

采用 GPU 加速的压缩感知图像恢复算法

苗 壮¹, 隋 冬², 李棚艳², 柯 钧²

(1 昆明物理研究所 微光夜视技术重点实验室, 陕西 西安 710000; 2 北京理工大学 光电学院, 北京 100081)

摘 要: 压缩感知(Compressed Sensing, CS)的信号重构部分需要进行大数据量的计算, 然而传统的 CPU 对大量的矢量计算并没有优势. 为了解决这一问题, 我们以 CUDA 作为并行计算架构, 通过 GPU-CPU 并行编程技术, 实现了三种快速高效的压缩感知图像恢复算法, 包括正交匹配追踪 OMP 算法、两步阈值迭代 TwIST 算法和线性 Bregman 算法.

关键词: 压缩感知; CUDA; GPU; 正交匹配追踪 OMP 算法; 两步阈值迭代 TwIST 算法; 线性 Bregman 算法

中图分类号: TP41

文献标识码: A

文章编号: 1000-7180(2016)12-0125-05

Accelerate Compression Sensing Reconstruction Algorithms Using GPU

MIAO Zhuang¹, SUI Dong², LI Peng-yan², KE Jun²

(1 Low-Light-Level Night Vision Technology Laboratory, Kunming Institute of Physics, Xi'an 710000, China;

2 School of Optoelectronics, Beijing Institute of Technology, Beijing 100081, China)

Abstract: The signal reconstruction in compressive sensing (CS) requires a large amount of data processing. However the traditional CPU has no advantage on vector calculation. To deal with this issue, we design a parallel computing architecture using CUDA. Based on GPU-CPU parallel programming technology, and we realize three fast and efficient CS image reconstruction algorithms, including OMP, TwIST, and linear Bregman algorithm.

Key words: compressed sensing; CUDA; GPU; OMP; TwIST; linear Bregman algorithm

1 引言

随着 CPU 主频增长接近极限, 人们的注意力从提高单个 CPU 的处理速度转移到多个处理芯片的并行处理. 如何充分利用并行系统, 在并行系统上进行编程成为学术界的一个研究重点. 并行系统可以分为两类: 一类是同构并行; 另一种是异构并行. 前者主要解决并发处理, 如网站对网页的并发请求处理. 后者更多是用于设计数值运算. CPU-GPU 的组合是目前异构并行系统中发展最快的领域.

CUDA^[1] (Compute Unified Device Architecture, 统一计算设备架构) 是 NVIDIA 公司为其制造的显卡提供的用于并行计算的统一架构和编程接口. 利用 CUDA, 程序员可以编写出 CPU-GPU 上

的混合程序, 使 CPU 与 GPU^[2] 协同工作, 可以让应用在效率上获得极大的提升. CUDA 还具有 C 语言的编程接口和关键字, 可以方便地进行编写.

在 CUDA 架构下, 一个应用程序被分为两个部分分别执行, 即设备端程序和主机端程序. 设备端程序(也称为 Device 端), 指专门在 GPU 中运行的程序模块, 又称为核函数(Kernel). 主机端程序(也称为 Host 端), 是指运行在 CPU 中的程序模块. CUDA 源程序经过 NVIDIA 公司开发的 C 编译器 NVCC 编译, 一部分代码会被编译成标准的 C/C++ 程序, 在 CPU 上执行, 即成为主机端程序; 另一部分代码首先转换成 PTX 文件, 然后再转换成可由 GPU 执行的指令, 即成为设备端程序. GPU 执行端程序, 进行具体的数学运算. 当计算任务结束后,

主机端程序再将设备端的结果从 GPU 的显存中取回,返回给调用程序使用。

2 压缩感知技术

压缩感知(CS)^[3-6]是近十年发展极为迅速的一个研究领域.它突破传统尼奎斯特定理的对采样频率的制约,通过少量测量值就可以以高概率重构原始信号。

压缩感知可以由如下的数学形式表述:

$$\mathbf{y} = \boldsymbol{\varphi} \mathbf{x} \quad (1)$$

式中, $\boldsymbol{\varphi}$ 是一个 $m \times n$ 维的矩阵, $m \ll n$; $\mathbf{x}$ 是一个 $n \times 1$ 的向量,表示输入信号; $\mathbf{y}$ 是一个 $m \times 1$ 的向量,表示观测值. 因为 $\boldsymbol{\varphi}$ 的大小 $m \times n$, 方程组的个数远小于未知数,所以在数学上,这一方程组是欠定的并且存在无穷多个解. 要从测量值 $\mathbf{y}$ 中直接反解 $\mathbf{x}$ 十分困难. 但如果 $\mathbf{x}$ 是一个稀疏信号,通过构造观测矩阵,利用各类重建算法,从 $\mathbf{y}$ 中恢复 $\mathbf{x}$ 是可以实现的. 总的来说,CS 的关键技术主要分为以下三个部分: (1) 信号的稀疏表示; (2) 测量矩阵的选取; (3) 信号的精确重构. 其中重构算法是 CS 的一个重要部分。

3 压缩感知恢复算法

压缩感知恢复算法是在已知 $\mathbf{y}$ 和 $\boldsymbol{\varphi}$ 的情况下要从 $\mathbf{y} = \boldsymbol{\varphi} \mathbf{x}$ 中求出原始信号. 如果 $\mathbf{x}$ 可以在某正交基上稀疏表示,即 $\mathbf{x} = \boldsymbol{\psi} \boldsymbol{\theta}$ 则有 $\mathbf{y} = \mathbf{A} \boldsymbol{\theta}$, 其中 $\mathbf{A} = \boldsymbol{\varphi} \boldsymbol{\psi}$, $\boldsymbol{\psi}$ 为稀疏矩阵. 由于系统是欠定的,无法直接求解方程. 在压缩感知技术中,原始信号具有稀疏性,所以我们将信号恢复问题转化为如下的最优解问题:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{x}\|_0 \\ \text{subject to: } & \mathbf{y} = \mathbf{A} \mathbf{x} \end{aligned} \quad (2)$$

直接求解式(2)是一个 NP-hard 问题,是不现实的. 所以往往放低对 L_0 -norm 的严格要求或者寻找局部的最优解来实现快速迭代. 学者们提出了一系列寻找次优解的优化方法,本文介绍其中典型的三种恢复算法。

3.1 OMP 算法

正交匹配算法^[7](OMP)是贪婪迭代算法中的典型算法,基本思想是在观测矩阵 $\boldsymbol{\varphi}$ 中选择与信号最匹配的列向量来构建稀疏逼近,然后对选取的支撑集进行 Gram-Schmidt 正交化运算,最后用最小二乘法求原始值. 其具体算法流程可参考文献[7].

3.2 两步阈值迭代(TwIST)算法

两步迭代收缩^[8](two-step iterative shrinkage,

TwIST)是 IST(iterative shrinkage)和 IRS(iterative reweighted shrinkage)基础之上提出的一种新的迭代算法. 较普通的阈值迭代,它具有收敛速度快的特点. 该算法的核心是利用前两个估计值更新当前值,这也就是所谓的“两步”迭代收缩。

算法的更新过程为

$$x_1 = S_\lambda(x_0) \quad (3)$$

$$\begin{aligned} x_t &= (1 - \alpha) * x_{t-2} + (\alpha - \beta) * \\ & x_{t-1} + \beta * S_\lambda(x_{t-1}) \end{aligned} \quad (4)$$

式中, c 为线性搜索参数, S_λ 是阈值(Thresholding)函数, β 和 α 是参数,

$$\alpha = \frac{2}{1 + \sqrt{1 - p^2}}, \beta = \frac{2\alpha}{\lambda_1 + \lambda_N},$$

其中 $\lambda_N = 1$,

$$\lambda_1 = 1e - 4, p = \frac{\lambda_N - \lambda_1}{\lambda_N + \lambda_1}.$$

采用传统的算法选取的 c 值没有自适应性,本文通过 Barzilai-Borwein(BB) 方法引入自适应搜索步长. 在压缩感知中, c 的求解为下式:

$$c_t = \frac{\|\mathbf{A}^T \mathbf{A} s_t\|_2}{\|\mathbf{A} s_t\|_2} \quad (5)$$

我们也采用自适应的方法确定阈值参 λ . 因为对于压缩感知问题来说,信号中的大部分值为 0,所以在迭代前预估出一个稀疏度 K ,在每次迭代中把信号中第 K 大的值作为 λ . 另外我们定义迭代的终止函数为

$$C(x_t, x_{t-1}) = \frac{|f(x_t) - f(x_{t-1})|}{f(x_{t-1})} \quad (6)$$

下面给出完整的 TwIST 算法:

步骤 1: 初始化 $x_0 = 0, t = 1$, 设定迭代终止值

ϵ , 惩罚项权重 $\lambda = 1, c = 1$, 参数 $\alpha = \frac{2}{1 + \sqrt{1 - p^2}}$,

$\beta = \frac{2\alpha}{\lambda_1 + \lambda_N}$, 其中 $\lambda_1 = 1e - 4, p = \frac{\lambda_N - \lambda_1}{\lambda_N + \lambda_1}$;

步骤 2: 计算

$$x_{t+1} = S_\lambda(x_t + \frac{1}{c} \mathbf{A}^T (\mathbf{A} x_t - \mathbf{y})), \text{ 其中 } \lambda \text{ 为自适应值};$$

步骤 3: 利用式(5)更新 c , 更新 $t = t + 1$;

步骤 4: 计算

$$\begin{aligned} x_t &= (1 - \alpha) * x_{t-2} + (\alpha - \beta) * \\ & x_{t-1} + \beta * S_\lambda(x_{t-1}) \end{aligned};$$

步骤 5: 计算 $C(x_t, x_{t-1})$, 若 $C(x_t, x_{t-1}) < \epsilon$, 算法终止, 返回 x_t , 否则返回步骤 2。

3.3 线性 Bregman 算法

如果将压缩感知中的 $\frac{1}{2} \| \mathbf{y} - \mathbf{A}\mathbf{x} \|_2^2$ 定义为 $H(\mathbf{x}, \mathbf{A})$, 而稀疏信号 L1-norm 的惩罚项定义为 $J(\mathbf{x})$, 根据文献[9]中的 Bregman 算法, 我们能够快速求出下面问题:

$$\min_u \{J(\mathbf{x}) + H(\mathbf{x}, \mathbf{A})\} \quad (7)$$

如果我们定义 $u, v \in R^M$ 两点间的 Bregman 距离为

$$D_f^p(u, v) = J(u) - J(v) - \langle p, u - v \rangle,$$

式中, p 为 $\partial J(v)$ 中的一个次梯度^[9], 那么 Bregman 迭代算法流程如下所示.

初始化: $k = 0, x^0 = 0, p^0 = 0$

步骤 1:

$$x^{k+1} = \arg \min_x D_f^p(x, x^k) + H(x);$$

步骤 2:

$$p^{k+1} = p^k - \nabla H(x^{k+1});$$

步骤 3: $k = k + 1$; 若 $\mathbf{x}$ 未收敛或者没有满足停止条件, 跳转步骤 1 (收敛条件可自定义), 否则停止, 返回 $\mathbf{x}$.

在上述算法流程中, 每一步需要计算 $D^p(\mathbf{x}, x^k) + H(\mathbf{x})$. 其中 $H(\mathbf{x})$ 项在压缩感知中涉及大量矩阵运算, 所以在线性 Bregman 算法^[10]中为了减少算法时间复杂度, 需要将 $H(\mathbf{x})$ 函数进行泰勒展开, 然后进行多次化简. 最终所得线性 Bregman 算法如下所示.

初始化:

$$x_0 = 0, v_0 = 0, k = 0, \lambda, \delta, \varepsilon;$$

$$\text{步骤 1: } v_{k+1} = v_k + \mathbf{A}^T(\mathbf{y} - \mathbf{A}x_k);$$

$$\text{步骤 2: } x_{k+1} = \delta T(v_{k+1}, \lambda);$$

步骤 3: $k = k + 1$, 如果 $C(x_k, x_{k-1}) < \varepsilon$ 则算法终止, 返回 x_k , 否则跳转步骤 1. 步骤 2 中函数 T 为

$$T(\alpha, \beta) = \begin{cases} \alpha - \beta, & \text{if } \alpha \in (\beta, +\infty) \\ 0, & \text{if } \alpha \in (-\beta, \beta) \\ \alpha + \beta, & \text{if } \alpha \in (-\infty, -\beta) \end{cases}$$

算法中的阈值参数 λ 与 TwIST 算法中的选择相似, 参数 δ 需要自行调节. 而判断算法终止的条件, 我们使用和 TwIST 算法一样的函数, 即

$$C(x_t, x_{t-1}) = \frac{|f(x_t) - f(x_{t-1})|}{f(x_{t-1})}.$$

4 恢复算法 GPU 加速

4.1 CPU-GPU 并行计算框架设计

为了快速实现算法和隐藏复杂的底层控制逻辑,

我们设计了一种智能的内存控制系统, 在此基础上构建了用于压缩感知重构的数据结构. 这包括内存控制系统、计算框架的数据容器设计、线程池-数据读写 IO 系统, 以及数据库设计.

传统的 GPU-CPU 并行计算中, 在运算之前, 需要用 malloc 函数和 cudamalloc 函数进行 Host 端和 Device 端的内存空间申请, 然后通过系统函数 cudaMemcpy 进行 Host 与 Device 的数据通信. 在 GPU 和 CPU 计算和传输数据的过程中需要反复调用底层函数, 这对于开发者来说十分不便, 而且很容易造成错误. 为了简便开发, 减少错误, 我们将 host 端的内存与 device 端的内存抽象出来, 统一定义名为 Cmemory 的类. 在 Cmemory 类中, 私有成员包括 cpu 与 gpu 数据指针, 以及数据的大小, 表示如下:

```
void * cpu_ptr;
void * gpu_ptr;
size_t size_;
```

为了隐藏底层复杂的数据传输过程, 我们用一个枚举变量来控制数据的转移. 通过枚举出数据的存在方式, 我们可以非常容易地监控当前数据的位置. 然后我们定义两个私有函数 to_cpu() 以及 to_gpu() 来完成数据的传输工作.

Cmemory 是内存的统一抽象, 但是我们最终还是需要得到具体的数据, 所以在 Cmemory 类中我们提供两个接口, 用来返回指向数据的指针.

完成以上定义后, 对于用户来说, 只需要指定一个 Cmemory 的类并提供数据大小, 然后通过函数接口就可以获得数据.

在统一了内存实现 Cmemory 类之后, 我们需要计算的容器. 对于压缩感知恢复算法来说有两类数据结构, 第一类为向量, 比如我们需要恢复的信号以及测量值, 第二类为矩阵. 我们定义了 Container 类来抽象容器的概念, 一个 Container 本质上是一个三维数组, 对于 n 维向量来说 Container 的属性为个数 1, 长度 1, 宽度 n . 长度和宽度可以分别表示矩阵或向量的维度, 另外增加的个数属性主要是为了兼容批量处理程序. 下面为 Container 类的私有成员变量:

```
size_t num_;//个数
size_t h;//长度
size_t w;//宽度
size_t size_;//大小 = 个数×长度×宽度
shared_ptr<Cmemory> memory_ptr;//指向
Cmemory 的智能指针
```

memory_ptr 是指向实际数据所在内存的指针, 为了防止内存泄露我们使用了 Boost 库提供的智能共享指针(shared_ptr). Container 内的数据其实是 Cmemory 来维护的, 所以减少了很多复杂的底层操作. 在 Container 的成员函数里, 有两个成员函数用来获取数据:

```
T * mutable_cpu_data(); //获取 cpu 数据
T * mutable_gpu_data(); //获取 gpu 数据
```

之前计算容器定义中的个数属性, 是为了能够提供批量服务. 我们可以把每一次压缩感知恢复看作一次服务, 每当一个请求到达就创建一个新的服务对象, 然后在新的服务对象中为请求服务. 但当有大量请求并发访问时, 服务器不断创建和销毁对象的开销很大. 所以提高服务器效率的一个手段就是尽可能减少创建和销毁对象的次数, 特别是一些很耗资源的对象的创建和销毁, 这样就引入了“池”的概念. “池”的概念使得人们可以定制一定量的资源, 然后对这些资源进行复用, 而不是频繁创建和销毁.

线程池是预先创建线程的技术. 线程池在还没有任务到来之前, 创建一定数量的线程, 放入空闲队列中. 这些线程都是处于睡眠状态, 即均为启动, 不消耗 CPU, 只占用较小的内存空间. 当请求到来之后, 缓冲池给这次请求分配一个空闲线程, 把请求传入此线程中运行, 进行处理. 当预先创建的线程都处于运行状态, 即一定数量的新线程, 用于处理更多的请求. 当系统比较闲的时候, 也可以通过移除一部分一直处于停用状态的线程. 我们用 python 实现了线程池, 其原理如图 1 所示.

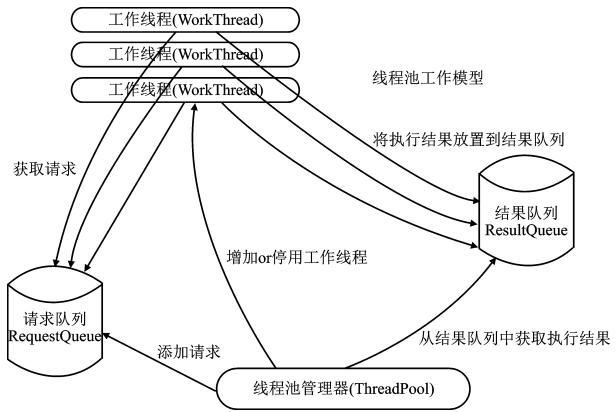


图 1 线程池原理

除了以上三方面的设计外, 感知信号重构中存在大量的矩阵-向量、向量-向量运算, 这些运算可以由 Blas(Basic Linear Algebra Subprograms, 基础线性代数程序集)来完成. 我们使用 NVIDIA 公司的

并行计算数学库 Cublas^[11]. 在 NVIDIA 公司的网站上性能测试显示其与最新的 MKL BLAS(业内领先的收费 BLAS 库) 相比, 可实现 6-17 倍性能提升.

4.2 并行计算优化技巧

为了进一步提高性能, 我们在上述 GPU 并行计算框架的基础上, 使用一系列针对压缩感知并行算法的优化技巧. 包括: (1) 求数组中的最大最小值, 或者求数组的 L1\|L0-norm, 在常规的串行算法中我们只需要简单的遍历一遍数组就可以得到结果, 在并行的实现中使用归约算法; (2) 在矩阵运算中, 为了减少从内存中读取数据的时间, 优化程序, 使用循环展开的技巧.

4.3 实验结果

我们将三种算法在上述 GPU 并行计算框架和 CPU 上分别运行. 测试数据如表 1、2 所示, 每个算法的运行时间是取十次运行时间的均值.

表 1 运行时间(CPU)

矩阵 A 大小	Linear Bregman/ms	TwIST/ms	OMP/ms
A(512,1 024)	60	35	212
A(1 024,4 096)	150	274	1 343
A(2 048,1 000)	296	370	1 665

表 2 运行时间(GPU)

矩阵 A 大小	Linear Bregman/ms	TwIST/ms	OMP/ms
A(512,1 024)	101	64	583
A(1 024,4 096)	783	904	13 151
A(2 048,1 000)	1 891	2 527	24 461

5 结束语

本文在介绍压缩感知理论的基础上, 重点研究了压缩感知的三种典型重构算法, 包括正交匹配追踪 OMP 算法、两步阈值迭代 TwIST 算法和线性 Bregman 算法. 为了提高其运行速度, 设计了基于 CUDA 并行计算架构的 GPU 并行计算框架. 将三种算法分别在 GPU 和 CPU 上运行, 由实验结果可看出, 在 GPU 并行计算框架下的压缩感知重构算法相比未经优化的串行版本算法在速度上有明显地提升, 其中 OMP 所需时间最长, 当观测矩阵 A 越大时, Linear Bregman 相对于其他两种恢复算法, 速度更快.

参考文献:

[1] Nvidia Corporation. Nvidia cuda c best practice guide [R]. 2014.

[2] Owens J D, Houston M, Luebke D. GPU computing [J]. proceedings of the IEEE, 2008, 96(5): 879-899.

[3] Donoho D. Compressed sensing[J]. IEEE Trans Inf Theory, 2006,52(4):1289-1306.

[4] Richard G, Baraniuk. Compressive sensing[J]. IEEE Signal Processing Magazine, 2007,24(4):118-224.

[5] 石光明,刘丹华,高大化,等. 王良君压缩感知理论及其研究进展[J]. ACTA Electronica Sinica, 2009,37(5): 1070-1081.

[6] 张锐. 基于压缩感知理论的图像压缩初步研究[J]. 电脑知识与技术,2010,6(4):958-959.

[7] Tropp J A, Gilbert A C. Signal recovery from random measurements via orthogonal matching pursuit [J]. IEEE Transactions on Information Theory 2007, 53 (12):4655-4666.

[8] Bioucas-Dias J, Figueiredo M. A new TwIST: Two-step iterative shrinkage/thresholding algorithms for image restoration [J]. IEEE Trans Image Process, 2007,16(12): 2992-3004.

[9] Wotao Yin, Stanley Osher, Donald Goldfarb, aet al. Bregman iterative algorithms for l1-minimization with applications to compressed sensing[J]. SIAM J Imaging Sci, 2008,1(1):143-168.

[10] StanleyOsher, Yu Mao, Bin Dong, et al. Fast linearized Bregman iteration for compressive sensing and sparse denoising[J]. Commun Math Sci, 2010,8(1): 93-111.

[11] Nvidia C. Cublas library. NVIDIA Corporation [R]. Santa Clara, California, 2008.

作者简介:

苗 壮 女,(1981-),硕士,高级工程师. 研究方向为微光夜视技术.

隋 冬 男,(1988-),硕士研究生. 研究方向为压缩感知、并行加速.

李棚艳 女,(1991-),硕士研究生. 研究方向为压缩成像、并行计算.

柯 钧(通讯作者) 女,(1974-),博士,副教授. 研究方向为压缩成像、计算成像、压缩感知. E-mail:jke@bit. edu. cn.

(上接第 124 页)

[7] 郝金光,冯宇,邱相艳. Hogenauer CIC 滤波器算法研究及 FPGA 设计实现[J]. 微型电脑应用, 2006, 22 (9): 15-17.

[8] Donadio M P. Cic filter introduction[EB/OL]. [2016-01-22]. www. read. pudn. com.

[9] Steven W Smith. 实用数字信号处理-从原理到应用 [M]. 北京. 人民邮电出版社. 2010:220-222.

[10] Baker R J. CMOScircuit design, layout and simulation

[M]. New York:John Wiley& Sons, 2010.

作者简介:

柳林刚 男,(1990-),硕士研究生. 研究方向为数模混合集成电路设计. E-mail:liulingang1025@163. com.

李荣宽 男,(1968-),博士,副教授. 研究方向为混合信号集成电路设计.