GPU 加速的自适应仿射传播聚类方法①

陈艳阳, 曾卫明

(上海海事大学 信息工程学院, 上海 201306)

摘 要: 自适应仿射传播聚类作为一种新兴的聚类算法, 不需要指定初始类心以及类数, 对解决聚类中类数不确定性问题非常有效. 然而, 自适应仿射传播聚类存在时间消耗过大的问题, 当样本数量较大时运行速度缓慢. 为了提高自适应仿射传播聚类的运行速度, 基于 NVIDIA 公司的统一计算设备架构(Compute Unified Device Architecture, CUDA)和 Matlab 并行工具箱, 提出了一种自适应仿射传播聚类的并行化方法. 实验结果表明, 基于 GPU 并行化的自适应仿射传播聚类在运行速度上有了明显提高, 与该算法的串行执行方式相比, 运行速度提升 2 倍以上, 并且随着样本数量的增长, 加速性能越来越好.

关键词: 自适应仿射传播聚类; 并行化; 统一计算设备架构; 并行工具箱; GPU 加速

GPU-Accelerated Adaptive Affinity Propagation Clustering Method

CHEN Yan-Yang, ZENG Wei-Ming

(Lab of Digital Image and Intelligent Computation, College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China)

Abstract: Adaptive affinity propagation clustering(adaptive Affinity propagation clustering, adAP), as a new clustering algorithm, does not need to specify the initial “exemplars” and the class number, which is effective to solve the problem of class number uncertainty in clustering. Then, as a result of the adAP is extremely time consuming, the larger the number of samples is, the slower the speed is. In order to improve the speed of the adAP, this paper realizes a parallel method, which is based on NVIDIA’s Compute Unified Device Architecture (CUDA) and Matlab parallel computing toolbox. The experiment results show that the GPU-based parallel adAP method has a certain speedup effect, and it is more than 2 times faster than the serial execution. With the increase of the number of samples, the acceleration performance is getting better and better.

Key words: adaptive affinity propagation clustering; parallelization; compute unified device architecture; parallel computing toolbox; GPU acceleration

仿射传播聚类(Affinity Propagation clustering, AP)是 Frey 等人于 2007 年在《Science》上提出的一种新型聚类算法, 相比于其他聚类方法, 在样本类数较多时分类能力更好[1]. 然而, AP 算法仍然需要手动设置聚类个数, 且容易陷入局部最优. 针对 AP 算法存在的问题, 王开军等人在文献[2]中提出了一种自适应仿射传播聚类(adaptive Affinity Propagation clustering, adAP)方法, 该方法通过自适应扫描偏向参数空间来搜索聚类个数来寻找最优聚类结果, 并能够自适应地调整阻尼因子以消除算法迭代过程中出现的震荡. 目前, adAP 算法已经在文本聚类、大规模数据聚类、手写字符识别和功能磁共振成像数据分析中得到了应用[3-6]. 尽管 adAP 算法克服了 AP 算法的缺点并获得了更好的聚类性能, 但需要消耗更多的计算时间, 当数据样本数量稍大时, 算法运行时间急剧增加, 甚至超过了可容忍的等待限度[2]. 因此, 提高该 adAP 算法的运行速度是当前需要解决的重要问题. 由于 adAP 算法的核心过程是 AP 算法, 因此, adAP 算法的加速方法可参考

① 基金项目:国家自然科学基金(31470954)

收稿时间:2016-02-24;收到修改稿时间:2016-03-31 [doi:10.15888/j.cnki.csa.005406]

AP 算法的加速方法. 2009 年, Hussein 等人于提出一种近似逼近方法表示 AP 迭代过程中的归属度和吸引度, 并采用 GPU 对算法进行了加速, 遗憾的是该方法缩小了 AP 的适用范围[7]. 随后, Kurdziel 等人提出在多个 GPU 上同步运行 AP 算法的方法, 该方法将原数据集划分为子数据集在多个 GPU 上分布式处理, 从而达到算法加速的目的[8]. 多 GPU 并行加速固然好, 但是对硬件要求较高, 一般情况下很难满足. 最近, 在 adAP 用于 fMRI 数据分析的应用中, 由于数据集过大、算法复杂度高, adAP 甚至无法运行, 任天龙等人采用分组二次聚类的方法, 解决了算法无法运行的问题, 取得了一定效果[6]. 二次聚类方法降低了单次执行的内存开销需求, 但没有从根本上解决算法运行速度缓慢的问题. 综上所述, 本人在借鉴前人研究经验的基础上, 分析了 adAP 算法的流程和复杂度, 基于 NVIDIA 公司的统一计算设备架构(Compute Unified Device Architecture, CUDA)[9]和 Matlab[10]平台, 提出了一种采用共享内存和多线程的 adAP 算法并行化方法, 以提高该算法的运行速度.

1 自适应仿射传播聚类

adAP 是一种改进的 AP 算法[2], adAP 将每个样本均视为潜在的聚类中心, 以样本点之间的“距离”作为样本间相似度的度量方式, 通过在样本的特征空间中迭代更新样本间的归属度和吸引度以最小化聚类能量函数, 从而达到对样本分类的目的, 其中, 样本间吸引度和归属度与“距离”有关, 以下介绍 adAP 算法的具体过程.

对于一个大小为 $m\times n$ 的数据集 X, m 为特征数量, n 为样本个数, X 中任意两个样本 x_i 和 x_j ($1\le i,j\le n$ 且 $\{x_i,x_j\}\in X$)之间的相似度定义为:

$$S(i,j)=-\left\|x_i-x_j\right\|^2 \tag{1}$$

$A(i,j)$ 为归属度, 表示 x_i 选择 x_j 作为其类代表的可信程度. $R(i,j)$ 为吸引度, 表示候选类心 x_j 适合作为 x_i 的类代表的可信程度. 若样本集被划分为 k 个类, 共有 k 个类心 $C_i,i=1,...,k$, adAP 的迭代目标是最小化能量函数 $J(C)=-\sum_{i=1}^{n}S(i,C_i)$. 在第 t 次迭代中, 吸引度和归属度的计算公式为:

$$R(i,j)^{(t)}=S(i,j)-\max_{j'\neq j}(A(i,j')^{(t-1)}+S(i,j')),j'\neq j \tag{2}$$

$$A(i,j)^{(t)}=\begin{cases}\min\{0,R(j,j)^{(t)}+\sum_{j'\neq i}\max\{0,R(j',j)^{(t)}\}\},i\neq j\\ \sum_{j'\neq i}\max\{0,R(j',i)^{(t)}\},i=j\end{cases} \tag{3}$$

相似度矩阵 S 对角线上的值 $S(i,i)$ 为偏向参数, 记为 p, 由公式(2)、(3)可得, 当 $p(i)$ 较大时, $R(i,i)$ 较大, $A(i,j)$ 也较大, 从而 x_i 成为聚类中心的可能性越大. 在第 t 次迭代中, $R^{(t)}$ 和 $A^{(t)}$ 还需要与上一步的 $R^{(t-1)}$ 与 $A^{(t-1)}$ 进行加权更新, 吸引度和归属度最终更新为:

$$R^{(t)}=(1-\lambda)\times R^{(t)}+\lambda\times(R^{(t-1)}) \tag{4}$$

$$A^{(t)}=(1-\lambda)\times A^{(t)}+\lambda\times(A^{(t-1)}) \tag{5}$$

其中, λ 为阻尼因子, 与 adAP 算法收敛速度有关. 迭代过程中会出现震荡, 即类数发生摆动, 由于描述算法震荡特征很困难, 非震荡特征的描述相对容易, 因此在迭代过程中设置移动监视窗记录非震荡特征的出现次数, 当非震荡特征出现次数低于一定比例, 则认定算法产生了震荡, 从而通过调节 λ 消除震荡[2]. 聚类个数的确定与 p 有关, 在算法的迭代过程中, 另外设置聚类个数的移动监视窗, 当监视窗中的聚类个数满足连续不变次数阈值时, 通过延迟迭代确定收敛聚类个数和对应的类心, 然后调节 p 以逃离当前聚类个数. 整体上, 当聚类个数不再改变、迭代变化量低于预先设定的阈值, 或者迭代次数达到预先设定的最大迭代次数, 则停止更新和迭代.

adAP 迭代结束后将获得一系列不同聚类数目的聚类结果, 为找到最佳聚类个数, 采用 Sihouette 指标进行聚类有效性评测, 单个样本 x_i ($i=1,...,n$)的 sil 值计算方法为[2,11]:

$$Sil(i)=\frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \tag{6}$$

其中, $a(i)$ 为样本 x_i 与其所属的类内其他样本的平均不相似度, $b(i)$ 为样本 x_i 与其不同类样本之间平均不相似度的最小值. X 中所有样本的平均 sil 值反映了聚类结果的质量, sil 值越大, 表示聚类质量越好, 聚类最优个数选择的详细过程见参考文献[2].

2 基于GPU的adAP算法并行化方法

本文 1 部分描述的 adAP 算法过程按功能可划分为三部分: ①计算样本间相似度; ②迭代更新吸引度和归属度; ③聚类结果有效性评测, 三部分顺序执行, 且在逻辑和数据上无交叉影响. 当迭代次数为 t 时,

①、②、③部分的时间复杂度分别为 $O(mn^2)$、$O(t)$和$O(kn^2)$. 时间复杂度分析表明, ①部分和③部分时间复杂度较高, ②部分的时间复杂度与算法的迭代次数有关. GPU 是并行的、面向吞吐量的计算部件, 可能有些在 CPU 上能以串行方式高效执行的任务在 GPU 上效率很低[12]. adAP 算法的具体过程显示①、③部分计算较密集, 适合并行化; ②部分计算量小、访问存储频繁, 并且自适应逃离震荡和自适应寻找聚类个数过程中涉及较多分支指令, 影响 GPU 并行化效果. 为充分利用 CPU 和 GPU 的计算资源, 本文根据算法流程中的计算和数据特点, 基于 NVIDIA 公司的 CUDA 平台和 Matlab 并行工具箱, 提出了①、③部分的 GPU 并行化方法, ②部分仍然在 CPU 上以串行方式执行, 最终对各功能部分进行整合, 从而实现 adAP 算法的整体加速.

2.1 计算样本间相似度

样本之间的相似度是 adAP 迭代更新的重要数据依据, 也是众多聚类算法的基础. 样本集 $X_{m\times n}$ 中任意样本 x_i 和样本 x_i 之间的相似度计算方法如公式(1)所示, 全部样本之间相似度为 $S_{n\times n}$. 计算 $S_{n\times n}$ 的时间复杂度为 $O(mn^2)$、空间复杂度为 $O(n^2)$, 时间复杂度和空间复杂度均较高. 为提高 GPU 的计算和访存速度, 计算样本间相似度部分的并行化采用 CUDA 中的二维线程块和共享内存实现, 如图 1 所示, X 为样本矩阵, S 为需要计算的相似度矩阵. 首先, 将样本矩阵 X 从主机端拷贝至设备端的全局内存中; 然后, 对 S 进行分块处理, 即分成子矩阵(Sub Matrix), 每个大小为16×16的二维线程块负责计算一个子矩阵中的全部值; S 的全体元素共需要开启 $n\times n$ 个线程. Sub Matrix 中的任意元素值 $S(i,j)$ 的生成过程如下:

① 将 X 的第 i 列和 j 列元素分段, 16 个元素为一段, 总共 k 段, k 的值为 $m/16$ 向上取整;

② 二维线程块中的线程以协作方式访问全局内存中①结果的一段数据, 分别保存至共享内存的 d_M 和 d_N 对应列中;

③ 使用 16 个线程计算 d_M 和 d_N 对应位置上元素差值的平方;

④ 求③结果的累加和, 线程同步;

⑤ 重复步骤②~④共 k 次;

⑥ 将累加和的负数作为 $S(i,j)$ 的值保存至全局内存变量 S 中.

⑦ 全部线程运行结束后, 将结果矩阵 S 从设备端拷至主机端, 相似度矩阵计算完成.

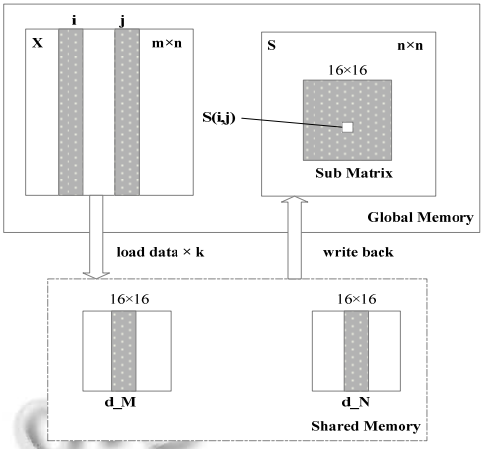

图 1　计算相似度矩阵

2.2 聚类有效性评测

adAP 采用 Silhouette 指标进行聚类有效性评测, 以选出最优聚类结果[2]. 聚类有效性测评的核心为 *Sil* 值, 其计算方法如公式(6)所示. 本文聚类有效性评测部分的并行化采用 Matlab mexFunction 与 CUDA 结合的方式, 如图 2 所示, 首先将样本标签 *Labels* 和相似度矩阵 S 从主机端以 gpuArray 的方式传送至 GPU 设备端; 然后, 在 mexFunction 中, 参考 GPU 并行化性能评价方法[13], 使用一维线程块, 每个线程块中包含 512 个线程, 依次调用计算类内平均不相似度(avgDistWithinClass) 和类间平均不相似度(avgDistBetweenClass)对应的 kernel 函数, 其中每个线程负责计算结果矩阵中单个值, 最后将全部样本的类内和类间平均不相似度结果返回至主机端, 以完成后续 sil 值的计算步骤.

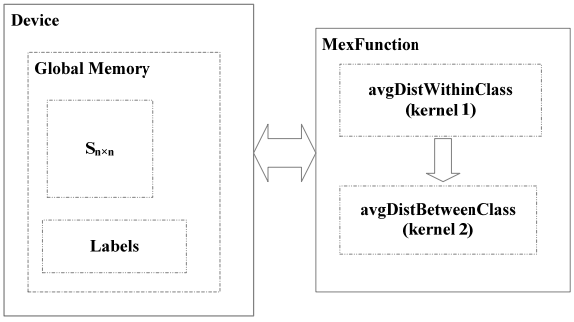

图 2　计算 Sil 值

3　实验结果与分析

本文使用的实验平台为: 操作系统: Ubuntu 12.04;

CPU: Intel Xeon E5-2650; GPU: NVIDIA Quadro 4000; CUDA: 5.5; Matlab: 2014a. 实验中用到的数据集来自文献[6]的单被试静息态 fMRI 数据, 大小为120×10000 , 数据特征为 120, 包含 10000 个样本.

按照本文第 2 部分所述的 adAP 并行化方法, 在算法的并行化效果验证实验中, 首先, 将数据集划分成大小不同的子数据集; 然后, 针对每个数据集, 分别执行算法的 GPU 和 CPU 版本(并行化前和并行化后)各 15 次并记录相应的运行时间; 最后, 在不考虑最大值和最小值的情况下, 统计平均运行时间.

在 GPU 和 CPU 上计算相似度矩阵的运行时间如表 1 所示, 随着样本数量的增多, GPU 加速效果越来越明显. 这是由于样本数量增多时共享内存方式的线程协作提高了访存效率, 缩短了线程等待时间, 体现了 GPU 的多线程并行执行优势.

表 1 GPU 和 CPU 上计算相似度矩阵的运行时间

样本数量	GPU 耗时(ms)	CPU 耗时(ms)	加速比
100	0.7	12.4	17.67
400	4.1	191.5	47.18
1600	55.4	3038.9	54.83
3200	216.7	12141	56.01

聚类结果有效性评测部分并行化前后的运行时间对比如表 2 所示. 该部分需要的运行时间较长, CPU 和 GPU 上的运行时间对比结果表明, 样本数量越多, GPU 加速效果越明显.

表 2 GPU 和 CPU 上聚类结果有效性评测耗时

样本数量	GPU 耗时(s)	CPU 耗时(s)	加速比
100	0.53	4.41	8.38
200	1.63	32.35	19.91
400	6.6	213.72	32.38
800	38.31	1987.9	51.89

以上各部分在并行化前和并行化后的运行时间对比表明, 当样本数量较大时, 计算样本间相似度和聚类有效性评测部分的 GPU 加速效果非常明显, 另外, 相对于其他部分, 聚类有效性评测需要消耗更多时间, 对该部分进行加速非常有必要. 根据以上各部分的 GPU 加速效果, 本文保留计算相似度矩阵和聚类有效性评测部分在 GPU 上执行, 其他部分仍然在 CPU 上执行, 从而对 adAP 算法整体进行加速.

adAP 聚类算法整体的 GPU 加速效果如图 3 所示, 其中 Tcpu/Tgpu 代表 GPU 加速比, 结果表明本文提出的方法有一定的加速效果, 随着样本数量的增多, GPU 加速倍数也在上升.

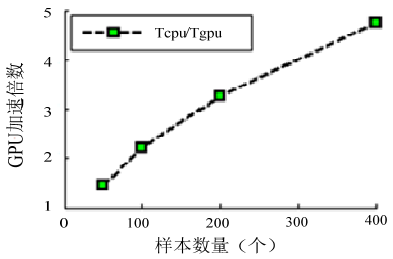

图 3 adAP 聚类的 GPU 加速比

4 结语

adAP 的聚类性能优于传统聚类算法, 具有广阔的应用前景, 提高该算法的运行速度非常有意义. 本文在分析 adAP 算法的过程后, 将该算法按功能分为三部分, 在分析各部分的时间复杂度后, 基于 GPU 的强大计算性能, 分别提出了并行化方法, 对于不利于并行化的部分仍然保留在 CPU 上以串行方式执行, 从而在整体上加速仿 adAP 算法. 实验结果表明, 随着样本数量的增加, GPU 并行化的 adAP 方法加速效果逐步升高. 本文提出的基于 GPU 的快速自适应仿射传播聚类方法有一定的加速效果, 对于 adAP 算法和相似算法的速度提升有借鉴价值.

参考文献

1 Frey BJ, Delbert D. Clustering by passing messages between data points. Science, 2007, 315(5814): 972–976.

2 王开军,张军英,李丹,张新娜,郭涛.自适应仿射传播聚类.自动化学报,2007,33(12):1242–1246.

3 He Y, Chen Q, Wang X, Xu R. An adaptive affinity propagation document clustering. 2010 The 7th International Conference on Informatics and Systems (INFOS). IEEE. 2010. 1–7.

4 Sun C, Wang C, Song S, Wang Y. A local approach of adaptive affinity propagation clustering for large scale data . International Joint Conference on Neural Networks, 2009. IJCNN 2009. IEEE. 2009. 2998–3002.

5 杨怡,王江晴,朱宗晓.基于仿射传播聚类的自适应手写字符识别.计算机应用,2015,35(3):807–810.

6 Ren T, Zeng W, Wang N, Chen L, Wang C. A novel approach for fMRI data analysis based on the combination of sparse approximation and affinity propagation clustering. Magnetic

Resonance Imaging, 2014, 32(6): 736–746.

7 Hussein M, Abd-Almageed W. Efficient band approximation of Gram matrices for large scale kernel methods on GPUs. Sc Conference. 2009. 1–10.

8 Kurdziel M, Boryczko K. Dense affinity propagation on clusters of GPUs. Lecture Notes in Computer Science, 2011, 7203: 599–608.

9 NVIDIA. CUDA programming manual. http://docs.nvidia.com/cuda/cuda-c-programming-guide/.[2015-11-25].

10 Matlab. Parallel Computing Toolbox. http://cn.mathworks.com/help/distcomp/index.html. [2015-11-10].

11 Dudoit S, Fridlyand J. A prediction-based resampling method for estimating the number of clusters in a dataset, Genome Biology. Statistica Sinica, 2002, 3.1(7): 129–137.

12 David B, Wen M. 赵开勇,汪朝辉,程亦超,译.大规模并行处理器编程实战.北京:清华大学出版社,2013.

13 王锋,杜云飞,陈娟.GPGPU 性能模型研究.计算机工程与科学,2013,35(12):1–7.