基于 GPU 的并行协同过滤算法*

许 建[1], 林 泳[2], 秦 勇[3], 黄 翰[2]
(1. 清远职业技术学院 网络信息中心, 广东 清远 511510; 2. 华南理工大学 软件学院, 广州 510006; 3. 东莞市理工学院 计算机学院, 广东 东莞 523808)

摘 要: 为提高协同过滤算法的可伸缩性,加快其运行速度,提出了一种基于 GPU(graphic processing unit)的并行协同过滤算法来实现高速并行处理。GPU 的运算模式采用单指令多数据流,适用于逻辑性弱、数据量巨大的运算,而这正是协同过滤算法所具有的特点。使用统一计算设备框架(compute unified device architecture,CUDA)实现了此协同过滤算法。实验表明,在中低端的 GPU 上该算法与在高端的四核 CPU 上的协同过滤算法相比,其加速比达到 40 倍以上,显著地提高了算法的可伸缩性,而算法在准确率方面也有优秀的表现。

关键词: 协同过滤; 图形处理器; 统一计算设备框架
中图分类号: TP301 **文献标志码:** A **文章编号:** 1001-3695(2013)09-2656-04
doi:10.3969/j.issn.1001-3695.2013.09.024

GPU-based parallel collaborative filtering algorithm

XU Jian[1], LIN Yong[2], QIN Yong[3], HUANG Han[2]
(1. *Network & Information Center, Qingyuan Polytechnic, Qingyuan Guangdong* 511510, *China*; 2. *School of Software, South China University of Technology, Guangzhou* 510006, *China*; 3. *School of Computer Science, Dongguan College of Technology, Dongguan Guangdong* 523808, *China*)

Abstract: In order to improve the scalability of collaborative filtering algorithm, this paper provided an algorithm of GPU based parallel collaborative filtering. GPU used single instruction multiple data (SIMD), and was proper for calculating weak logic and large amount data, which was exactly the features of collaborative filtering. It designed the whole procedure of the GPU based parallel collaborative filtering and implemented it using CUDA. The experimental results show the algorithm is very efficient to process the large-scale datasets with good accuracy.

Key words: collaborative filtering; GPU; CUDA framework

0 引言

协同过滤作为当前推荐系统中广泛应用的、最成功的推荐算法,已经成为推荐系统的核心组成部分和学术界的主要研究内容[1~3]。作为协同过滤算法研究的一大挑战,可伸缩性问题是指随着用户和项目的数量急剧增加,算法运行效率变得太低以致不能满足生产环境的需要[4]。国内外的很多学者都对协同过滤的可伸缩性问题进行了研究,提出了很多有效的方法,取得了不错的研究成果,但这些方法都有其不足之处。

一种方法是采用聚类技术,通过减小最近邻的搜寻空间来提高协同过滤的可伸缩性。根据聚类对象(项目或者用户)和聚类结构(层次聚类或者平面聚类)的不同,相应有多种基于聚类的协同过滤算法,主要采用 K-means 聚类,也有采用 KNN 聚类[5]、人工鱼群聚类[6]、MinHash 等其他聚类方法。但总体来说,聚类技术的个性化推荐质量偏低。

另一种采用数据集缩减方法。通过对用户评分数据集进行人为缩减,可以降低数据集规模,提高运算速度,从而改善可伸缩性。最简单的方法就是对除目标用户之外的所有用户进行随机抽样作为候选邻居用户集。Li 等人[7]采用信息论的互信息(mutual information)来计算项目之间的依赖,提出基于从整个用户评分数据集中选出一个小子集来产生推荐的思想。由于数据集缩减技术略去了大量用户信息,从而影响到算法推荐质量。

还有人使用奇异值分解(singular value decomposition, SVD)降低数据集的维度[1]。维度降下来了,但在精度方面做了妥协。Kato 等人设计了一个并行的 SVD 算法,使得对偏好矩阵的奇异值分解可以在 GPU 上运行。Kato 还提出了一个使用 GPU 计算 KNN(K-nearest neighbor)的算法,结合并行的 SVD 算法,可以形成完整的 GPU 并行的协同过滤。但这个方法的本质还是 SVD 方法,它以牺牲计算的准确率为代价,来换得运算速度的提升。

综上所述,协同过滤算法的可伸缩性问题主要是数据的维度太过于庞大而硬件的运算性能有限。本文将结合 GPU 的编程模式和软件体系特点,改进传统的协同过滤算法,使其对数据的运算转换为矩阵运算,提出了基于 GPU 的并行协同过滤

收稿日期: 2013-01-31; **修回日期:** 2013-03-20 **基金项目:** 国家自然科学基金资助项目(61170193);广东省自然科学基金资助项目(S2012010010613);东莞市高等院校科研机构科技计划资助项目(2012108102035)

作者简介: 许建(1964-),男,湖南岳阳人,副教授,硕士,主要研究方向为数据仓库商业智能、数据挖掘(jianux8290@126.com);林泳(1984-),男,广东潮州人,硕士研究生,主要研究方向为数据挖掘;秦勇(1970-),男,湖南邵阳人,教授,博士,主要研究方向为计算机网络、信息安全;黄翰(1980-),男,广东汕头人,副教授,硕导,博士,主要研究方向为智能算法、智能软件.

算法。

1　问题定义与算法框架

协同过滤算法[5]针对的是推荐问题,这里对推荐问题进行了定义。

假设有一个给定的用户集合 $U=\{u_1,u_2,\cdots,u_m\}$,一个项目集合 $I=\{i_1,i_2,\cdots,i_n\}$。m 为用户的数量,n 为项目的数量。用户对项目的评分用一个 $m\times n$ 的矩阵 R 来表示,其中矩阵元素 $R_{u,i}$ 表示用户 u 对项目 i 的评分,若用户 u 对项目 i 没有评分则此元素值为 0。也称矩阵 R 为偏好矩阵,因为它表示了用户对项目的偏好情况。U_i 表示评价了项目 i 的所有用户的集合。I_u 表示被用户 u 评价过的所有项目的集合。

可以把推荐算法问题定义如下:对于一个给定的偏好矩阵 R 和用户 u 评分过的所有项目的集合 I_u,求出一个项目集合 $\hat{I}_u$ 推荐给用户 u,使得 $|\hat{I}_u|\leqslant n$(数量小于等于 n)并且 $\hat{I}_u\cap I_u=\varnothing$(两个集合没有相同项目)。

本文使用 CUDA 编程技术实现此算法,然后对算法的准确率和性能进行了测试和分析。从测试中可以看出,此算法的准确率与传统的协同过滤算法相近,但性能上却提高了 50 倍左右。算法的流程框架如图 1 所示。

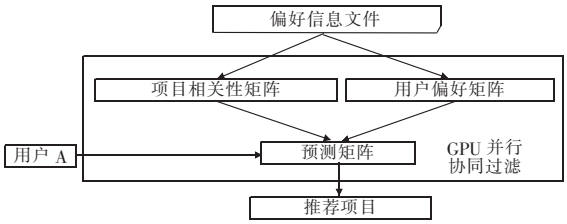

图 1　GPU 并行协同过滤推荐框架

2　基于 GPU 的并行协同过滤算法

研究提出的算法包含四个具体步骤,如图 2 所示。

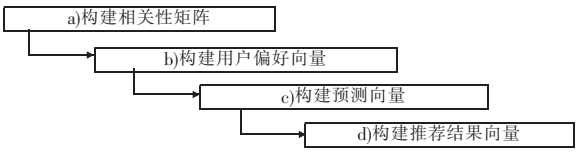

图 2　GPU 并行协同过滤推荐框架

2.1　构建相关性矩阵算法

要构建一个 $n\times n$ 的矩阵 C,行或列的数量都是项目的个数,元素 $C_{i,j}$ 的值是项目 a 和项目 b 之间的相关性。这里的相关性,是使用同时评价了这两个项目的用户数量来表示。即项目 a 和项目 b,有 x 个用户在评价了项目 a 的同时评价了项目 b,元素 $C_{a,b}$ 的值就是 x,如果没有用户同时评价这两个项目,那它的值就是 0。两个项目出现在一起的次数越大,证明它们的相关性越强,它们之间的相似性越大。该矩阵是一个对称矩阵,而且对角线表示项目被评价过的次数。矩阵的形式如式(1)所示。

$$C=\begin{bmatrix} C_{1,1} & \cdots & C_{1,n} \\ \vdots & & \vdots \\ C_{n,1} & \cdots & C_{n,n} \end{bmatrix} \tag{1}$$

矩阵 C 的每一行就是一个向量,反映了这个项目与所有其他项目之间的相关性。

为了求得矩阵 C,可以让 GPU 上的每一个线程一次负责计算两个项目之间的相关性,比如项目 a 和项目 b,线程从用户偏好矩阵 R 中分别读取这两个项目的评分列向量,由式(2)计算两个向量之间的相关性,得到的值作为矩阵 C 的元素 $C_{a,b}$ 的值。

$$C_{a,b}=\text{one}(a)\cdot\text{one}(b) \tag{2}$$

其中,one( )函数将其参数向量的所有非零元素转换为 1,并返回这个向量。

要构建一个 $n\times n$ 的矩阵 C,行或列的数量都是项目的个数,元素 $C_{i,j}$ 的值是项目 a 和项目 b 之间的相关性。

a)以 COO 形式读入数据矩阵;

b)使用 cusparseXcoo2csr 函数,把数据矩阵转换为 CSR 形式;

c)使用 cusparseScsrmm 函数,计算出矩阵乘积。

CusparseScsrmm 函数计算 $C=\text{alpha}\times\text{op}(A)\times B+\text{beta}\times C$,这里把 beta 设置为 0,而 alpha 设置为 1,并且在读入偏好矩阵的时候,让其行代表项目、列代表用户,op(A)选择为倒置。这样应用这个函数运算,可以轻易地得到相关性矩阵。算法如下所示:

```
procedureCalcRelationalMatrix(R)
  #R 是 COO 形式的偏好矩阵
  cudaMemset(R, 1.0, m*n)#把元素值都设置为 1.0
  Copy R from host memory to GPU global memory
  initializecusparse library
  create and setup matrix descriptor
  cusparseXcoo2csr(R)
  #把矩阵从 COO 形式转换为 CSR 形式
  malloc GPU global memory for new matrix B
  cusparseScsr2dense(R, B)
  #把 R 转换为实矩阵 B
  malloc GPU global memory for new matrix C
  #计算 R*B^T,结果在于矩阵 C
  C = cusparseScsrmm(R, B, CUSPARSE_OPERATION_TRANSPOSE)
```

2.2　构建预测向量算法

把用户偏好矩阵 R 的每一行作为一个偏好向量 V,它表示的是用户对所有项目的偏好程度。偏好向量的维度刚好就是项目的数量,每一个项目就是向量的一个维度,维度的值就是用户对项目的偏好值。用户偏好矩阵是本文算法的一个输入,当需要为某一个用户推荐项目的时候,只要从偏好矩阵中取出相对应的行作为此用户的偏好向量。偏好向量的形式如式(3)所示。

$$V=[V_1,V_2,\cdots,V_n] \tag{3}$$

为用户 i 推荐项目的时候,使用此用户的偏好向量 V 与用户的偏好矩阵 C 相乘,得到此用户的预测向量 P。计算方法如式(4)所示。

$$P=C\times V=\begin{bmatrix} C_{1,1} & \cdots & C_{1,n} \\ \vdots & & \vdots \\ C_{n,1} & \cdots & C_{n,n} \end{bmatrix}\times\begin{bmatrix} V_1 \\ \vdots \\ V_n \end{bmatrix}=\begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix} \tag{4}$$

预测向量的维度与项目的数量相等,每个维度代表一个项目。维度的值越大,表示它对应的项目越值得推荐给这个用户。

为了计算所有用户的预测向量,可以先把用户偏好矩阵 R

进行倒置,得到倒置的偏好矩阵 R^T,R^T 的每个列向量就是一个用户的偏好向量。接下来,只要把相关性矩阵 C 乘以用户偏好矩阵 R,得到矩阵 P。矩阵 P 的每一个列向量就是用户的预测向量。在并行计算上,可以使用一个 GPU 线程来计算矩阵 P 的一个元素,即矩阵 C 中的一行乘以矩阵 R 中的一列。

2.3 推荐结果向量构建算法

对于上一个步骤计算出来的用户预测向量,要使用 top-N 算法,获得最大的前 N 个元素对应的项目,作为用户推荐项目集。

在 GPU 中,每个线程负责一个用户的推荐计算。GPU 线程从偏好矩阵 R 和预测矩阵 P 中分别读取一个列向量,即一个用户对所有项目的偏好向量和预测向量,这两个向量长度一样,元素一一对应。然后对预测向量进行如下操作:对于预测向量的每个元素 e,如果偏好向量的对应元素的值不为 0,则把元素 e 的值设为 0,否则不改变。然后,遍历修改后的预测向量的 n 个元素,遍历时使用大小为 N 的小根堆保留 N 个当前最大的元素,当遍历结束时,堆中元素对应的项目就是所求。具体算法如下所示:

```
procedure Get top-N(predict_vector)
  for i:=0 to m-1 do#对于 m 个项目中的每一个
    if item(i) is rated by the user then#已经被用户评分过
      continue#则忽略它
    end if
    if minHeep.size<k#推荐数量未达到指定个数 k
    or
    #或者,此项目更加值得推荐
      minHeep.root.value< predict_vector[i]
    then#则把此项目加入到推荐堆栈中
      element.index:=i
      element.value:=predict_vector[i]
      minHeep.setRoot(element)
      minHeep.heeptify()
     end if
  end for
  return minHeep#返回获得的 k 个推荐项目
```

3 实验结果和分析

在本章中设计实验从两个方面对算法进行比较验证。首先,从准确率方面比较基于 GPU 的并行协同过滤算法与传统的基于项目的协同过滤算法,验证算法具有可靠的准确率。然后,从运行效率方面比较基于 GPU 的并行算法与其 CPU 版本,验证算法在性能上得到的极大提高。

实验所使用的 CPU 是 Intel Core 2 E7500,频率是 2.9 GHz,二级缓存是 3 MB,外加 2 GB 内存;所使用的 GPU 是 NvidiaGeforceGT 220,核心频率是 475 MHz,显存是 512 MB,SP 数量是 48。在实验中,算法使用 C++ 语言实现,使用 Windows7 32 位操作系统。开发环境是 Microsoft Visual Studio 2008,使用了 CUDA3.0。

3.1 算法评估指标

本文将使用一个被广泛运用的评价方法,称为 leave-one-out 方法[8]。它针对于每个用户会去掉一个评分项目,然后利用剩下的评分项目对这个去年的项目作预测评分。之后比较这个预测评分和用户的实际评分之间的差别来评分推荐质量。本文将这个方法作了修改。对于去掉了某个项目的用户 A,如果最终为他推荐的结果集中包含了这个项目,就认为命中(hit)了。然后,计算召回率(recall),方法如式(5)所示。

$$\text{recall} = \frac{\text{number of hits}}{N} \tag{5}$$

在式(5)中,N 表示本文评估用的所有用户—项目评分对的数量。因为本文为每一个用户选择一个评分对进行评估,所以 N 等于测试集中用户的数量。在为用户选择评估用户的评分对时,本文选用用户给出最高分的评分对。

下面比较算法的 CPU 实现和 GPU 实现的运行效率。计算加速比的方法如式(6)。

$$\text{Speedup} = \frac{\text{TimeCPU}}{\text{TimeGPU}} \tag{6}$$

在公式中,TimeCPU 表示算法在 CPU 上的平均运行时间,TimeGPU 表示算法在 GPU 上的运行时间。

3.2 性能对比

本文使用数据集 A 和 B 来进行性能测试,前者数据量较小,后者较大。分别在 CPU 上和 GPU 上实现算法的运算过程。然后多次在 CPU 和 GPU 上进行,最后取平均值。将这个平均值作为本文的实验结果。在算法运算过程的计时上,忽略了数据从文件中读取的时间。

在算法构建相关性矩阵时,加速比情况如表 1 所示。

表 1 构建相关性矩阵时的加速比

比较项	方法	数据集 A	数据集 B
GPU/s	本文方法	0.93	11.16
	文献[5]方法	1.31	20.42
CPU/s	本文方法	25.8	298.6
	文献[5]方法	31.5	435.7
Speedup	本文方法	27.74	26.76
	文献[5]方法	24.05	21.34

在算法构建预测矩阵时,加速比情况如表 2 所示。

表 2 构建预测矩阵时的加速比

比较项	方法	数据集 A	数据集 B
GPU/s	本文方法	0.14	3.36
	文献[5]方法	0.23	4.04
CPU/s	本文方法	25.4	655.3
	文献[5]方法	29.5	720.8
Speedup	本文方法	181.43	195.03
	文献[5]方法	128.26	178.42

在算法构建推荐结果矩阵时,加速比情况如表 3 所示。

表 3 构建推荐结果矩阵时的加速比

比较项	方法	数据集 A	数据集 B
GPU/s	本文方法	0.01	0.03
	文献[5]方法	0.02	0.35
CPU/s	本文方法	0.02	0.58
	文献[5]方法	0.03	0.66
Speedup	本文方法	2	1.93
	文献[5]方法	1.5	1.89

整体算法的加速比情况如表 4 所示。

表 4 整体算法的加速比与准确率

比较项	方法	数据集 A	数据集 B
GPU/s	本文方法	1.08	14.82
	文献[5]方法	1.56	24.81
CPU/s	本文方法	51.22	954.48
	文献[5]方法	61.03	1157.16
Speedup	本文方法	47.43	64.4
	文献[5]方法	39.12	46.64
召回率	本文方法	8.1	8.4
	文献[5]方法	7.6	8.05

从实验结果可以看出,GPU 版本的算法在性能上明显高于 CPU 版本,整体算法的加速比达到了 50 倍左右。可以看到,在构建预测矩阵的时候,算法的效率得到了最大的提高。而在最后构建推荐结果矩阵时,加速比却明显偏低,只有 2 倍左右,这是因为该步骤里面使用了复杂的 Top-*N* 算法,而 GPU 的处理单元是不擅长于复杂的逻辑运算。但因为该步骤所使用的时间在整个算法的运算时间中所占比例比较小,所以对整个算法效率的提高影响不大。

从召回率的实验结果(表 4)可以看出,无论是在计算时间还是在准确率方面,本文的算法相比 Puntheeranurak 等人[8]在 2011 年提出的方法有明显的优势。因为在数据集 *A* 和 *B* 中,某些用户对电影的评价总是偏向于给较低的分数而另一些用户则偏向于给较高的分数。本文设计的项目相关性矩阵、用户偏好向量、预测向量和推荐结果向量有利于得出这一特征信息。再者,这些矩阵与向量的代数结构比较容易实现于 CUDA 框架中,所以,本文提出的并行协同过滤算法在推荐准确率和处理时间上都有较优的性能。

4　结束语

为了解决协同过滤算法的介绍性问题,本文设计了一套基于 CUDA 的协同过滤算法。首先,定义了算法问题和参数意义。其次,提出了算法的设计思想,本文提出的算法主要是在基于项目的协同过滤算法的基础上进行设计的。然后,提出了算法的详细实现步骤,主要分为构建相关性矩阵、构建用户偏好向量、构建预测向量、构建推荐结果向量,共四个步骤。接着,本文介绍了如何使用 CUDA 实现这个并行算法。最后,本文从准确率和性能两个方面对这个算法进行测试,测试结果显示该算法有着较高的加速比。

参考文献:

[1] ADOMAVICIUS G, TUZHILIN A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions[J]. IEEE Trans on Knowledge and Data Engineering, 2005, 17(6): 734-749.

[2] 陈健,印鉴. 基于影响集的协作过滤推荐算法[J]. 软件学报, 2007, 18(7): 185-194.

[3] MELVILLE P, MOONEY R J, NAGARAJAN R. Content-boosted collaborative filtering for improved recommendations[C]//Proc of the 19th National Conference on Artificial Intelligence. Menlo Park: American Association for Artificial Intelligence, 2002: 187-192.

[4] CGEORGE T, MERUGU S. A scalable collaborative filtering framework based on co-clustering[C]//Proc of the 5th International Conference on Data Mining. Washington DC: IEEE Computer Society, 2005: 625-628.

[5] 吕成戍,王维国,丁永健. 基于 KNN-SVM 的混合协同过滤推荐算法[J]. 计算机应用研究, 2012, 29(5): 1707-1709.

[6] 吴月萍,杜奕. 基于人工鱼群算法的协同过滤推荐算法[J]. 计算机工程与设计, 2012, 33(5): 1852-1856.

[7] LI Rui-feng, ZHANG Yin, YU Hai-han. A social network-aware top-*N* recommender system using GPU[C]//Proc of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries. 2011: 287-296.

[8] PUNTHEERANURAK S, CHAIWTOOANUKOOL T. An item-based collaborative filtering method using item-based hybrid similarity[C]//Proc of the 2nd IEEE International Conference on Software Engineering and Service Science. 2011: 469-472.

[9] SONG Guo-hui, SUN Shu-tao, FAN W. Applying user interest on item-based recommender system[C]//Proc of International Conference on Computational Sciences and Optimization. 2012: 635-638.

[10] ZHENG Si-ting, HONG Wen-xing, ZHANG Ning, *et al*. Job recommender systems: a survey[C]//Proc of International Conference on Computer Science & Education. 2012: 920-924.

(上接第 2655 页)

[2] 胡祥培,丁秋雷,张漪,等. 干扰管理研究评述[J]. 管理科学, 2007, 20(2): 2-8.

[3] 陈庭贵,琚春华. 多干扰的资源约束项目调度问题[J]. 计算机集成制造系统, 2012, 18(11): 2409-2418.

[4] 刘志霞. 资源受限项目调度问题及其任务扰动的干扰管理研究[D]. 沈阳:沈阳工业大学, 2011.

[5] 陈卫明. 动态环境下产品开发项目调度问题及其求解研究[D]. 武汉:华中科技大学, 2011.

[6] LAMBRECHTS O, DEMEULEMEESTER E, HERROELEN W. Time slack-based techniques for robust project scheduling subject to resource uncertainty[J]. Annals of Operations Research, 2011, 186(1): 443-464.

[7] KLIMEK M, LEBKOWSKI P. Resource allocation for robust project scheduling[J]. Bulletin of the Polish Academy of Sciences-Technical Sciences, 2011, 59(1): 51-55.

[8] HAZIR O, HAOUARI M, EREL E. Robust scheduling and robustness measures for the discrete time/cost trade-off problem[J]. European Journal of Operational Research, 2010, 207(2): 633-643.

[9] ZHU Z, BARD J F, YU G. Disruption management for resource-constrained project scheduling[J]. Journal of the Operational Research Society, 2005, 56(4): 365-381.

[10] AL-FAWZAN M A, HAOUARI M. A bi-objective model for robust resource-constrained project scheduling[J]. International Journal of Production Economics, 2005, 96(2): 175-187.

[11] KOLISCH R. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation[J]. European Journal of Operational Research, 1996, 90(2): 320-333.

[12] ZHANG Hong, LI Xiao-dong, LI Heng, *et al*. Particle swarm optimization-based schemes for resource-constrained project scheduling[J]. Automation in Construction, 2005, 14(3): 393-404.

[13] ZHANG Hong, LI Heng, TAM C M. Particle swarm optimization for resource-constrained project scheduling[J]. International Journal of Project Management, 2006, 24(1): 83-92.

[14] CHEN R M, WU C L, WANG C M, *et al*. Using novel particle swarm optimization scheme to solve resource-constrained scheduling problem in PSPLIB[J]. Expert Systems with Applications, 2010, 37(3): 1899-1910.

[15] ZHANG Hong, XING Feng. Fuzzy-multi-objective particle swarm optimization for time-cost-quality tradeoff in construction[J]. Automation in Construction, 2010, 19(8): 1067-1075.

[16] 王东升,曹磊. 混沌、分形及其应用[M]. 合肥:中国科学技术大学出版社, 1995.