DOI:10.7544/issn1000-1239.2015.20148151
52(11):2555-2567,2015

CPU-GPU 异构计算环境下的并行 T 近邻谱聚类算法

张 帅 李 涛 焦晓帆 王艺峰 杨愚鲁
(南开大学计算机与控制工程学院计算机科学与信息安全系 天津 300071)
(zhangshuai@mail.nankai.edu.cn)

Parallel TNN Spectral Clustering Algorithm in CPU-GPU Heterogeneous Computing Environment

Zhang Shuai, Li Tao, Jiao Xiaofan, Wang Yifeng, and Yang Yulu
(*Department of Computer Science and Information Security, College of Computer and Control Engineering, Nankai University, Tianjin* 300071)

Abstract Spectral clustering is one of the most popular clustering algorithms in the data mining field. However, this algorithm suffers from the storage and computational bottlenecks heavily when dealing with large-scale datasets. Current work focuses on improving the spectral clustering on both algorithm and implementation levels. But how to design an efficient spectral clustering algorithm, which can handle million scale datasets on a single node with multicore CPU and manycore accelerators, is still an unsolved problem. A parallel spectral clustering using T-nearest-neighbors (TNN) and its implementation for CPU-GPU heterogeneous computing environment, named parallel spectral clustering for hybrids (PSCH), is proposed in this paper. It breaks the GPU device memory limitation by partitioning the TNN similarity matrix into blocks, so the dataset scale only subjects to the size of the host memory. In PSCH, the 4-stage pipeline mechanism with dual rotating buffers is designed to compute the TNN similarity matrix using CUDA, which keeps all the CPU, GPU, and PCIe bus busy to achieve high performance gains while breaking the device memory limitation. The implicitly restarted Lanczos method (IRIM) on GPU is employed for the eigen-decomposition of the sparse TNN similarity matrix, alleviating the computational bottleneck of the eigensolver. The results show that PSCH is highly-efficient at exploring the GPU memory bandwidth and hybrid CPU-GPU computation power. PSCH is able to cluster million scale datasets on a single node equipped with one GTX 480 GPU and achieve 2.0~4.5 times performance gains compared with the MPI parallel spectral clustering implementation PSC using 16 processes for 4 datasets.

Key words spectral clustering; T-nearest-neighbors (TNN); CPU-GPU heterogeneous computing; CUDA; OpenMP

摘 要 谱聚类是数据挖掘领域最常用的聚类算法之一,但对于如何利用多核 CPU 与资源有限的众核加速器设计并实现一个在异构单节点上能够处理大规模数据集的高效谱聚类算法,目前尚无理想的解决方案.PSCH(parallel spectral clustering for hybrids)算法是专为 CPU-GPU 异构计算环境设计的并行 T 近邻(T-nearest-neighbors, TNN)谱聚类算法,通过分块计算相似性矩阵打破了 GPU 设备内存的限制,所能处理的数据集规模仅受限于 CPU 主存的容量.PSCH 算法中使用 CUDA 设计实现双缓冲轮转 4 段流水机制,通过重叠计算与传输在打破存储瓶颈的同时保证了高计算性能.PSCH 算法采用隐式

收稿日期:2014-10-27;**修回日期**:2015-04-16
基金项目:国家自然科学青年基金项目(61212005,61201424);天津市科技特派员项目(14JCTPJC00501);高等学校博士学科点专项科研基金项目(20130031120029)
通信作者:李 涛(litao@nankai.edu.cn)

重启动 Lanczos 方法(implicitly restarted Lanczos method, IRIM)在异构硬件上计算稀疏特征矩阵的特征分解,减轻了特征分解步骤的计算瓶颈.PSCH 算法在配有一块 GTX 480 GPU 的单节点上能够对百万以上规模的数据集进行聚类,并对实验中的 4 个数据集取得了相对于使用 16 进程的 MPI 并行谱聚类 PSC 算法 2.0～4.5 倍的性能.

关键词 谱聚类;T 近邻;CPU-GPU 异构计算;计算统一设备架构;OpenMP

中图法分类号 TP311

谱聚类方法由 Hagen 和 Kahng[1] 于 1992 年提出,并由 Ng 等人[2] 完善了其理论基础,是目前数据挖掘领域最常用的聚类算法之一.谱聚类的基本思想是通过对数据样本的相似性矩阵进行特征分解,将高维数据映射到低维特征向量空间中进行聚类.与其他分类方法相比,谱聚类能够处理非凸样本空间,并且能够保证全局最优收敛,已经被成功应用于图像处理、计算机视觉、网络安全等领域[3-5].然而,实现一个能够高效处理大规模数据集的谱聚类算法是十分困难的.对规模较大的数据集,存储相似性矩阵需要远超过普通 PC 机内存容量的存储空间,且计算过程非常耗时.如何减少谱聚类的空间消耗、提高谱聚类的运行速度,在数据集规模日益增加的大数据时代是一个十分重要且迫切需要解决的问题.

现有工作从算法和实现 2 个层面对谱聚类进行了优化.1)算法层面优化工作的主要出发点是避免存储完整的相似性矩阵.例如,Fowlkes 等人[6] 提出了 Nyström 方法,用近似子矩阵代替完整的相似性矩阵;Luxburg[7] 提出使用 T 近邻(T-nearest-neighbors, TNN)建立稀疏矩阵代替稠密相似性矩阵的方法.采用这些方法,单节点已经能够满足谱聚类的存储需求,但其时间开销仍然是单个计算节点难以承受的.2)实现层面的工作集中于利用并行手段解决单点的性能瓶颈[8].Chen 等人[9] 提出基于集群的并行谱聚类算法 PSC(parallel spectral clustering),其 MPI 实现能够处理百万规模的数据集;但是该算法并未考虑利用节点内多核 CPU 及众核加速器的并行性,且当节点数目较多时存在耐故障性差、高通信开销导致计算效率降低等问题.另一方面,清华大学 Zheng 等人[10] 尝试在单计算节点上使用 OpenMP 或 CUDA 对谱聚类进行并行化,显著减少了计算相似性矩阵的时间;但该方法仅考虑了单独使用 OpenMP 或 CUDA,并未充分利用 CPU-GPU 异构系统的性能,而且所提出的 CUDA 实现能够处理的问题规模受限于 GPU 设备内存的大小,导致应用范围有限.对于如何利用多核 CPU 与资源有限的众核加速器设计并实现一个在异构单节点上能够处理大规模数据集的高效谱聚类算法,目前尚无理想的解决方案.

利用众核加速器的高性能,同时打破设备内存对计算规模的限制并非不可能.谱聚类中耗时最多的步骤是计算相似性矩阵[6,11],该步骤的一个有用性质是其计算密度正比于数据规模(类似 Level-3 BLAS 标准中的接口).对于有 n 个样本、平均特征数为 d 的数据集,计算相似性矩阵的时间复杂度为 $O(n^2d)$,空间复杂度为 $O(nd)$.实际计算中,每读取 1 个数据值平均需要进行 $n/2$ 次计算.这个性质使得对足够大的数据集,数据传输的时间会小于计算花费的时间,因此有可能设计出一种恰当的分块策略,在不损失性能的前提下,利用资源受限的众核加速器计算规模超过设备内存规模的相似性矩阵.本文提出为 CPU-GPU 异构计算环境设计的异构并行谱聚类(parallel spectral clustering for hybrids, PSCH)算法,通过分块计算相似性矩阵打破了 GPU 设备内存的限制,所能处理的数据集规模仅受限于主存的容量.PSCH 算法中采用了双缓冲轮转 4 段流水机制,能够减少 CPU,GPU 和 PCIe 总线 3 个部件的空闲时间,在打破存储瓶颈的同时保证了高计算性能.

尽管计算相似性矩阵的步骤具有很好的性质,设计并实现 PSCH 算法仍需克服下列难点:1)GPU 的设备内存有限且无法扩展,必须针对实际的设备内存容量和数据集特征设计合理的分块策略,分块过小或过大都会导致性能下降;2)必须仔细设计计算步骤并进行实现,才能使多个分块的 CPU 计算、GPU 计算和 CPU-GPU 数据传输步骤重叠进行;3)在解决计算相似性矩阵步骤的性能瓶颈之后,谱聚类中的其他步骤(如特征分解)所占时间比例增加,成为新的性能瓶颈,需要通过异构计算手段予以解决.

具体地,本文作了如下贡献:

1) 提出用于 CPU-GPU 异构系统的 PSCH 算

法,能够在单异构节点上对百万以上规模的数据集进行高效谱聚类.对实验中的 4 个数据集,在单机上的 PSCH 算法取得了相对于使用 4 节点共 16 进程的 MPI 并行谱聚类 PSC 算法 2.0～4.5 倍的加速,性能优势明显.

2) PSCH 算法中采用双缓冲轮转 4 段流水机制处理耗时最多的计算稀疏相似性矩阵步骤,能够通过重叠计算与通信传输充分利用异构节点中多核处理器和众核加速器,且使所处理的数据集规模不受限于 GPU 设备内存的大小.

3) PSCH 算法采用 HLanc[12] 算法库在异构硬件上计算稀疏特征矩阵的特征分解.HLanc 算法库能够达到仅使用 CPU 的 PARPACK 8.8 倍的性能,解决了特征分解步骤的计算瓶颈.

1　谱聚类

1.1　谱聚类概述

谱聚类作为极具竞争力的聚类算法,目前已经取得了较广泛的应用.谱聚类的思想来源于谱图划分理论[13-14],其本质是通过特征分解将高维数据空间映射到特征向量空间,即低维的线性测度空间,然后在特征向量空间中对数据点进行聚类.通过高维数据空间到低维特征向量空间的映射,谱聚类能够处理非凸样本空间,同时避免"维度灾难".对于规模为 n 的给定数据样本集 $X=\{\boldsymbol{x}_i\in\mathbb{R}^d\}$,谱聚类首先根据 $\boldsymbol{x}_i$ 之间的相似关系建立相似性矩阵;然后通过计算相似性矩阵的特征向量构造特征向量空间,并将全部样本点映射到特征向量空间中;最后采用其他聚类方法将全部样本点聚类为若干簇.

由于在建立相似度矩阵、选取聚类参数及样本点聚类等步骤中采用的具体方法不同,谱聚类实际上包含一系列具体算法.其中,Ng 等人[2] 的工作是目前使用最为广泛的谱聚类算法之一.在此基础上产生了很多对谱聚类算法的理论研究:Zelnik-Manor 等人[15] 提出了一种自动选择聚类数目的方法;Dhillon 等人[16] 建立了带权核 K-means 方法与谱聚类之间的联系.Verma 等人[17] 比较了包括文献[2]在内的 4 种谱聚类算法的聚类质量.国内的肖宇等人[18] 等提出了一种加权的自适应相似度度量,能够描述多密度聚类簇中数据点间的相似性,同时降低离群点的影响.

谱聚类算法在实际应用中的一个问题在于,相似性矩阵的规模正比于数据集规模 n 的平方.因此当处理海量数据时,相似性矩阵需要的存储空间可以轻易超过普通机器的内存容量.近年来若干改进谱聚类算法被提出,以避免存储完整的稠密相似性矩阵.这些近似性技术主要分为 2 类[9]:1)稀疏化方法,即仅存储相似性矩阵中的非零元素;2)子矩阵方法,即对相似性矩阵的行/列进行采样,仅保留原稠密矩阵的一个矩形部分.

1.2　基于 TNN 的谱聚类

基于 TNN 的谱聚类是采用稀疏化方法的谱聚类改进算法之一[7,9].该方法仅使用与每个样本点最相关的 t 个样本点(即 t 个邻居)构造一个稀疏的近似相似性矩阵(通常 $t\ll n$),因此能够避免传统谱聚类算法的存储瓶颈.

基于 TNN 的谱聚类算法的第 1 步是构造稀疏相似性矩阵 $\boldsymbol{S}\in\mathbb{R}^{n\times n}$.对每个数据样本点 $\boldsymbol{x}_i$,算法维护 1 个大小为 t 的最大堆 H_i,计算 $\boldsymbol{x}_i$ 与其他样本点的相似性距离并插入 H_i 中,最后得到与 $\boldsymbol{x}_i$ 最为相似的 t 个样本点和对应的相似度,作为稀疏相似矩阵 $\boldsymbol{S}$ 的第 i 行.一个常用的相似性距离计算方法是高斯函数:

$$S_{ij}=\exp\left(-\frac{\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}{2\sigma^2}\right),\tag{1}$$

其中,σ 是一个控制 S_{ij} 随 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 之间距离变化速度的缩放因子.然后根据相似矩阵 $\boldsymbol{S}$ 计算规范化的 Laplacian 矩阵[19]:

$$\boldsymbol{L}=\boldsymbol{I}-\boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2},\tag{2}$$

其中,$\boldsymbol{D}$ 是一个对角矩阵:

$$D_{ii}=\sum_{j=1}^{n}S_{ij}.\tag{3}$$

易证对任何 $S_{ij}\geqslant 0$ 的相似矩阵 $\boldsymbol{S}$,其 Laplacian 矩阵是对称且半正定的.在理想情况下,即若一个聚簇中的样本与其他簇中的样本互不相关,Laplacian 矩阵中的非零元素只会以块对角矩阵的形式出现.这样的 $\boldsymbol{L}$ 有 k 个零特征值,即 k 个最小特征值[6].应的特征向量组成了一个 $\mathbb{R}^{n\times k}$ 上的矩阵 $\boldsymbol{V}$,其中的元素具有明确的非零元分布,因而采用 K-means 等简单的聚类算法能够轻易地将矩阵 $\boldsymbol{V}$ 的 n 行聚为 k 类.

然而在实际中,所得到的特征向量的形式为

$$\boldsymbol{V}'=\boldsymbol{V}\boldsymbol{Q},\tag{4}$$

其中,$\boldsymbol{Q}$ 是一个正交矩阵,此时不能够直接得到矩阵 $\boldsymbol{V}$ 并对其 n 行进行聚类.通过将 $\boldsymbol{V}'$ 标准化[9] 得到矩阵 $\boldsymbol{U}$,其元素为

$$U_{ij}=V'_{ij}\Big/\sqrt{\sum_{r=1}^{k}V'^2_{ir}},\tag{5}$$

其中, $i=1,2,\cdots,n; j=1,2,\cdots,n$. $\boldsymbol{V}'$的每一行都被标准化为单位长度. 由于 $\boldsymbol{Q}$ 的正交性,矩阵 $\boldsymbol{U}$ 的 n 行在 k 维空间中的单位球上形成 k 个彼此正交的聚簇,因此矩阵 $\boldsymbol{U}$ 的 n 行能够被 K-means 或其他简单的聚类算法聚为 k 类[20-21]. 原空间中的 n 个点(对应于矩阵 $\boldsymbol{U}$ 的 n 行)对应地被聚为 k 类.

实际计算过程中,为保证数值稳定性及加速特征向量的求解过程,可以求矩阵 $\boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}$ 的前 k 个最大特征值(理想情况下全为 1)对应的特征向量,然后对这 k 个特征向量组成的矩阵进行标准化和聚类. 易证这 k 个特征向量等价于 $\boldsymbol{L}$ 的 k 个最小特征值对应的 k 个特征向量.

2 PSCH 算法

2.1 概 述

PSCH 算法是专为 CPU-GPU 异构系统设计的一种具体的基于 TNN 的并行谱聚类算法,具体过程见算法 1. PSCH 算法接受一个 d 维的数据集 X、邻居数 t 和聚类数 k 作为输入,首先通过分块的方式计算出稀疏相似性矩阵 $\boldsymbol{S}$;然后使用隐式重启动 Lanczos 方法(implicitly restarted Lanczos method, IRLM)[22]求得 $\boldsymbol{S}$ 的前 k 个最大特征值对应的特征向量;对 k 个特征向量组成的矩阵 $\boldsymbol{V}$ 进行行标准化后得到矩阵 $\boldsymbol{U}$;最后使用 K-means 方法将 $\boldsymbol{U}$ 的 n 行(对应 X 中的 n 个样本)聚为 k 类. 指定较大的邻居数 t 会提高聚类精度,但会增加计算量.

算法 1. PSCH 算法.

输入: $X=\{\boldsymbol{x}_i\in\mathbb{R}^d\}_{i=1}^n, t, k$;

输出: X 的 k 个聚类.

函数: $eigenFactor(\boldsymbol{S}, k)$—计算对称矩阵 $\boldsymbol{S}$ 的前 k 个特征向量.

① 构造 n 个容量为 t 的最大堆 $\boldsymbol{H}[n]$;

② for ($i=1$; $i\leqslant n$; $i+=batch$) do

③ for each $\{\boldsymbol{x}_{i\leqslant j\leqslant \min(i+batch-1,n)}, \boldsymbol{x}_{(i+batch)\leqslant k\leqslant n}\}$ do

④ $dis_{jk}=\|\boldsymbol{x}_j-\boldsymbol{x}_k\|^2$ on GPU;

⑤ end for

⑥ for each $\{\boldsymbol{x}_{i\leqslant j\leqslant \min(i+batch-1,n)}, \boldsymbol{x}_{j\leqslant k\leqslant \min(i+batch-1,n)}\}$ do

⑦ $dis_{jk}=\|\boldsymbol{x}_j-\boldsymbol{x}_k\|^2$ on CPU;

⑧ end for

⑨ for each $\{\boldsymbol{x}_{i\leqslant j\leqslant \min(i+batch-1,n)}, \boldsymbol{x}_{j\leqslant k\leqslant n}\}$ do

⑩ $H[j].insert(\{dis_{jk}, k\})$;

⑪ $H[k].insert(\{dis_{jk}, j\})$;

⑫ end for

⑬ end for

⑭ for $i=1:n$ do

⑮ for each $\{dis, j\}$ in $H[i]$ do

⑯ $S_{ij}\leftarrow dis$;

⑰ end for

⑱ end for

⑲ 对称化 $\boldsymbol{S}$;

⑳ $\boldsymbol{D}\leftarrow\boldsymbol{O}_{n\times n}$($n\times n$ 的全零矩阵);

㉑ for $i=1:n$ do

㉒ $D_{ii}=\sum_{j=1}^{n}S_{ij}$;

㉓ end for

㉔ $\boldsymbol{L}=\boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}$;

㉕ $\boldsymbol{V}\leftarrow eigenFactor(\boldsymbol{L}, k)$;

㉖ $\boldsymbol{U}\leftarrow$行标准化 $\boldsymbol{V}$;

㉗ 使用 K-means 方法将矩阵 $\boldsymbol{U}$ 的 n 行聚为 k 类.

在算法 1 中,计算稀疏相似性矩阵、求前 k 个最大特征值对应的特征向量和 K-means 聚类这 3 个步骤(以下分别称为 TNN, EVD, KMEANS)花费的时间最多,这 3 个步骤都被设计为能够充分利用 CPU-GPU 异构系统的性能.

2.2 TNN

在 TNN 步骤中,通过分块的方式计算稀疏相似性矩阵 $\boldsymbol{S}$. 该过程首先为每个样本点初始化一个容量为 t 的最大堆,将数据集 X 划分为若干固定大小的分块,分块大小记为 $batch$. 对每一分块,由 GPU 计算其中样本点与其后所有分块中的样本点的距离,由 CPU 计算该分块中不同样本点之间的距离,每对样本点计算出的距离将被 2 次插入到不同的最大堆中. 计算结束后,每个样本点对应的最大堆中保留了与该样本点距离最近的 t 个其他样本,即 t 个最近邻居. 通过分块,TNN 步骤能够处理的数据集规模不会受限于 GPU 设备内存;同时 PCIe 数据传输能够与 GPU 计算及 CPU 计算同时进行,提高了硬件利用率.

通过最大堆得到的相似性矩阵可能是非对称的,因此需要进行对称化,即在每个非零元素的对称位置补充同样的非零元,得到对称的相似性矩阵 $\boldsymbol{S}$. 对称化增加了 $\boldsymbol{S}$ 中的非零元,使每行最多有 $2t$ 个非零元,但 $\boldsymbol{S}$ 整体仍然是稀疏的.

2.3　EVD

EVD 步骤采用 IRLM 求对称稀疏相似性矩阵 $\boldsymbol{S}$ 的前 k 个特征值. IRLM 是隐式重启动 Arnoldi 方法(implicitly restarted Arnoldi method, IRAM)对于对称矩阵的优化算法,IRAM 常用于求解一般稀疏矩阵的部分特征值问题. 与其他矩阵特征值方法(如幂方法)相比,IRLM 具有不破坏源矩阵的稀疏结构、空间占用正比于 $k\times n$(n 为矩阵规模)、能够按多种标准(数值最大或最小、绝对值最大或最小等)筛选特征值的优点,因此非常适合求解大规模稀疏矩阵部分特征值问题.

IRLM 用迭代的方式求矩阵的前 k 个特征值和特征向量的近似值. 每个近似特征值称为黎兹值(Ritz value),和对应的近似特征向量合起来称为一个黎兹对(Ritz pair). IRLM 的具体过程见算法 2. 为求前 k 个黎兹值,IRLM 从一个构造好的 m 步 Lanczos 分解($m>k$)开始,迭代至前 k 个黎兹对全部收敛时结束. 若未全部收敛,则选取最不希望的 $m-k$ 个黎兹值作为偏移(shift)进行 QR 分解,弱化这些黎兹值在 Lanczos 分解中的信息. 然后进行重启动过程:取 m 步 Lanczos 分解的前 k 列并将其扩展为一个新的 m 步 Lanczos 分解. 重启动过程使 IRLM 不会占用过多的存储空间,并且有助于加速黎兹对的收敛.

算法 2. IRLM 算法.

输入:$(\boldsymbol{A},\boldsymbol{V}_m,\boldsymbol{H}_m,\boldsymbol{f}_m)$,满足 $\boldsymbol{AV}_m=\boldsymbol{V}_m\boldsymbol{H}_m+\boldsymbol{f}_m\boldsymbol{e}_m^{\mathrm{T}}$,一个 m 步 Lanczos 分解;

输出:k 个黎兹对$\{(\boldsymbol{x}_1,\lambda_1),(\boldsymbol{x}_2,\lambda_2),\cdots,(\boldsymbol{x}_k,\lambda_k)\}$,即矩阵 $\boldsymbol{A}$ 的前 k 个特征值和对应特征向量的近似值;

函数:$qrFactor(\boldsymbol{T})$—计算三角矩阵 $\boldsymbol{T}$ 的 QR 分解.

① until $\|\boldsymbol{Ax}_i-\lambda_i\boldsymbol{x}_i\|<\varepsilon$, $\forall i=1,2,\cdots,k$ do
② 　求 $\boldsymbol{H}_m$ 的全部特征值 $\sigma(\boldsymbol{H}_m)$;
③ 　选择 p 个最不期望的特征值 $\{\sigma_1,\sigma_2,\cdots,\sigma_p\}$,$p=m-k$;
④ 　$\boldsymbol{q}^{\mathrm{T}}\leftarrow\boldsymbol{e}_m^{\mathrm{T}}$;
⑤ 　for $j=1,2,\cdots,p$ do
⑥ 　　$[\boldsymbol{Q}_j,\boldsymbol{R}_j]=qrFactor(\boldsymbol{H}_m-\sigma_j\boldsymbol{I})$;
⑦ 　　$\boldsymbol{H}_m\leftarrow\boldsymbol{Q}_j^{\mathrm{T}}\boldsymbol{H}_m\boldsymbol{Q}_j$;
⑧ 　　$\boldsymbol{V}_m\leftarrow\boldsymbol{V}_m\boldsymbol{Q}_j$;
⑨ 　　$\boldsymbol{q}\leftarrow\boldsymbol{q}^{\mathrm{T}}\boldsymbol{Q}_j$;
⑩ 　end for
⑪ 　$\boldsymbol{f}_k\leftarrow\boldsymbol{V}_m(:,k+1)\times\boldsymbol{H}_m(k+1,k)+\boldsymbol{f}_m\boldsymbol{q}$;
⑫ 　$\boldsymbol{V}_k\leftarrow\boldsymbol{V}_m(1:n,1:k)$;
⑬ 　$\boldsymbol{H}_k\leftarrow\boldsymbol{H}_m(1:k,1:k)$;
⑭ 　将所得到的 k 步 Lanczos 分解 $\boldsymbol{AV}_k=\boldsymbol{V}_k\boldsymbol{H}_k+\boldsymbol{f}_k\boldsymbol{e}_k^{\mathrm{T}}$ 扩展为一个新的 m 步 Lanczos 分解;
⑮ end until

2.4　KMEANS

KMEANS 步骤中使用 K-means 方法对标准化矩阵 $\boldsymbol{U}$ 的 n 行进行聚类. K-means 方法首先随机选择 k 个初始的聚簇中心,然后开始迭代过程. 每步迭代计算 k 个聚簇中心与 n 个数据点(即矩阵 $\boldsymbol{U}$ 的 n 行)之间的距离,将每个数据点分配给距离最近的中心,然后重新计算每个聚簇中心的位置,当各中心位置不再变动时停止迭代.

2.5　算法复杂度分析

PSCH 算法中,计算 Laplacian 矩阵 $\boldsymbol{L}$ 和标准化矩阵 $\boldsymbol{U}$ 等操作开销较小,下面分别对 TNN,EVD,KMEANS 三个计算成本较高的步骤进行分析.

1) TNN 步骤. 该步骤计算所有样本点对间的距离,但只保留与每个样本点 $\boldsymbol{x}_i$ 最相似的 t 个样本点. 在大小为 t 的堆中插入或删除 1 个元素的复杂度是 $O(\log t)$,故 TNN 步骤的时间复杂度为 $O(n^2d+n^2\log t)$. TNN 需要维护 n 个大小为 t 的堆,故空间复杂度为 $O(nt)$.

2) EVD 步骤. 该步骤采用 IRLM 求前 k 个最大特征值对应的特征向量,IRLM 需要由用户指定一个不小于 k 的值 m 进行迭代. 每次迭代将一个 Lanczos 分解扩展最多 $m-k$ 步,每一步中需要对最大 $n\times m$ 的稠密矩阵进行矩阵向量乘、对稀疏矩阵 $\boldsymbol{L}$ 进行矩阵向量乘和对 $m\times m$ 的矩阵进行特征分解. 故 EVD 步骤的时间复杂度为 $(O(nm)+O(nt)+O(m^3))\times O(m-k)\times N_{\mathrm{restart}}$,其中 N_{restart} 为重启动次数. EVD 步骤需存储 $n\times m$ 的稠密矩阵,故其空间复杂度为 $O(nm)$.

3) KMEANS 步骤. 该步骤也是一个迭代过程,每步迭代的主要操作是计算 k 个聚簇中心与 n 个数据点之间的距离,其时间复杂度为 $O(nk^2)\times N_{\mathrm{iteration}}$,其中 $N_{\mathrm{iteration}}$ 为迭代次数,空间复杂度为 $O(k^2)$.

3　OpenMP/CUDA 异构计算模型

3.1　CUDA 程序设计模型

GPU 因其远高于通用 CPU 的计算能力和内存带宽,已被广泛应用于科学计算领域[23],越来越多的超级计算机采用 GPU 作为加速器[24]. CUDA 是 NVIDIA 公司提供的用于该公司 GPU 硬件的并行

程序设计工具,包括专用的语言(CUDA C)、相应的编译器和一组软件库. CUDA 允许程序员在需要时动态创建大量 GPU 线程,完成高吞吐量的计算.

CUDA 程序设计需要程序员显式地管理 GPU 的计算和存储资源. GPU 线程具有层次化的组织,32 个线程组成一个 warp,这是 GPU 调度线程的最小单位;若干 warp 组成一个线程块(thread block),一个 thread block 内的线程能够进行高效的同步及通信;若干 thread block 组成一个线程网格(grid),其中各 thread block 的执行相互独立,这种独立性保证了 CUDA 程序能够在具有不同性能的 GPU 硬件上自动扩展. 程序员每次创建一个 grid,创建 grid 的过程也称为 kernel 函数调用. CUDA 程序设计的困难之处在于如何将应用映射到线程的层次化结构上,映射的好坏决定了程序的性能. 另一方面,CUDA 提供了一组接口用于显式申请、释放显存以及在显存与内存之间进行数据传输. 由于 GPU 访问显存的带宽(200～300 GBps)远高于 CPU 访问内存的带宽(10～20 GBps),而连接 CPU 和 GPU 的 PCIe 带宽速度最慢(5～6 GBps),如何优化数据布局、避免慢速数据传输或将数据传输与计算时间重叠对提高 CUDA 程序的整体性能十分重要.

3.2 OpenMP/CUDA 异构程序设计模型

GPU 适合规则的数据并行型计算,而 CPU 对逻辑判断和不规则的计算模式的支持能够弥补 GPU 的不足. GPU 计算对 CPU 是异步的,即在 GPU 线程创建后,CPU 能够继续执行后续代码. 目前 CPU 大多包含若干计算核心,在 GPU 进行计算的同时充分利用多核心 CPU 能够显著提高 CUDA 程序的整体性能.

OpenMP[25]是用于共享内存环境并行程序设计的一种简便且高效的方式. OpenMP 提供了一组编译指导(compiler directive),对用户屏蔽了线程创建和同步的细节. 通过添加合适的编译指导,计算任务能够在多个 CPU 计算核心之间达到动态负载均衡. 在编译 CUDA 程序时通过添加特定的选项即可打开 OpenMP 支持,实现多核 CPU 与众核 GPU 的异构并行. 通过将应用中不规则的部分分配给多核 CPU,OpenMP/CUDA 异构程序设计模型能够充分利用硬件资源、提高应用程序的计算性能.

4 PSCH 算法异构并行实现

4.1 PSCH 算法计算流程

TNN,EVD,KMEANS 是基于 TNN 稀疏矩阵的谱聚类算法中消耗时间最多的 3 个步骤. 在 PSCH 算法中,以上 3 个步骤都以 CPU-GPU 协同计算的方式实现. PSCH 算法的完整流程如图 1 所示. 在并行 TNN 步骤中,使用双缓冲轮转 4 段流水机制重叠计算与数据传输,保持 CPU,GPU 以及 PCIe 总线同时处于繁忙;在并行 EVD 步骤中,空间占用最多的矩阵 $\boldsymbol{V}$ 被分配在 GPU 全局内存中,减少了跨 PCIe 慢速数据传输且能够有效利用 GPU 的高显存带宽;在并行 KMEANS 步骤中,使用 GPU 共享内存(GPU shared memory)分块计算数据点与聚簇中心之间的距离,减少了对 GPU 设备内存的访问.

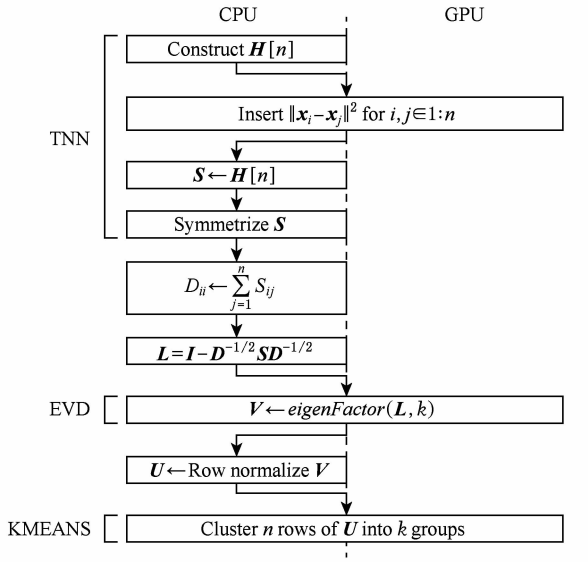

Fig. 1 PSCH algorithm procedure.

图 1 PSCH 算法计算流程

4.2 并行 TNN

TNN 步骤是 PSCH 算法中耗时最多的步骤,也是 PSCH 算法设计与优化的重点. 在 TNN 步骤中,由 GPU 计算数据集中两两样本点之间的距离,由 CPU 计算与每个样本点最相似的 t 个样本点,构造稀疏相似性矩阵 $\boldsymbol{S}$. 为了使用有限的 GPU 设备内存对较大规模的数据集进行分类,必须对数据集进行分块. 非流水并行实现即采用分块策略的并行 TNN 实现,能够处理超过 GPU 设备内存的数据集. 分块的另一目的是通过重叠 CPU 计算、GPU 计算与 PCIe 数据传输,提高硬件利用率. 为此,PSCH 算法中设计了双缓冲轮转 4 段流水机制. 通过采用该机制,流水并行实现能够保持 CPU,GPU 和 PCIe 总线三者同时繁忙,在打破 GPU 设备内存限制的同时提高了算法的求解性能.

4.2.1　数据分块策略

目前的 GPU 一般提供 1～12GB 不等的设备内存,较大规模的数据样本集可能达到甚至超过 GPU 的设备内存大小.另外,尽管 TNN 步骤的最终结果是稀疏相似性矩阵,但在计算过程中实际已计算出了完整的稠密相似性矩阵.稠密相似性矩阵的规模是数据样本集规模的平方,可以轻易超过普通机器的内存容量.因此,为了使用具有有限设备内存的 GPU 对较大规模的数据集进行分类,必须对数据样本集进行切分,分块地计算出稠密相似性矩阵的每一部分.

在并行 TNN 步骤中,将数据样本集 X 划分为 *batch* 大小的分块(以下称为 i 块,对应于算法 1 中的外层循环),分块大小 *batch* 的值经实验确定为 16.对每个 i 块,由 CPU 按式(1)计算出块内两两样本点之间的距离(即稠密相似矩阵中对角线上 *batch* ×*batch* 大小的子矩阵)并插入最大堆;同时由 GPU 计算出该分块内的样本点与位于其后的所有样本点之间的距离,即相当于稠密相似矩阵中对角线下宽度为 batch 的矩形区域.数据集中位于该分块后的部分有可能超出 GPU 设备内存的大小,在实际计算中将位于该 i 块后的部分切分成尽可能大的块(以下称为 j 块,对应于算法 1 中的内层循环),即一个 i 块对应一个或多个 j 块,每次将一个 j 块拷贝到 GPU 端进行处理.每个 j 块计算出的距离将被拷贝回 CPU 端,由 CPU 插入最大堆.

4.2.2　非流水并行实现

非流水并行实现是采用数据分块策略的并行 TNN 实现.在 GPU 端计算 i 块与 j 块间样本点相似性距离的过程中,注意到:

$$\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2=\|\boldsymbol{x}_i\|^2-2\boldsymbol{x}_i\boldsymbol{x}_j+\|\boldsymbol{x}_j\|^2,\quad(6)$$

因此 GPU 端的过程被分解为 2 步:1)以稀疏矩阵乘的方式计算 i 块与 j 块的乘积,即计算出 $\boldsymbol{x}_i\boldsymbol{x}_j$ 部分;2)计算出样本点间的相似性距离,即式(1).这样分解的目的在于:第 1 步中的矩阵乘能够借助于已有的高性能计算软件库,在简化整体实现的同时能够达到较高的性能.在 TNN 步骤中调用 cuSPARSE 完成稀疏矩阵乘,使用 cuSPARSE 计算稀疏矩阵乘时要求参与计算的矩阵之一必须是稠密的,因此在并行 TNN 步骤中将每个 i 块拷贝到 GPU 端后展开为稠密矩阵.第 2 步在稀疏矩阵乘后由一个单独调用的 kernel 函数完成,其中每个 GPU 线程完成一个元素 S_{ij} 的相似性距离计算.计算过程中用到的所有样本点的二范数 $\|\boldsymbol{x}_i\|^2$ 在并行 TNN 步骤开始阶段一次性计算完成并拷贝到 GPU 端.

非流水并行实现中对每个 i 块的具体计算过程如下:

1) 将该 i 块拷贝到 GPU 端并展开为稠密矩阵;

2) 根据 GPU 显存大小及位于该 i 块后的样本点的数目决定 j 块大小及数量;

3) 对每个 j 块,将其拷贝到 GPU 端;

4) 由 GPU 完成 i 块与 j 块中样本点间的距离计算;

5) 将距离计算的结果拷贝回 CPU 端;

6) 由 CPU 将距离计算的结果插入最大堆中,继续计算其他 j 块.

每个 i 块内两两样本点间的距离计算在 TNN 步骤开始阶段在 CPU 端以 OpenMP 多线程的方式一次性完成.计算 i 块内两两样本点间的距离、计算所有样本点的二范数 $\|\boldsymbol{x}_i\|^2$ 及拷贝和展开 i 块等步骤,在计算的整体过程中占用时间极少,非流水并行实现的时间消耗主要来自于拷贝 j 块、GPU 距离计算、回拷相似性结果及插入最大堆 4 个操作上.

通过分块,非流水并行实现能够打破 GPU 设备内存的局限,实现对主存规模的数据样本集的聚类;但是 CPU 计算、GPU 计算以及 PCIe 数据传输三者是串行的,每个部件在整个计算过程中有相当比例的时间保持空闲.通过分析非流水并行实现中的对每个 j 块的计算模式,PSCH 算法中设计了双缓冲轮转 4 段流水机制,以达到通过重叠 CPU 计算、GPU 计算与数据传输提高硬件利用率和整体计算性能的目的.

4.2.3　双缓冲轮转 4 段流水机制

尽管非流水并行实现仅利用数据分块策略达到了打破 GPU 设备内存局限的目的,但数据分块策略也提供了通过重叠计算与通信提高算法性能的可能.双缓冲轮转 4 段流水机制通过重叠 j 块计算过程中的各步骤,实现了 CPU 计算、GPU 计算与数据传输三者的重叠,能够大幅度提高并行 TNN 步骤的整体性能.

一个 j 块的计算过程主要分为数据拷贝至 GPU、GPU 距离计算、数据回拷至 CPU 和 CPU 插入最大堆这 4 个操作,占用 CPU,GPU 以及 PCIe 总线 3 种资源,如图 2 所示.不同 j 块的计算是相互独立的,因此,如果连续 j 块的计算能够以流水的方式完成,那么就可以将 4 个计算操作重叠在一起,提高计算效率.虽然 4 个操作中有 2 次数据拷贝,但由于 GPU 距离计算操作的时间复杂度较高,对足够大的分块,计算时间将超过通信时间.在并行 TNN

步骤中,大部分情况下 2 次数据拷贝步骤的时间之和明显少于距离计算或插入最大堆操作的时间,因此数据传输能够完全与计算重叠.

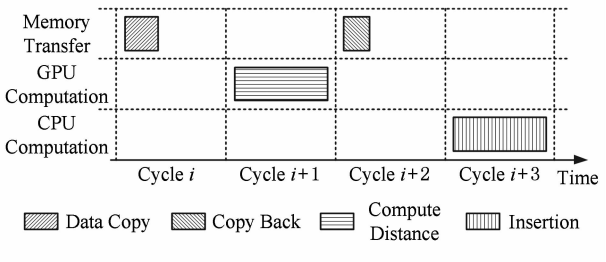

Fig. 2 Procedure for computing a j-block.

图 2 一次 j 块计算过程

在双缓冲轮转 4 段流水机制中,一个 j 块的计算过程被视为 4 阶段的流水线,每个阶段有自己的输入缓冲及输出缓冲,上一阶段的输出缓冲即为下一阶段的输入缓冲.但同一缓冲区无法同时被上一阶段写和被下一阶段读,为了解决相邻阶段之间的读写依赖,在双缓冲轮转 4 段流水机制中为每个流水阶段设置轮转使用的双缓冲.例如,距离计算阶段的输入缓冲(即数据拷贝阶段的输出缓冲)有 2 份,记为缓冲 A 与缓冲 B.假设在周期 i 将某个 j 块拷贝到缓冲 A,此时缓冲 B 中已经有上个 j 块的数据正在被读取以进行距离计算,那么在周期 $i+1$ 则会将下个 j 块拷贝到缓冲 B,缓冲 A 则被读取以进行距离计算.尽管流水线填满后每个周期有 4 个不同的 j 块在被处理,但轮转使用缓冲区的方式只需要为每个阶段设置 2 份缓冲,减少了缓冲区的数量,因而可以支持较大的 j 块.

使用双缓冲轮转 4 段流水机制处理 n 个 j 块需要 $n+3$ 个周期,这些周期分为启动、循环及排空 3 个阶段,如图 3 所示.图 3 中表示操作的方格中的数字表示该操作对应的 j 块的序数.例如,图 3(a) GPU Computation 一行中的"1"表示正在进行 GPU Computation 操作,即 GPU 距离计算的是第 1 个 j 块.第 i 个 j 块参与从第 i～$i+3$ 共 4 个周期.在循环阶段,每个周期都对 4 个不同的 j 块执行 4 种计算步骤,保持 CPU,GPU 及 PCIe 总线 3 个部件繁忙,减少了硬件空闲时间.

4.2.4 流水并行实现

流水并行实现即采用双缓冲轮转 4 段流水机制的并行 TNN 实现.为了实现数据拷贝、GPU 计算与 CPU 计算的重叠,2 个 CUDA 流(拷贝流与计算流)在 CPU 端被创建.在流水线每个周期开始时,首先同步 2 个 CUDA 流等待上一周期的操作完成,

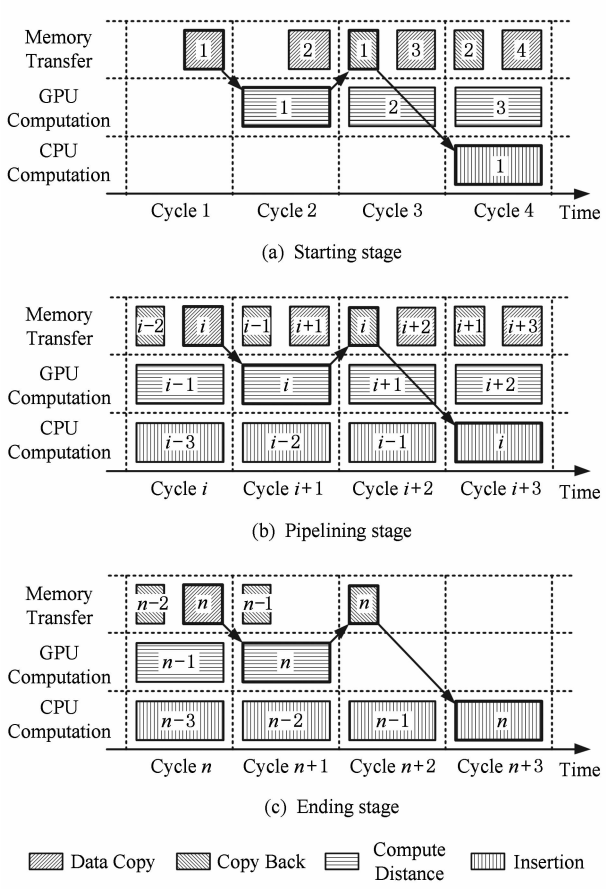

Fig. 3 Parallel pipelined computing procedure.

图 3 并行流水计算过程

然后将 2 次数据拷贝及 GPU 计算操作压入相应的 CUDA 流中,最后在 CPU 端以 OpenMP 多线程的方式完成最大堆插入步骤.所有参与数据拷贝的 CPU 端的缓冲区都被分配为页锁定内存(page-locked host memory)以使拷贝操作能够以异步的方式被压入 CUDA 流中.

计算一个 j 块前,需要将对应的 i 块拷贝至 GPU 并展开为稠密矩阵.在 GPU 端分配有以轮转方式使用的 2 份稠密 i 块缓冲区,但由于一个 i 块对应一个或多个 j 块,稠密 i 块缓冲区的轮转是独立于双缓冲轮转 4 段流水机制的.当一个 i 块被拷贝至某一稠密 i 块缓冲区后,该缓冲区将被一直用于距离计算操作,直到对应的全部 j 块都被拷贝至 GPU.新的 i 块将会在下一 j 块拷贝前被拷贝至另一缓冲区,而原 i 块所在的缓冲区将继续参与一轮距离计算操作.

通过引入双缓冲轮转 4 段流水机制,并行 TNN 的流水并行实现不仅能够打破设备内存的限制处理任意规模的数据集,而且能够充分利用 CPU,GPU 以及 PCIe 总线达到较高的计算性能.

4.3　并行 EVD

在 PSCH 算法的 EVD 步骤中,采用 HLanc 求解 Laplacian 矩阵 $\boldsymbol{L}$ 的前 k 个特征值[12]. HLanc 是基于 CUDA 的 IRLM 异构并行实现,由于 IRLM 中与稀疏矩阵有关的唯一操作是在扩展 Lanczos 分解时进行稀疏矩阵向量乘,因此 HLanc 被设计为相互分离的若干 IRLM Solver 和若干 SPMV Operator,且一个 IRLM Solver 可以与任意 SPMV Operator 组合使用. SPMV Operator 处理在特定硬件上对某种具体稀疏格式进行矩阵向量乘的细节;而 IRLM Solver 实现了 IRLM 算法中与稀疏矩阵无关的部分,并调用某个 SPMV Operator 进行稀疏矩阵向量乘操作. 这种设计使 HLanc 能够处理任意格式的稀疏矩阵,只要 HLanc 中已经实现或用户能够提供一个用于特定稀疏格式的 SPMV Operator.

HLanc 中实现了仅使用 CPU 和使用 CPU+GPU 两种不同的 IRLM Solver,它仅包含 IRLM 算法中的稠密部分,其中最耗时的操作是对矩阵 $\boldsymbol{V}$ 进行的稠密矩阵向量乘. 在使用 CPU+GPU 的 IRLM Solver 中,矩阵 $\boldsymbol{V}$ 被分配在 GPU 的设备内存中,所有对矩阵 $\boldsymbol{V}$ 的操作都在 GPU 端进行,最终结果被拷贝回 CPU 端. 由于稠密矩阵向量乘是内存瓶颈型操作,在 GPU 端进行该操作能够利用 GPU 的高带宽优势,大幅度提升 IRLM 的整体性能. 而 IRLM 中的其他操作,如 QR 分解、排序等,都是在 CPU 端通过调用 Intel 数学核心函数库(math kernel library, MKL)实现的.

在 PSCH 算法实现过程中,并行 EVD 步骤是使用 HLanc 中 CPU+GPU 的 IRLM Solver 和在 GPU 上进行 CSR 格式稀疏矩阵向量乘的 SPMV Operator 实现的.

4.4　并行 KMEANS

在 PSCH 算法的 KMEANS 步骤中,使用 K-means 算法将标准化矩阵 $\boldsymbol{U}$ 的 n 行聚为 k 类. K-means 算法首先随机选择 k 个初始的聚簇中心,然后进行更新聚簇中心的迭代过程,直到所有聚簇中心不再发生变化[26]. 选择初始聚簇中心的计算是在 CPU 端以 OpenMP 多线程的方式完成的.

K-means 每步迭代的主要操作是计算 k 个聚簇中心与矩阵 $\boldsymbol{U}$ 的每行之间的距离,其计算模式较为规则,且计算量较大. PSCH 算法的 KMEANS 步骤在 GPU 上以类似矩阵-矩阵乘的方式实现了 K-means 算法的每步迭代. 由于表示聚簇中心的矩阵 $\boldsymbol{C}$ 中的每个元素都被多次重复使用,故将矩阵 $\boldsymbol{C}$ 划分为 $p\times p$ 固定大小的子矩阵,每次将一个子矩阵拷贝至 GPU 的共享内存中进行计算,以减少对高延迟的 GPU 设备内存的访问. 每个 GPU 线程按式(7)计算距离矩阵中的一行:

$$dis_{ij}=\sum_{k}(U_{ik}-C_{jk})^2, \tag{7}$$

然后,在 GPU 端求出该行的最小距离对应的聚簇中心的索引. GPU 计算完成后,将 n 个索引值拷贝到 CPU 端,由 CPU 更新聚簇中心矩阵 $\boldsymbol{C}$ 后进行下一轮或停止迭代过程.

通过将 TNN,EVD,KMEANS 这 3 个步骤以 CPU-GPU 协同计算的方式实现,PSCH 算法能够充分利用单节点内的众核加速器的高计算性能和高带宽优势,达到远超过单节点或 CPU 集群上已有谱聚类实现的性能.

5　实验结果与分析

5.1　实验准备

用于评估 PSCH 算法性能的实验环境为一台 PC 机,硬件配置为 Intel® Core™ i7-2600 CPU(物理 4 核)、8 GB 内存以及 1 颗 Fermi 架构的 NVIDIA GTX 480 GPU(1.5 GB 显存),通过 PCIe x16 总线与主板接连. 软件环境为 Ubuntu 12.04.3,GCC 4.6.3,CUDA Toolkit 6.5 和 Intel Parallel Studio XE 2013. 用于运行与 PSCH 算法对照的基于 MPI 的并行谱聚类 PSC 算法[9]的环境为 4 台以上配置的 PC 机组成的集群,节点间通过千兆以太网进行通信.

5 个具有不同属性的数据集:corel,RCV1,covtype,kddb,picasa 被选择用来评估 PSCH 算法. 这 5 个数据集来自 UCI 知识库、Delve 数据库和 Google 的 Picasa 图片数据集,每个数据集可以被看作一个稀疏矩阵. 各数据集的属性如表 1 所示,其中稠密度定义为平均每个样本的非零元数目. 规模最大的 picasa 数据集读入后需要约 2 GB 的内存空间,已经超过 GTX 480 的显存大小.

Table 1　Attributes of the Datasets

表 1　数据集属性

Datasets	Samples	Features	Density
corel	2 074	144	135
RCV1	193 844	47 236	74
covtype	581 012	54	32
kddb	826 048	29 890 095	23
picasa	1 730 897	144	94

5.2 实验结果及分析

本文从聚类质量、流水机制效率和程序总体性能3个方面对PSCH算法进行评估. 1)在聚类质量实验中,通过对比E-kmeans算法、Fixed-σ SC算法[9]与PSCH算法对corel和RCV1数据集的聚类结果评估PSCH算法的聚类准确程度. 2)在流水机制效率验证实验中,通过分别统计并行TNN中各步骤的时间开销并与总时间消耗进行比较,评估并行TNN步骤中流水并行机制的效率. 3)在总体性能实验中,使用规模较大的4个数据集,将PSCH算法的性能与基于MPI的并行谱聚类PSC算法进行对比,以说明使用CPU-GPU异构系统的PSCH算法相对于仅使用CPU的并行谱聚类实现的性能优势.

5.2.1 聚类质量

聚类质量实验使用corel和RCV1作为评价聚类质量的测试数据,将PSCH算法的聚类结果与E-kmeans算法和TNN谱聚类的Matlab版本实现Fixed-σ SC算法的结果进行比较,通过标准化互信息(normalized mutual information, *NMI*)和聚类精度(clustering accuracy, *CA*)2个指标衡量PSCH算法的聚类质量.

*NMI*用于衡量2个随机变量*cat*(category label)和*cls*(cluster label)之前的相关程度,其定义为

$$NMI(cat;cls)=\frac{I(cat;cls)}{\sqrt{H(cat)H(cls)}},\qquad(8)$$

其中,$I(cat;cls)$是cat与cls的互信息,用于标准化互信息的熵$H(cat)$和$H(cls)$都在[0,1]范围内.

实际中使用式(9)来计算*NMI*值[27]:

$$NMI=\frac{\sum_{i=1}^{k}\sum_{j=1}^{k}n_{ij}\ln\left(\frac{n_{ij}n}{n_in_j}\right)}{\sqrt{\left(\sum_{i=1}^{k}n_i\ln\frac{n_i}{n}\right)\left(\sum_{j=1}^{k}n_j\ln\frac{n_j}{n}\right)}},\qquad(9)$$

其中,n是样本的数量;n_i和n_j代表了类别i和类别j中样本的数量;n_{ij}则代表同时在类别i和类别j中样本的数量. 如果聚类结果能够完全匹配类别标签,$NMI=1$;如果样本随机分配,则*NMI*值接近0[28]. *NMI*值越高,说明聚类质量越高.

*CA*是所有样本聚类正确性的平均值[29],其定义为

$$CA=\sum_{i=1}^{n}\delta(y_i,map(c_i))\Big/n,\qquad(10)$$

其中,y_i和c_i代表样本的真实类别标号和聚类标号;$map(\cdot)$是一个置换函数,它将每个聚类标号映射到一个类别标号;$\delta(a,b)$为比较函数,如果$a=b$,则$\delta(a,b)=1$,否则为0. 与*NMI*类似,*CA*值越高,则说明聚类质量越高.

在聚类质量实验中,将corel数据集聚为18类,RCV1聚为103类. 3种算法的实验结果见表2,其中E-kmeans算法与Fixed-σ SC算法的结果来自文献[9]. Fixed-σ SC算法和PSCH算法都包含K-means过程,而K-means聚类的结果与随机选择的初始聚簇中心有关,故表2中给出的数据都是10次以上运行结果的平均值,并同时给出了标准差. Fixed-σ SC算法和PSCH算法的*NMI*和*CA*值均略高于E-kmeans算法,说明2种谱聚类算法均略优于E-kmeans算法. 由于PSCH算法和Fixed-σ SC算法本质上都是基于稀疏矩阵的TNN谱聚类算法,因此聚类质量十分接近. 表2说明PSCH算法能够达到同类TNN谱聚类算法的聚类质量.

Table 2 Clustering Quality Analysis

表2 聚类质量分析

Method	Algorithm	*CA*(Standard Deviation)	
		corel	RCV1
NMI	E-kmeans	0.368 9(±0.012 2)	0.273 7(±0.006 3)
	Fixed-σ SC	0.381 1(±0.005 0)	0.286 1(±0.001 0)
	PSCH	0.381 0(±0.005 4)	0.286 1(±0.001 5)
CA	E-kmeans	0.358 7(±0.025 3)	0.165 9(±0.006 2)
	Fixed-σ SC	0.382 6(±0.008 6)	0.185 5(±0.002 5)
	PSCH	0.382 6(±0.009 0)	0.185 5(±0.002 8)

5.2.2 流水机制效率验证

流水机制效率实验用于评估并行TNN步骤中双缓冲轮转4段流水机制的效率. 实验选取RCV1作为测试数据集,设置$t=100$,分别统计TNN步骤中4种主要操作(数据拷贝、数据回拷、距离计算、插入最大堆)的累计时间开销,并将流水前和流水后的运行时间进行比较,具体结果见表3. 其中,流水前与流水后的时间未包括TNN步骤中读取并初始化数据、将结果数据写入文件的时间,因此比完整的TNN步骤的时间略短.

Table 3 Running Time of the Pipeline Mechanism

表3 流水机制运行时间 s

Operation	Running Time
Data Copy	123.19
Copy Back	23.24
Compute Distance	285.69
Insertion	224.48
Without Pipeline	670.81
With Pipeline	314.15

流水前的运行时间为 670.81 s,略长于 4 种操作的时间之和;而流水后的时间缩短为 314.15 s,略长于最复杂操作花费的时间,即距离计算操作花费的时间.这说明 TNN 步骤中的双缓冲轮转 4 段流水机制有效地并行了各个操作,使 CPU-GPU 通信(数据拷贝、数据回拷)、GPU 计算(距离计算)与 CPU 计算(插入最大堆)三者在时间上有较大程度的重叠.通过 CPU,GPU 以及 PCIe 总线的并行工作,双缓冲轮转 4 段流水机制能够有效地减少 CPU-GPU 异构系统中各部分的空闲时间,提高系统的整体利用率和 TNN 步骤的整体性能.

5.2.3　PSCH 算法总体性能

在总体性能实验中,使用规模较大的 4 个数据集 RCV1,covtype,kddb,picasa 评估 PSCH 算法的总体性能,并通过与基于 MPI 的并行谱聚类 PSC 算法[9]对比说明 PSCH 算法的性能优势.其中,PSC 算法运行在 4 个计算节点组成的集群上,使用单节点单进程的串行方式(PSC 1×1)和每个计算节点启动一个进程(PSC 4×1)、每个计算节点启动 4 进程(PSC 4×4)这 2 种并行方式运行.程序运行时间如表 4 所示,其中 PSC 算法的串行方式由于运行时间过长,仅对规模较小的 RCV1 和 covtype 数据集进行了测试.PSCH 算法中计算 Laplacian 矩阵的时间和计算规范化矩阵 $\boldsymbol{U}$ 的时间分别被包含在 TNN 和 KMEANS 两个步骤中.

Table 4　Running Time by Step

表 4　分步运行时间　　s

Datasets	Version	TNN	EVD	KMEANS	Total
RCV1	PSC 1×1	19 449.72	250.39	25.78	19 725.89
	PSC 4×1	5 228.23	75.53	6.86	5 310.62
	PSC 4×4	1 716.59	33.07	2.41	1 752.07
	PSCH	344.81	44.49	2.51	391.81
covtype	PSC 1×1	34 475.91	7 015.24	63.54	41 554.69
	PSC 4×1	10 807.49	2 119.41	16.90	12 943.80
	PSC 4×4	3 976.46	1 259.46	7.33	5 243.25
	PSCH	2 125.83	488.27	5.60	2 619.70
kddb	PSC 1×1				
	PSC 4×1	39 170.31	3 294.20	32.63	42 497.14
	PSC 4×4	13 541.25	1 957.58	14.15	15 512.98
	PSCH	4 123.27	242.28	6.30	4 371.85
picasa	PSC 1×1				
	PSC 4×1	230 432.63	2 456.75	101.27	232 990.65
	PSC 4×4	123 871.94	2 447.03	32.54	126 351.51
	PSCH	51 974.32	195.74	21.23	52 191.29

1)对于 4 个数据集,PSCH 算法的性能都要明显好于 PSC 算法,说明使用 CPU-GPU 异构系统的 PSCH 算法相对于仅使用 CPU 的多机并行谱聚类实现 PSC 算法有明显的性能优势.2)PSCH 算法的性能提升主要来自于耗时最多的 TNN 步骤.对数据集 RCV1,PSCH 算法中 TNN 步骤的性能大约是串行方式的 PSC 算法的 56 倍;对数据集 kddb,PSCH 算法中 TNN 步骤的性能大约是使用 16 进程的 PSC 算法的 3.3 倍.这说明 PSCH 算法中的双缓冲轮转 4 段流水机制能够有效利用 CPU-GPU 异构系统的高计算性能,大幅度减少计算时间.3)EVD 步骤花费的时间占总体运行时间的比重远小于 TNN 步骤,但在 CPU-GPU 异构系统上实现 TNN 步骤后,EVD 步骤就会成为明显的性能瓶颈.对数据集 covtype,串行 EVD 花费的时间甚至大于 PSCH 算法中的 TNN 步骤.EVD 步骤的核心操作是内存瓶颈型的稀疏及稠密的矩阵向量乘,PSCH 算法中使用 HLanc 实现了 CPU-GPU 异构系统上的 EVD 步骤,对规模最大的 picasa 数据集,PSCH 算法中 EVD 步骤的性能是使用 16 进程的 PSC 算法的 12.5 倍.这个结果说明通过调用 HLanc,PSCH 算法中的 EVD 步骤能够有效利用 GPU 的高显存带宽,减少计算时间.对数据集 RCV1,PSCH 算法的 EVD 步骤花费的时间略长于使用 16 进程的 PSC 算法,原因主要有 2 个方面:①RCV1 的 EVD 步骤串行部分比重较大,在 44.49 s 中,超过 20 s 的时间花费在读写输入、输出数据文件上,能够并行的计算部分所占比重较少;②由于 GPU 与 CPU 浮点精度的差异,对数据集 RCV1,PSCH 算法的 EVD 步骤比 PSC 算法多迭代了若干次才达到收敛,因此花费了较多时间.4)KMEANS 步骤占总体时间的比重较小,但 PSCH 算法的 KMEANS 步骤仍然比串行的 PSC 算法有大幅度的性能提升,对除规模最小的 RCV1 以外的 3 个数据集,PSCH 算法的 KMEANS 步骤性能明显超过了使用 16 进程的 PSC 算法.

图 4 是 4 个数据集总运行时间的直观表示.在仅使用单节点、单进程的情况下,对于规模较小的 RCV1 和 covtype 数据集,PSCH 算法取得了相对于 PSC 算法 15 倍以上的加速.在 PSC 算法使用 16 进程的情况下,对 4 个数据集 PSCH 算法的加速比分别为 4.5,2.0,3.5,2.4,性能优势明显.对 covtype 和 picasa 数据集加速比较低的原因在于,这 2 个数据集非零元素较为稠密,因而在占用时间最长的 TNN 步骤加速比较低.Chen 等人[9]给出了 PSC

算法在256节点的集群上对规模最大的picasa数据集进行聚类所需的时间(62 263 s),较为接近单机PSCH算法的结果.但PSC算法的一个问题是其并行效率会随所使用的节点数目增多而下降.使用256节点的PSC算法只能取得相对于单节点204.5倍的加速.而且,PSCH算法运行在单机环境,其运行及部署较之于需要MPI集群的PSC算法都更为简单和方便.

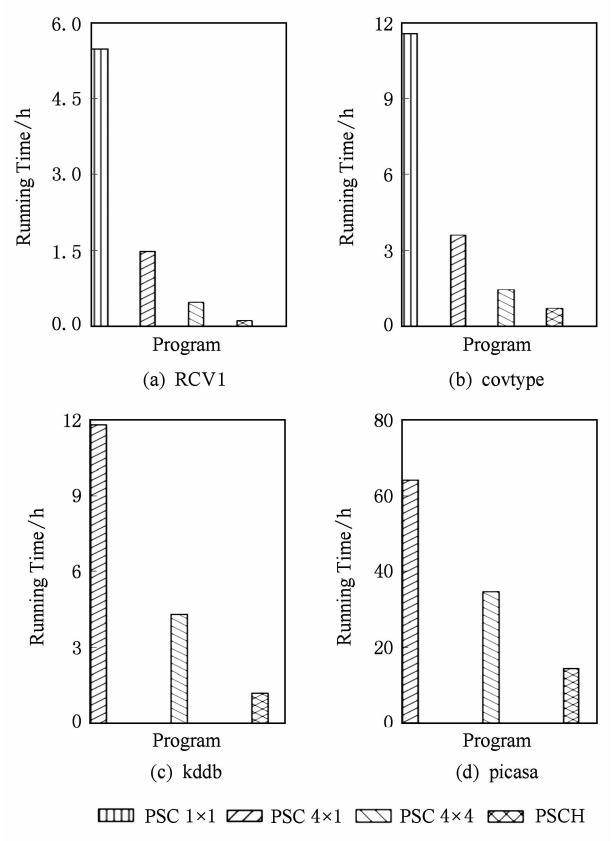

Fig. 4 Running time for large-scale datasets.
图4 大规模数据集运行时间

6 结 论

本文提出为CPU-GPU异构系统设计的基于TNN的谱聚类算法PSCH及其异构并行实现,PSCH算法充分考虑了如何发挥CPU-GPU异构系统的高计算性能和高显存带宽优势.在TNN步骤中,通过双缓冲轮转4段流水机制重叠计算与通信,使CPU,GPU和PCIe总线保持繁忙,在打破GPU设备内存限制的同时达到了较高的计算性能;在EVD步骤中,使用HLanc进行特征分解,解决了TNN步骤加速后的性能瓶颈.PSCH算法在单节点上能够对百万以上规模的数据集进行聚类,并对实验中的4个数据集取得了相对于使用16进程的MPI并行谱聚类PSC算法2.0~4.5倍的性能.

对PSCH算法的后续研究包括:扩展PSCH算法,使其能够利用单节点中的多GPU进一步提升性能;扩展PSCH算法到GPU集群环境,使其能够处理Billion级别的数据集.

参 考 文 献

[1] Hagen L, Kahng A B. New spectral methods for ratio cut participating and clustering [J]. IEEE Trans on Computed-Aided Design, 1992, 11(9): 1074-1085

[2] Ng A Y, Jordan M I, Weiss Y. On spectral clustering: Analysis and an algorithm [G] //Advances in Neural Information Processing Systems. Vancouver, CA: NIPS Foundation, 2002: 849-856

[3] Li Qiao, He Hui, Fang Binxing, et al. Awareness of the network group anomalous behaviors based on network trust [J]. Chinese Journal of Computers, 2014, 1(1): 1-14 (in Chinese)
(李乔, 何慧, 方滨兴, 等. 基于信任的网络群体异常行为发现[J]. 计算机学报, 2014, 1(1): 1-14)

[4] Sundaram N, Keutzer K. Long term video segmentation through pixel level spectral clustering on GPUs [C] //Proc of IEEE Int Conf on Computer Vision. Piscataway, NJ: IEEE, 2011: 475-482

[5] Gou Shuiping, Zhuang Xiong, Zhu Huming, et al. Parallel sparse spectral clustering for SAR image segmentation [J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2013, 6(4): 1949-1963

[6] Fowlkes C, Chung F, Malik J. Spectral grouping using the Nyström method [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2004, 26(2): 214-225

[7] Luxburg U. A tutorial on spectral clustering [J]. Statistics and Computing, 2007, 17(4): 395-416

[8] Hefeeda M, Gao Fei, Abd-Almageed W. Distributed approximate spectral clustering for large-scale datasets [C] // Proc of the 21st Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2012: 223-234

[9] Chen Wen-Yin, Song Yangqiu, Bai Hongjie, et al. Parallel spectral clustering in distributed systems [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2011, 33(3): 568-586

[10] Zheng Jing, Chen Wenguang, Chen Yurong, et al. Parallelization of spectral clustering algorithm on multi-core processors and GPGPU [C] //Proc of the 13th IEEE Asia-Pacific Computer Systems Architecture Conf (ACSAC 2008). Piscataway, NJ: IEEE, 2008: 1-8

[11] Liu Rong, Zhang Hao. Segmentation of 3D meshes through spectral clustering [C] //Proc of the 12th Pacific Conf on Computer Graphics and Applications. Piscataway, NJ: IEEE, 2004: 298-305

[12] Zhang Shuai, Li Tao, Jiao Xiaofan, et al. HLanc: Heterogeneous parallel implementation of the implicitly restarted Lanczos method [C] //Proc of the 3rd Int Workshop on Heterogeneous and Unconventional Cluster Architectures and Applications. Piscataway, NJ: IEEE, 2014: 403-410

[13] Fielder M. Algebraic connectivity of graphs [J]. Czechoslovak Mathematical Journal, 1973, 23(2): 298-305

[14] Jordan F, Bach F R. Learning spectral clustering [G] // Advances in Neural Information Processing Systems. Vancouver, CA: NIPS Foundation, 2004: 305-312

[15] Zelnik-Manor L, Perona P. Self-tuning spectral clustering [G] //Advances in Neural Information Processing Systems. Vancouver, CA: NIPS Foundation, 2004: 1601-1608

[16] Dhillon I S, Guan Y, Kulis B. Kernel k-means: Spectral clustering and normalized cuts [C] //Proc of the 10th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2004: 551-556

[17] Verma D, Meila M. A comparison of spectral clustering algorithms, UWCSE030501[R]. Seattle, WA: University of Washington, 2003

[18] Xiao Yu, Yu Jian. A weighted self adaptive similarity measure [J]. Journal of Computer Research and Development, 2013, 50(9): 1876-1882 (in Chinese)
(肖宇, 于剑. 加权的自适应相似度度量[J]. 计算机研究与发展, 2013, 50(9): 1876-1882)

[19] Chung F R K. Spectral Graph Theory [M]. Providence, RI: American Mathematical Society, 1997

[20] Yu S X, Shi Jianbo. Multiclass spectral clustering [C] //Proc of the IEEE Int Conf on Computer Vision. Piscataway, NJ: IEEE, 2003: 313-319

[21] Zha Hongyuan, He Xiaofeng, Ding C, et al. Spectral relaxation for K-means clustering [G] //Advances in Neural Information Processing Systems. Vancouver, CA: NIPS Foundation, 2001: 1057-1064

[22] Calvetti D, Reichel L, Sorensen D C. An implicitly restarted Lanczos method for large symmetric eigenvalue problems [J]. Electronic Trans on Numerical Analysis, 1994, 2(1): 1-21

[23] Domanski L, Bednarz T, Gureyev T, et al. Applications of heterogeneous computing in computational and simulation science [J]. International Journal of Computational Science and Engineering, 2013, 8(3): 240-252

[24] TOP500. org. TOP 500 supercomputer sites [EB/OL]. [2015-02-06]. http://www. top500. org

[25] OpenMP ARB. OpenMP [EB/OL]. [2015-02-06]. http://www. openmp. org

[26] Zechner M, Granitzer M. Accelerating k-means on the graphics processor via cuda [C] //Proc of the IEEE Int Conf on Intensive Applications and Services. Piscataway, NJ: IEEE, 2009: 7-15

[27] Strehl A, Ghosh J. Cluster ensembles—A knowledge reuse framework for combining multiple partitions [J]. Journal of Machine Learning Research, 2003, 3: 583-617

[28] Zhong Shi, Ghosh J. A unified framework for model-based clustering [J]. Journal of Machine Learning Research, 2003, 4: 1001-1037

[29] Wu Mingrui, Schölkopf B. A local learning approach for clustering [G] //Advances in Neural Information Processing Systems. Vancouver, CA: NIPS Foundation, 2007: 1529-1536

Zhang Shuai, born in 1986. Received his BS degree in computational mathematics from Nankai University in 2008. PhD candidate in computer science and technology at Nankai University, China. Student member of China Computer Federation. His main research interests include parallel and distributed processing, and GPGPU computing.

Li Tao, born in 1977. Received his PhD degree in computer science and technology from Nankai University in 2007. Associate professor at the College of Computer and Control Engineering, Nankai University, China. Member of the ACM and the IEEE Computer Society, and senior member of China Computer Federation. His main research interests include parallel and distributed processing, high-performance computing and network information mining.

Jiao Xiaofan, born in 1988. Received his MS degree in computer science and technology at Nankai University in 2014. His main research interests include parallel and distributed processing, and GPGPU computing.

Wang Yifeng, born in 1991. Received his MS degree in computer science and technology from Nankai University in June, 2015. His main research interests include parallel and distributed processing, and GPGPU computing.

Yang Yulu, born in 1961. Received his MS and PhD degrees from Keio University, Japan in 1993 and 1996, respectively. Professor at the College of Computer and Control Engineering, Nankai University, China. His main research interests include interconnection network, and parallel and distributed processing.