第44卷 第10期
2017年10月
计 算 机 科 学
COMPUTER SCIENCE
Vol.44 No.10
Oct.2017

多核/众核平台上推荐算法的实现与性能评估

陈 静 方建滨 唐 滔 杨灿群
(国防科学技术大学计算机学院 长沙 410073)

摘 要 用OpenCL语言标准设计并实现了推荐系统领域的两种经典算法:交替最小二乘法(Alternating Least Squares,ALS)与循环坐标下降法(Cyclic Coordinate Descent,CCD)。将其应用到CPU,GPU,MIC多核与众核平台上,探索了在该平台上影响算法性能的因子:潜在特征维数与线程个数。同时,将OpenCL实现的两种算法与CUDA和OpenMP的实现进行比较,得出了一系列结论。在同等条件下,与ALS算法相比,CCD算法的精度更高,收敛速度更快且更稳定,但所耗时间更长。ALS和CCD算法基于OpenCL的实现性能不亚于CUDA(CCD上加速比为1.03x,ALS上加速比为1.2x)和OpenMP的实现(CCD与ALS上加速比大约为1.6~1.7x),并且两种算法在CPU平台上的性能均比GPU与MIC好。

关键词 推荐系统,OpenCL,ALS,CCD
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.10.013

Implementation and Performance Evaluation of Recommender Algorithms Based on Multi-/Many-core Platforms

CHEN Jing FANG Jian-bin TANG Tao YANG Can-qun
(College of Computer,National University of Defense Technology,Changsha 410073,China)

Abstract In this paper,we designed and implemented two typical recommender algorithms,alternating least squares and cyclic coordinate descent in openCL. Then we evaluated them on Intel CPUs,NVIDIA GPUs and Intel MIC,and investigated the performance impacting factors:potential feature dimension and the number of thread. Meanwhile,we compared the OpenCL implementation with that of CUDA and OpenMP. Our experimental results show that in the same condition,CCD converges faster and performs more steadily,but is more time-consuming than ALS. We also observed that the performance based on OpenCL is better than CUDA and OpenMP when running on the same platform:the training time on GPU is slightly faster than that of the CUDA implementation (1.03x for CCD and 1.2x for ALS),and the training time on CPU is 1.6~1.7 times less than that of the OpenMP implementation with 16 threads. When running the OpenCL implementation on different platforms,we noticed that CPU performs better than both the GPU and the MIC.

Keywords Recommender system,OpenCL,ALS,CCD

1 引言

随着信息高科技技术和互联网的发展,人们逐渐从信息匮乏的时代走入了信息过载的时代。推荐系统,是信息过载时代的产物。推荐系统中最核心的部分是推荐算法,其实质是将用户与产品按照一定的方法联系起来。本文关注的两种推荐算法(ALS和CCD)属于协同过滤推荐算法。近年来,许多学者尝试着将多核/众核平台引入到该领域,例如Yu等人[5]通过提出新的CCD++算法实现了对CCD算法性能的优化;Zhou等人[4]提出了新的ALS-WR算法,相比于ALS,该算法获得了5.91%的性能提升。然而,这些相关工作局限于单一算法或者单一平台,多算法多平台系统的性能评估尚为空白。

本文基于OpenCL编程模型实现了两种典型的协同推荐算法:ALS与CCD。因为OpenCL的跨平台特性,该实现能够编译并运行在多种多核与众核平台上(CPU,GPU,MIC)。本文对推荐算法与处理平台展开了多角度的对比分析:比较了两种算法在相同情况下的精度与耗时,比较了同一种算法在不同处理平台上的性能表现,还分析了影响推荐算法性能的因子:潜在特征维数和线程维度。本文的评测结果一方面为在多核与众核平台上优化推荐算法指明了方向,另一方面将有助于用户根据自身需求选择合适的处理平台与推荐算法。

到稿日期:2016-12-21 返修日期:2017-01-11 本文受国家自然科学基金项目(61170049,61402488,61502514,61602501),国家863项目(2015AA01A301)资助。
陈 静(1995-),女,硕士生,主要研究方向为高性能计算、异构计算,E-mail:jingchen95@yeah.net;**方建滨**(1984-),男,助理研究员,主要研究方向为高性能计算、异构计算,E-mail:j.fang@nudt.edu.cn;**唐 滔**(1984-),男,助理研究员,主要研究方向为高性能计算、异构计算;**杨灿群**(1971-),男,研究员,主要研究方向为高性能计算、异构计算。

因此,本文的贡献如下:

(1)基于 OpenCL 并行编程模型实现了推荐系统领域内的两种经典算法,即 ALS 与 CCD。

(2)在主流多核与众核平台(CPU,GPU,MIC)上对比了 ALS 与 CCD 算法的精度与耗时。

(3)评估了两种推荐算法在多种处理平台上的性能表现,并分析了影响性能的因素。

2 推荐算法:ALS 与 CCD

2.1 交替最小二乘法

ALS 旨在最小化式(1)所示的平方误差损失函数,该损失函数加入了正则化项来避免过拟合等问题。

$$L(X,Y)=\sum_{u,i}(r_{ui}-x_u^{\mathrm{T}}y_i)^2+\lambda(|x_u|^2+|y_i|^2) \tag{1}$$

其中,r_{ui} 表示用户 u 对产品 i 的评分,向量内积 $x_u^{\mathrm{T}}y_i$ 是用户 u 对商品 i 评分的近似值,λ 是正则化项的系数。

对于式(1),固定 Y,将 $L(X,Y)$ 对 x_u 求偏导,令导数为 0,便得到式(2)。

$$x_u=(Y^{\mathrm{T}}Y+\lambda I)^{-1}Y^{\mathrm{T}}r_u \tag{2}$$

同理固定 X,可得式(3)。

$$y_i=(X^{\mathrm{T}}X+\lambda I)^{-1}X^{\mathrm{T}}r_i \tag{3}$$

ALS 算法的伪代码如算法 1 所示。

算法 1 ALS 算法

```
Input:R,k,λ
Output:X,Y
1. function ALS
2.    set X to zero matrix,Y to random initial guess
3.    while ite:1→iteration times
4.       for rows u:1→m
5.          Calculating x_u using x_u=(Y^T Y+λI)^-1 Y^T r_u
6.       end for
7.       for cols j:1→ n
8.          Calculating y_i using y_i=(X^T X+λI)^-1 X^T r_i
9.       end for
10.  end while
11. end function
```

2.2 循环坐标下降法

假定 $\bar{x}_t$ 代表的是用户矩阵 X 的一列,相应的 $\bar{y}_t$ 代表物品矩阵 Y 的一列,u^* 和 v^* 分别是由 $\bar{x}_t$ 和 $\bar{y}_t$ 映射的向量,那么最小化函数如式(4)所示:

$$L(X,Y)=\sum_{(i,j)\in\Omega}(R_{ij}+\bar{x}_{ti}\bar{y}_{tj}-u_iv_j)^2+\lambda(\|u\|^2+\|v\|^2) \tag{4}$$

使用式(5)对式(4)进行简化,从而得到式(6):

$$\hat{R}_{ij}=R_{ij}+\bar{x}_{ti}\bar{y}_{tj},\forall(i,j)\in\Omega \tag{5}$$

$$L(X,Y)=\sum_{(i,j)\in\Omega}(\hat{R}_{ij}-u_iv_j)^2+\lambda(\|u\|^2+\|v\|^2) \tag{6}$$

将式(6)对 u 求偏导,得到 u_i;对 v 求偏导,得到 v_j。

$$u_i=\frac{\sum_{j\in\Omega_i}\hat{R}_{ij}v_j}{\lambda+\sum_{j\in\Omega_i}v_j^2},i=1,\cdots,m \tag{7}$$

$$v_j=\frac{\sum_{i\in\Omega_j}\hat{R}_{ij}u_i}{\lambda+\sum_{i\in\Omega_j}u_i^2},j=1,\cdots,n \tag{8}$$

更新 $\bar{x}_t$ 与 $\bar{y}_t$:

$$u^*\Rightarrow\bar{x}_t,v^*\Rightarrow\bar{y}_t \tag{9}$$

更新 R_{ij},并进入下一次迭代:

$$R_{ij}=\hat{R}_{ij}-u_i^*v_j^*,\forall(i,j)\in\Omega \tag{10}$$

CCD 算法的伪代码如算法 2 所示。

算法 2 CCD 算法

```
Input:R,k,λ
Output:X,Y
1. function CCD
2.    Set X to random initial guess,Y to zero matrix
3.    while ite:1→iteration times
4.       for t:0→k
5.          Calculating R̂_ij
6.          for in_ite:1→inner iteration times
7.             Calculating u_i using (8)
8.             Calculating v_j using (9)
9.          end for
10.         updating x̄_t,ȳ_t using (10)
11.         updating R using (11)
12.      end for
13.  end while
14. end function
```

3 基于 OpenCL 的 ALS 与 CCD 算法的设计与实现

3.1 总体框架

图 1 给出了基于 OpenCL 的两种推荐算法的设计与实现的总体框架。该框架包括两个部分:kernel 设备端和主机 host 端。其中,算法的主要实现于 Device 端的 kernel 函数,Host 端主要实现固定代码部分以及调用 kernel、设备主机间传送数据、申请 buffer、释放 buffer 等。

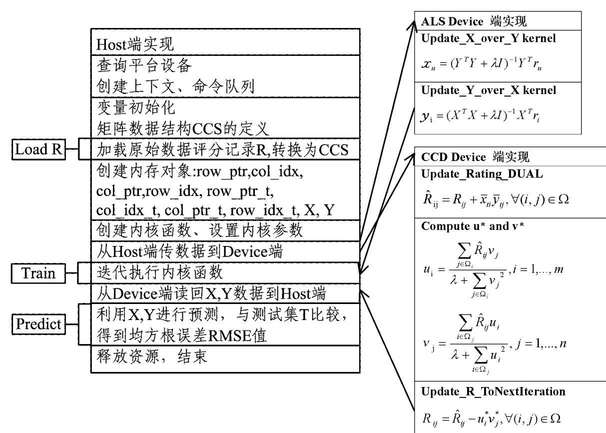

图 1 基于 OpenCL 的算法框架设计

3.2 kernel 端的设计与实现

3.2.1 ALS 算法

交替最小二乘法在 kernel 端共实现了两个内核函数:update_X_over Y,update_Y_over_X。由于两个 kernel 函数类似,本文只阐述 update_X_over_Y 函数,其伪代码如算法 3 所示。

算法 3　update_X_over_Y 函数

```
Input:X,Y
Output:X
1. kernel function update_X_over_Y
2.   for Rw ←1,rows
3.        * Xu ←&X[Rw * k];
4.     if omegaSize>0
5.          subMatrix ← Multiply transpose matrix by matrix
6.          subMatrix ← subMatrix+ lambda
7.        Inverse subMatrix using Cholesky Method
8.        for c ← 0,k
9.          for idx ←row_ptr[Rw],row_ptr[Rw+1]
10.            idx2 ← colMajored_sparse_idx[idx];
11.            subVector[baseV+c] +=val[idx2] ×Y[(col_idx
               [idx] * k)+c];
12.          end for
13.        end for
14.        for c ← 0,k
15.        Xu[c] +=subVector * subMatrix
16.        end for
17.     end if
18.   end for
19. end function
```

算法 3 中的伪代码说明了 update_X_over_Y 内核函数具体的实现功能与 ALS 算法之间的对应关系。该函数的实现将式(2)分成两部分来求解。其中第一部分$(Y^{T}Y+\lambda I)^{-1}$的实现对应伪代码中第 5－7 行，第二部分$Y^{T}r_{u}$的实现对应第 8－13 行。

3.2.2　CCD 算法

循环坐标下降法在 kernel 端共定义了 3 个内核函数：

(1)Compute_u* _and_v* ;

(2)Update_Rating_DUAL_kernel;

(3)Update_R_ToNextIteration。

Compute_u* _and_v* 函数实现了算法中式(7)、式(8)表示的u^{*}和v^{*}。函数实现的伪代码如算法 4 所示。其中，第 2－4行实现了v^{*}的计算，第 5－7 行实现了u^{*}的计算。

算法 4　Compute_u* _and_v* 函数

```
Input:上一次迭代得到的 u,v
Output:本次更新后的 u,v
1. kernel function Compute_u* _and_v*
2.     for c:1,cols
3.     v[c]← Compute(col_ptr,row_idx,val,c,u,lambda * (col_ptr
       [c+1]-col_ptr[c]),v[c]);
4.     end for
5.     for c:1,cols_t
6.       u[c]←Compute(col_ptr_t,row_idx_t,val_t,c,v,lambda *
         (col_ptr_t[c+1]-col_ptr_t[c]),u[c]);
7.     end for
8. end kernel
```

Update_Rating_DUAL_kernel 函数实现了式(5)对最小化函数的简化。算法 5 为该函数实现的伪代码。代码第 5 行阐述了通过每次迭代计算所得的 X,Y 矩阵值，根据式(1)，对应元素相乘并累加更新 $R.value$。

算法 5　Update_Rating_DUAL_kernel 函数

```
Input:X,Y
Output:R. value
1. kernel function Update_Rating_DUAL_kernel
2.     for i:1,cols
3.         yi←Y[i]
4.         for idx:col_ptr[i],col_ptr[i+1]
5.           value [idx] +=X[row_idx[idx]] * yi;
6.         end for
7.     end for
8. end kernel
```

Update_R_ToNextIteration 函数实现了式(10)对评分矩阵 R 的更新，继而进入下一次的迭代循环。该函数的实现与 Update_Rating_DUAL_kernel 函数的实现类似，将函数内部的加法(算法 5 第 5 行)改为减法，不再赘述。

3.3　Host 端的设计与实现

ALS 算法 Host 端首先实现了查询平台、设备，然后创建上下文、命令队列、程序，最后编译程序。其中查询 platform 包括 3 种平台，即 CPU，GPU 和 MZC。

第 1 步　进行数据的初始化与数据结构的定义。数据结构采用 CCS 形式，定义用户矩阵 $X(m\times k)$并将其初始化为零矩阵，定义产品矩阵 $Y(n\times k)$并进行随机初始化。

第 2 步　申请设备端 Buffer，创建内核函数与设置内核参数。

第 3 步　数据传递与执行内核。由主机端传入设备端的数据包括 row_ptr，col_idx，col_ptr，row_idx，val，X，Y 等。循环迭代次数为 5，线程总数为 512，线程块数为 16，块内线程个数为 32。执行完毕后，将数据传回主机端，数据包括两部分：X 与 Y。

第 4 步　进行预测与精准度测试。使用式(1)得到预测结果，根据 RMSE 的计算公式得到预测精准度[4-5]。

第 5 步　释放资源。

4　结果与讨论

4.1　测试数据集与实验平台

本文使用的数据集为 Netflix Dataset，数据格式为 user id | item id | rating。其中，toy-example 数据集包含 6040 个用户和 3952 部电影，共 1000209 条评分记录，其中训练集共 900189 条评分记录，测试集共 100020 条评分记录；toy-example1 数据集包含 480189 个用户和 17770 个产品，共 100480507 条评分记录，训练集共 99935418 条评分记录，测试集共 545089 条评分记录。

本文测试平台包括 Intel Xeon CPU E5-2670，NVIDIA Tesla GPU K20c 与 Intel Xeon Phi (MIC)。宿主机 CPU 采用 Redhat Linux v7.0 (3.10.0-123.el7.x86_64)，协处理器 Phi 采用定制版 uOS (v2.6.38.8)，将 Intel MPSS(v3.6)用作驱动与 CPU 进行通信。

4.2　实验结果

本文评估的推荐算法为 ALS 算法和 CCD 算法，并且分别在 CPU，GPU，MIC 不同平台上进行了运行时间和准确性两方面的测试，同时将 OpenCL 的运行结果与 CUDA 和 OpenMP 版本进行了比较。表 1、表 2 分别列出两种算法在不同平台上的运行情况，表中的数据分别为多次(至少 3 次)运行结果的平均值。

表 1 CCD 算法在 toy-example1 数据集上的运行情况

Platform	Load R	Training	Predict	RMSE
OpenMP	75.46	32.4095	0.4256	0.9378
CUDA	78.74	117.073	0.4141	0.9378
OpenCL(GPU)	154.38	113.857	0.4133	0.9378
OpenCL(CPU)	155.04	20.0965	0.5565	0.9378
OpenCL(MIC)	153.02	140.386	0.4099	0.9378

表 2 ALS 算法在 toy-example1 数据集上的运行情况

Time	Load R	Training	Predict	RMSE
OpenMP	78.6254	29.7402	0.3256	1.0180
CUDA	75.4679	120.871	0.4078	1.0180
OpenCL(GPU)	152.672	107.856	0.3724	1.0180
OpenCL(CPU)	152.728	17.3752	0.3218	1.0180
OpenCL(MIC)	152.242	189.029	0.3707	1.0180

其中,测试运行时间包括了 3 部分:Load R 指的是加载评分数据集 R 将其转化成 CCS 数据结构所耗费的时间;Training 指的是调用内核函数并执行,以及将 X 和 Y 矩阵数据在主机与设备端进行传递所耗费的时间;Predict 指的是使用 Training 结果进行预测,并计算得到均方根误差 RMSE 所耗费的时间。

相比于 CUDA 与 OpenMP 的实现,基于 OpenCL 实现的 ALS 与 CCD 算法具有一定的优势。CCD 算法的 OpenCL(CPU)相比 OpenMP 的实现的加速比约为 1.61 倍;基于 OpenCL 的实现在 GPU 平台上与 CUDA 实现时间相当。ALS 算法基于 OpenCL(CPU)相比 OpenMP 的实现加速比大约是 1.71 倍;基于 OpenCL (GPU)相比 CUDA 实现加速比大约是 1.2 倍。通过比较不同的平台可以发现,基于 OpenCL 实现的 CCD 和 ALS 算法,在 CPU 平台上的运行速度最快,GPU 次之,MIC 最慢。特别地,ALS 在 CPU 平台上相比 GPU 加速比达到 5.67,与 MIC 相比加速比为 6.97。

4.2.1 算法精度对比

在同样的条件下,CCD 算法比 ALS 算法的预测精度高,但所耗时间更长。进一步测试发现,算法的精度在一定范围内随着迭代次数的增加而提高。图 2 是以 MIC 平台为例,对于 Netflix 数据集,ALS 与 CCD 算法迭代次数为 1~25 时,算法精度 RMSE 的对比。CCD 算法的总体 RMSE 均低于 ALS,并且 CCD 的变化趋势比 ALS 更平稳。这充分说明了在数据集处理上,CCD 算法比 ALS 算法的收敛速度更快且更稳定。

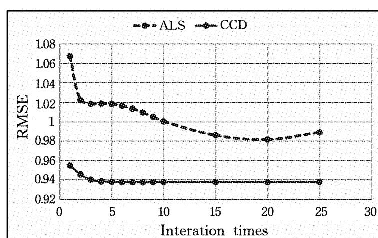

图 2 算法随着迭代次数的增加 RMSE 的变化

4.2.2 线程维度的影响

以 Netflix 数据集中的 toy-example1 为例,探索了在 3 种平台上,当线程个数不同时,ALS 和 CCD 算法的运行情况。设定线程块的变化范围是 256~8192,块内线程个数的变化范围是 32~512。得到如下结果:对于 CCD 算法,GPU 平台上在线程数为 8192×256 时取得最优性能,CPU 平台上在线程数为 8192×64 时取得最优性能,MIC 平台上在线程数为 2048×32 时取得最优性能;对于 ALS 算法,GPU 平台上在线程数为 4096×64 时取得最优性能,CPU 平台上在线程数为 4096×32 时取得最优性能,MIC 平台上在 1024×32 时取得最优性能。相比于 ALS 算法,CCD 算法在 3 种平台上获得最佳性能使用的线程数(包括线程快的个数与块内线程个数)更多,需要并行化的程度更大。不同算法在不同平台上取得最大性能时的线程维度不同,需要搜索空间来找寻最优的线程维度。

4.2.3 潜在特征维数的影响

经测试,潜在特征维数 k 对算法性能的影响较大。以 toy-example1 数据集为例,当 k 变化范围为 1~20 时,得到如下结果:ALS 在 $k=10$ 时精度最高,而 CCD 在 $k=15$ 时精度最高。在 toy-example 数据集中,ALS 在 $k=5$ 时精度最高,CCD 在 $k=10$ 时精度最高。因此,相比于 ALS,CCD 需要更多的特征维数 k 才能获得更好的准确度。

结束语 本文使用 OpenCL 编程模型设计并实现了两种经典的协同过滤推荐算法:ALS 算法与 CCD 算法,并将它们在多种平台上进行了性能评测与对比分析。测试结果发现,CCD 算法的精度比 ALS 算法的精度更高,但运行时间也更长。两种算法的 OpenCL 实现性能不亚于 CUDA(CCD 上加速比 1.03x,ALS 上加速比 1.2x)和 OpenMP(CCD 与 ALS 上加速比 1.6~1.7x)的实现。此外,两种算法在 CPU 平台上的性能均优于 GPU 和 MIC。与此同时,本文还分析了影响算法性能的两种因素:线程个数与潜在特征维数 k。下一步将重点考虑本文实现在多核与众核体系结构上的性能优化。

参 考 文 献

[1] RODRIGUES A V,JORGE A,DUTRA I. Accelerating Recommender Systems using GPUs[C]// ACM Symposium on Applied Computing. ACM,2015:879-884.

[2] GATES M,ANZT H,KURZAK J,et al. Accelerating Collaborative Filtering Using Concepts from High Performance Computing[C]//2015 IEEE International Conference on Big Data (Big Data). IEEE,2015:667-676.

[3] PATEREK A. Improving regularized singular value decomposition for collaborative filtering[C]// ACM International Conference on Knowledge Discovery and Data Mining. 2007:39-42.

[4] ZHOU Y H,WILKINSON D,SCHREIBER R,et al. Large-scale Parallel Collaborative Filtering for the Netflix Prize[C]// Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management. 2008:337-348.

[5] YU H F,HSIEH C J,SI S,et al. Scalable Coordinate Descent Approaches to Parallel Matrix Factorization for Recommender Systems[C]//2013 IEEE 13th International Conference on Data Mining(2012). 2012:765-774.

[6] KOREN Y,BELL R,VOLINSKY C. Matrix Factorization Techniques for Recommender Systems[J]. Computer,2009,42(8):30-37.

[7] ZHUANG Y,CHIN W S,JUAN W C,et al. A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems[C]// Proceedings of ACM Recommender Systems 2013. 2013:249-256.