruction [J]. IEEE Trans. on Image Processing, 2013, 22(4): 1444-1455

[12] Farber R. CUDA Application Design and Development [M]. Burlington: Morgan Kaufmann, 2011

[13] Wang Ze-huan, Wang Peng. Introduction to CPU Parallel Programming Technology [J]. E-science Technology & Application, 2013, 4(1): 81-87(in Chinese)
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

[14] 张舒，褚艳利. GPU高性能运算之CUDA[M]. 北京：中国水利水电出版社，2009

[15] Deng Chun-yuan, Wei Yi-min. Some New Results of the Sherman-Morrison-Woodbury Formula [C] // Proceeding of The Sixth International Conference of Matrices and Operators. Chengdu, China, 2011, 2: 220-223

[16] Hager W W. Updating the Inverse of a Matrix [J]. SIAM REVIEW, 1989, 31(2): 221-239