基于GPU的并行奇异值分解最小平方估计算法

李 繁[1,2] 金明录[1] 刘 继[3]
(大连理工大学电子信息与电气工程学部 大连 116024)[1]
(新疆财经大学网络与实验教学中心 乌鲁木齐 830012)[2]
(新疆财经大学统计与信息学院 乌鲁木齐 830012)[3]

摘 要 对奇异值分解(SVD)求解最小平方估计的问题进行了研究。提出迭代式分割与合并的算法(IDMSVD),目的是解决奇异值分解在估计参数时非常耗费内存空间的问题。基于IDMSVD提出了并行IDMSVD算法,并使用GPU实现之。实验结果显示,IDMSVD可以有效地解决SVD求最小平方解耗费运行时间与内存空间的问题,并行IDMSVD算法可进一步改善IDMSVD的运行时间。

关键词 奇异值分解,最小平方估计,并行处理
中图法分类号 TP393 **文献标识码** A

GPU-based Parallel SVD Least Squares Estimation Algorithm

LI Fan[1,2] JIN Ming-lu[1] LIU Ji[3]
(Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology, Dalian 116024, China)[1]
(Network & Experimental Teaching Center, Xinjiang University of Finance and Economics, Urumqi 830012, China)[2]
(School of Statistics and Information, Xinjiang University of Finance and Economics, Urumqi 830012, China)[3]

Abstract Singular value decomposition (SVD) for solving least-squares estimation problem was studied. The proposed iterative divide and merge algorithm (IDMSVD) which aims to improve the singular value decomposition in the estimation of parameters is very time-and memory space-consuming. Based on IDMSVD, a parallel IDMSVD algorithm was proposed and realized using the Nvidia GPUs. The experimental results show that IDMSVD can effectively improve the minimum run time and memory space consuming problems in the SVD squares solution, and the prallel IDMSVD algorithm can further improve IDMSVD operation time.

Keywords SVD, Least squares estimation, Parallel processing

1 引言

线性回归分析是用来探讨自变量和因变量之间关系的模型,可以分为简单回归分析和复回归分析,简单回归分析是探讨一个自变量和一个因变量之间的关系,复回归分析是简单回归分析的延伸,用来探讨多个自变量和一个因变量之间的关系。实际应用中,必须推论模型中的参数(回归系数),而通常所产生的线性方程式模型,可能不存在一个正确解。在这种情况下,必须求最接近于正确解的近似值,评估近似值的好坏最常使用的方法之一为最小平方法。最小平方法就是将求得的解代入模型中使其误差平方和最小,这称为最小平方问题。

目前,已经提出许多方法用于解决最小平方问题[1-5]。Björck和Yuan提出3个算法,利用LU分解求行向量为线性独立的矩阵。Cholesky分解方法大约比LU分解的速度快2倍。Foster提出使用序列的QR分解的算法,而QR分解可以提供比Cholesky分解方法更精确的近似值,但必须花费较多的计算时间。如果矩阵的秩(rank)不足,最小平方问题则会有无穷多组解。这种情况下,仍然可以使用奇异值分解(SVD)来求最小平方问题。Giraud等人提出的一项技术也是利用奇异值分解,他们声称求出来的近似值非常接近正确解。Lee和Ouyang提出递归式奇异值分解最小平方估计法(RSVD),在每一次迭代时,只加入一笔训练样本做奇异值分解,以解决最小平方问题。

对于较大型的数据集,在求最小平方问题时,如果直接利用奇异值分解则会耗费非常多的运算时间和内存空间。有许多人提出并行算法来解决这个问题,但是,他们是从如何加快奇异值分解[6-10]计算的角度着手。例如:Konda和Nakamura提出对于双对角矩阵并行运算的算法,以减少奇异值分解的运行时间。

本文提出的迭代式分割与合并的奇异值分解最小平方估计法(iterative divide and merge SVD-based least squares estimator, IDMSVD-LSE),可以有效地降低对于大型数据利用奇异值分解求最小平方问题时的运算时间和内存空间,但不同于上述那些方法,IDMSVD-LSE算法主要分为3个步骤:首先将输入数据分成许多个数据区块;然后利用奇异值分解对

到稿日期:2013-07-04 返修日期:2013-11-17 本文受国家自然科学基金(71261025)资助。
李 繁(1974—),男,博士生,讲师,主要研究方向为高性能计算、云计算,E-mail:lifanxj@163.com;金明录(1958—),男,教授,博士生导师,主要研究方向为信号处理快速算法、高性能计算等;刘 继(1974—),男,博士,副教授,主要研究方向为知识管理、并行计算等。

每个数据区块分别做分解；之后合并分解后的结果，作为下一层的输入矩阵。重复上述3个步骤，直到缩减后的数据足够小才停止。而在每一个阶层中，分解与合并数据区块时，彼此之间是互相独立的。

多核心中央处理器（multi-core CPU）和图形处理器（graphic processing unit，GPU）现今已非常普遍，常用来执行可并行运算的工作，属于分布式系统中的紧密耦合处理的系统架构。双核心、四核心和八核心是目前常见的多核心中央处理器，图形处理器的核心数是一般多核心中央处理器的数十倍甚至百倍，例如NVIDIA的GeForce GTX 570图形处理器有480个核心。利用图形处理器来计算原本由中央处理器处理的通用运算，称为通用图形处理器。在面对单指令流多数据流（SIMD）且数据处理的运算量远大于数据传输的需求时，通用图形处理器在性能上的表现远远超过中央处理器。虽然GPU执行每个数值运算比CPU慢，但在处理数据量庞大而简单的运算时，其速度比CPU快数十倍甚至数百倍。另外，GPU的内存带宽也比CPU大很多。因此本文设计了并行算法，以进一步缩减所需的时间。IDMSVD算法的每个数据区块运算是相同且独立的，属于SIMD型态的运算，因此很适合利用GPU执行。

2 最小平方估计法

2.1 最小平方问题

复回归分析是探讨多个自变量和一个因变量之间的关系。假设总共有M笔训练样本为$(x_1,y_1),(x_2,y_2),\cdots,(x_M,y_M)$。$x_i=[x_{i,1},x_{i,2},\cdots,x_{i,k}]$，共$k$个维度。$x$为自变量，$y$为因变量，$x$和$y$都是实数。线性复回归模型可表示为：

$$y=\beta_0+\beta_1x_1+\beta_2x_2+\cdots+\beta_kx_k \tag{1}$$

其中，$\beta_0,\beta_1,\cdots,\beta_k$为回归系数。将$M$笔训练样本代入式(1)中可得到$M$个线性方程式所构成的线性系统：

$$\begin{aligned}y_1&=\beta_0+\beta_1x_{1,1}+\beta_2x_{1,2}+\cdots+\beta_kx_{1,k}\\y_2&=\beta_0+\beta_1x_{2,1}+\beta_2x_{2,2}+\cdots+\beta_kx_{2,k}\\&\vdots\\y_M&=\beta_0+\beta_1x_{M,1}+\beta_2x_{M,2}+\cdots+\beta_kx_{M,k}\end{aligned}$$

此线性系统用矩阵形式可表示为：

$$Y=X\beta \tag{2}$$

$$X=\begin{bmatrix}1&x_{1,1}&\cdots&x_{1,k}\\1&x_{2,1}&\cdots&x_{2,k}\\\vdots&\vdots&\vdots&\vdots\\1&x_{M,1}&\cdots&x_{M,k}\end{bmatrix} \tag{3}$$

$$Y=\begin{bmatrix}y_1\\y_2\\\vdots\\y_M\end{bmatrix},\beta=\begin{bmatrix}\beta_0\\\beta_1\\\vdots\\\beta_k\end{bmatrix} \tag{4}$$

令N等于$k+1$，则X为$M\times N$的矩阵，Y为$M\times1$的矩阵，β为$N\times1$的矩阵。最小平方问题是要估算β，使得$\|Y-X\beta\|$为最小值。其目标函数为：

$$\min_{\beta}\|Y-X\beta\| \tag{5}$$

假设C是一个矩阵，$\|C\|=\sqrt{trace(C^{\mathrm{T}}C)}$。如果$\beta$满足式(5)，则称$\beta$为式(2)的最小平方解。

2.2 奇异值分解最小平方估计法

利用奇异值分解（singular value decomposition，SVD）可求得式(2)的最小平方解β。首先，利用SVD将X分解成3个矩阵，可表示为：

$$X=U\Sigma V^{\mathrm{T}} \tag{6}$$

式中，U是一个$M\times M$的正交（orthogonal）矩阵，Σ是一个$M\times N$的主对角矩阵，V是一个$N\times N$的正交矩阵。Σ矩阵中的主对角线的值依大小排序，$(\Sigma)_{11}\geqslant(\Sigma)_{22}\geqslant\cdots\geqslant(\Sigma)_{rr}>0$，$r\leqslant N$，$r$为$X$矩阵的秩。$\Sigma$矩阵中的$(\Sigma)_{ii}^2$为$X^{\mathrm{T}}X$和$XX^{\mathrm{T}}$的特征值，$1\leqslant i\leqslant N$。$U$为$XX^{\mathrm{T}}$的特征向量，$V$为$X^{\mathrm{T}}X$的特征向量。$\Sigma$矩阵中只有主对角线有值，其余部分为0，如式(7)所示：

$$\Sigma=\begin{bmatrix}\Sigma'&0_{r\times(N-r)}\\0_{(M-r)\times r}&0_{(M-r)\times(N-r)}\end{bmatrix} \tag{7}$$

Σ'为Σ的主要子矩阵，只有主对角线有值，大小为$r\times r$。而U和V则表示为：

$$U=[U'\quad U''],V^{\mathrm{T}}=\begin{bmatrix}V'^{\mathrm{T}}\\V''^{\mathrm{T}}\end{bmatrix} \tag{8}$$

U'为$M\times r$的矩阵，U''为$M\times(M-r)$的矩阵。V'为$N\times r$的矩阵，V''为$N\times(N-r)$的矩阵。

因此式(6)可表示为：

$$X=[U'\quad U'']\begin{bmatrix}\Sigma'&0_{r\times(N-r)}\\0_{(M-r)\times r}&0_{(M-r)\times(N-r)}\end{bmatrix}\begin{bmatrix}V'^{\mathrm{T}}\\V''^{\mathrm{T}}\end{bmatrix}=U'\Sigma'V'^{\mathrm{T}} \tag{9}$$

Σ矩阵中大部分为0，因此可省略一些不必要的运算，降低计算量。代入式(5)，得到

$$\min_{\beta}\|Y-U'\Sigma'V'^{\mathrm{T}}\beta\| \tag{10}$$

再经过转换后，可得到

$$\min_{\beta}\|U'^{\mathrm{T}}Y-\Sigma'V'^{\mathrm{T}}\beta\| \tag{11}$$

式(11)的解则表示如下：

$$\beta^*=(\Sigma'V'^{\mathrm{T}})^+U'^{\mathrm{T}}Y \tag{12}$$

其中，$(\Sigma'V'^{\mathrm{T}})^+$是$\Sigma'V'^{\mathrm{T}}$的伪逆矩阵，

$$\begin{aligned}(\Sigma'V'^{\mathrm{T}})^+&=(V'^{\mathrm{T}})^{\mathrm{T}}(V'^{\mathrm{T}}(V'^{\mathrm{T}})^{\mathrm{T}})^{-1}(\Sigma'^{\mathrm{T}}\Sigma')^{-1}\Sigma'^{\mathrm{T}}\\&=V'(V'^{\mathrm{T}}V')^{-1}\Sigma'^{-1}(\Sigma'^{\mathrm{T}})^{-1}\Sigma'^{\mathrm{T}}\\&=V'\Sigma'^{-1}\end{aligned} \tag{13}$$

最后，将$(\Sigma'V'^{\mathrm{T}})^+$替换为$V'\Sigma'^{-1}$，即可求出式(2)的最小平方解，如式(14)所示：

$$\beta^*=V'\Sigma'^{-1}U'^{\mathrm{T}}Y \tag{14}$$

3 迭代式分割与合并算法

本文提出了迭代式分割与合并奇异值分解最小平方估计法，并分析了其时间与空间复杂度。

3.1 迭代式分割与合并奇异值分解最小平方估计法

迭代式分割与合并奇异值分解最小平方估计法（iterative divide and merge SVD-based least squares estimator，IDMSVD-LSE）主要用来解决大型数据求最小平方估计的问题。如果直接使用奇异值分解最小平方估计法求最小平方解，当训练样本数（M）非常大时，必须耗费非常庞大的运算时间和内存空间。因此，迭代式分割与合并算法先利用奇异值分解缩减训练样本数（M）的大小，再利用奇异值分解最小平方估计法求解，以节省运算时间和内存空间。

迭代式分割与合并算法会有一个以上的阶层数，阶层从1开始计数。对于第j层来说，$X(j)$和$Y(j)$为输入矩阵，$X(j+1)$和$Y(j+1)$为输出矩阵，$M(j)$为训练样本数。假设$j=1$，$X(1)$等于式(3)的X矩阵，$Y(1)$等于式(4)的Y矩阵，$M(1)$等于原始训练样本的M。每个阶层可分成3个步骤：第1

步是将 $X(j)$拆成数个子矩阵;第 2 步是利用奇异值分别分解每一个子矩阵;第 3 步是将所有的子矩阵合并,得到 $X(j+1)$和 $Y(j+1)$。$X(j+1)$和 $Y(j+1)$矩阵会比 $X(j)$和 $Y(j)$矩阵还要小。

接下来,将对迭代式分割与合并奇异值分解最小平方估计法做完整的说明和介绍。在第 j 层时,$X(j)$和 $Y(j)$为输入矩阵。首先将 $X(j)$拆分为 s_j 个子矩阵,$X(j)=\bigcup_{i=1}^{s_j}X_{j,i}$,$X_{j,i}$为 $M_{j,i}\times N$ 矩阵,$M_{j,i}>0$,而 $M(j)=\sum_{i=1}^{s_j}M_{j,i}$,$s_j=\left\lceil\frac{N(j)}{L}\right\rceil$,$L$ 是使用者所设定的常数,L 必须大于 N,使得 $M_{j,1}=M_{j,2}=\cdots=M_{j,s_j-1}=L$,而 $M_{j,s_j}=M(j)-(s_j-1)L$,否则没有办法达到缩减训练样本数的目的。同样地,需将 $Y(j)$拆分为 s_j 个子矩阵,得 $Y(j)=\bigcup_{i=1}^{s_j}Y_{j,i}$。如此,可得到式(15):

$$\min_{\beta}\|Y-X\beta\|=\min_{\beta}\left\|\begin{bmatrix}Y_{j,1}\\Y_{j,2}\\\vdots\\Y_{j,s_j}\end{bmatrix}-\begin{bmatrix}X_{j,1}\\X_{j,2}\\\vdots\\X_{j,s_j}\end{bmatrix}\beta\right\| \tag{15}$$

将 $X(j)$的子矩阵分别利用奇异值分解,得到

$$X_{j,i}=U_{j,i}\Sigma_{j,i}V_{j,i}^{\mathrm{T}} \tag{16}$$

$U_{j,i}$为 $M_{j,i}\times M_{j,i}$的正交矩阵,$V_{j,i}$为 $N\times N$ 的正交矩阵,$\Sigma_{j,i}$为 $M_{j,i}\times N$ 的对角矩阵,$rank(X_{j,i})=r_{j,i}$,$r_{j,i}$为非零的对角元素。由式(7)得到

$$\Sigma_{j,i}=\begin{bmatrix}\Sigma'_{j,i} & 0\\0 & 0\end{bmatrix} \tag{17}$$

$\Sigma'_{j,i}$为 $r_{j,i}\times r_{j,i}$的对角矩阵。由式(8)得到

$$U_{j,i}=\lfloor U'_{j,i}\quad U''_{j,i}\rfloor,V_{j,i}^{\mathrm{T}}=\begin{bmatrix}V_{j,i}^{'\mathrm{T}}\\V_{j,i}^{''\mathrm{T}}\end{bmatrix} \tag{18}$$

$U'_{j,i}$为 $M_{j,i}\times r_{j,i}$的矩阵,$U''_{j,i}$为 $M_{j,i}\times(M_{j,i}-r_{j,i})$的矩阵。$V'_{j,i}$为 $N\times r_{j,i}$的矩阵,$V''_{j,i}$为 $N\times(N-r_{j,i})$的矩阵。由式(5)和式(11)合并矩阵,使得

$$Y(j+1)=\begin{bmatrix}Y_1(j)\\Y_2(j)\\\vdots\\Y_{s_j}(j)\end{bmatrix}=\begin{bmatrix}U_{j,1}^{'\mathrm{T}}Y_{j,1}\\U_{j,2}^{'\mathrm{T}}Y_{j,2}\\\vdots\\U_{j,s_j}^{'\mathrm{T}}Y_{j,s_j}\end{bmatrix} \tag{19}$$

$$X(j+1)=\begin{bmatrix}X_1(j)\\X_2(j)\\\vdots\\X_{s_j}(j)\end{bmatrix}=\begin{bmatrix}\Sigma'_{j,1}V_{j,1}^{'\mathrm{T}}\\\Sigma'_{j,2}V_{j,2}^{'\mathrm{T}}\\\vdots\\\Sigma'_{j,s_j}V_{j,s_j}^{'\mathrm{T}}\end{bmatrix} \tag{20}$$

$Y_i(j)$为 $r_{j,i}\times 1$ 的矩阵,$Y(j+1)$为 $M(j+1)\times 1$ 的矩阵。$X_i(j)$为 $r_{j,i}\times N$ 的矩阵,$X(j+1)$为 $M(j+1)\times N$ 的矩阵。而 $M(j+1)=\sum_{i=1}^{s_j}r_{j,i}$。因此最小平方估计问题可表示为:

$$\min_{\beta}\|Y(j)-X(j)\beta\| \tag{21}$$

式(21)和下列公式是相同的。

$$\min_{\beta}\|Y(j+1)-X(j+1)\beta\| \tag{22}$$

假设经过了 $z-1$ 层后,$z\geqslant 1$,$M(z)$会变得非常小。例如,$M(z)\leqslant L$。由式(5)可得到

$$\min_{\beta}\|Y(z)-X(z)\beta\| \tag{23}$$

最后直接使用奇异值分解最小平方估计法求最小平方解。利用式(14)得到

$$\beta^*=V(z)'\Sigma(z)'^{-1}U(z)'^{\mathrm{T}}Y(z) \tag{24}$$

即可得到式(23)的最小平方解。

迭代式分割与合并奇异值分解最小平方估计法的详细流程与步骤如算法 1 所示。

算法 1

INPUT:X,Y,M,N,L(L>2N)
OUTPUT:Regression coefficients β*
1. Set X(1)=X,Y(1)=Y,M(1)=M;
2. j=1;
3. repeat
4. //Step 1
5. Calculate the number of submatrices, $s_j=\lceil\frac{M(j)}{L}\rceil$;
6. Divide X(j) into s_j submatrices, such that $X(j)=\bigcup_{i=1}^{s_j}X_{j,i}$;
7. Divide Y(j) into s_j submatrices, such that $Y(j)=\bigcup_{i=1}^{s_j}Y_{j,i}$;
8. //Step 2
9. for i=1 to s_j do
10. Decompose $X_{j,i}$ into three matrices $U_{j,i}$, $\Sigma_{j,i}$, and $V_{j,i}$;
11. end for
12. // Step 3
13. Obtain Y(j+1);
14. Obtain X(j+1);
15. j=j+1;
16. until M(j)≤L;
17. Decompose X(j) and obtain three matrices U(j)′, Σ(j)′, and V(j)′;
18. Calculate the optimal solution β*.

3.2 复杂度分析与比较

在此将分析与比较奇异值最小平方估计法(SVD-LSE)和迭代式分割与合并奇异值分解最小平方估计法(IDMSVD-LSE)的复杂度。

首先分析与比较空间复杂度。SVD-LSE 必须计算整个 X 矩阵,所产生的最大矩阵为 U,大小为 $M\times M$。因此 SVD-LSE 的空间复杂度是 $O(M^2)$。而 IDMSVD-LSE 每一次会分解一个小矩阵 $X_{j,i}$,所产生最大的矩阵为 $U_{j,i}$,大小为 $L\times L$。因此 IDMSVD-LSE 的空间复杂度是 $O(L^2)$。由此可知,如果训练样本数非常多,M 会很大,将导致 SVD-LSE 需要非常大的空间来运算;而如果训练样本的维度不高,L 将远小于 M,因此 IDMSVD-LSE 所需要的运算空间会比 SVD-LSE 小很多。

接着分析与比较时间复杂度。SVD-LSE 在分解矩阵的时候,见式(6),其需要的时间为 $O(4M^2N+8MN^2+9N^3)$,再加上处理的时间,见式(12),其需要的时间为 $O(Nr^2+Mr+Nr)\approx O(N^3+MN+N^2)$。因此,SVD-LSE 的时间复杂度为:

$$O(4M^2N+8MN^2+9N^3)+O(N^3+MN+N^2)\approx O(M^2N) \tag{25}$$

而 IDMSVD 则需要考虑每一个阶层的运算时间。

首先,在第一层时

$$s_1=\left\lceil\frac{M(1)}{L}\right\rceil\approx\frac{M}{L} \tag{26}$$

分解所需要的时间为:

$$s_1(4L^2N+8LN^2+9N^3) \tag{27}$$

同样,在第 j 层时

$$s_j\approx\left\lceil\frac{M}{L}\left(\frac{N}{L}\right)^{j-1}\right\rceil \tag{28}$$

分解所需要的时间为：

$$s_j(4L^2N+8LN^2+9N^3) \tag{29}$$

因此，从第一层到第 $z-1$ 层，见式(23)与 $X(z)$，其分解所需的时间为：

$$\begin{aligned}&(4L^2N+8LN^2+9N^3)(1+\sum_{j=1}^{z-1}s_j)\\&\approx(4L^2N+8LN^2+9N^3)(1+\sum_{j=1}^{z-1}\frac{M}{L}(\frac{N}{L})^j)\\&\approx(4L^2N+8LN^2+9N^3)(1+\frac{M}{L-N})\end{aligned} \tag{30}$$

同样地，式(19)与式(20)从第一层到第 $z-1$ 层处理时，所需的时间为：

$$(LN+N^3)\sum_{j=1}^{z-1}s_j\approx(LN+N^3)(\frac{M}{L-N}) \tag{31}$$

最后必须求出 β^*，见式(24)，所需的时间为：

$$O(Nr^2+Lr+Nr)\approx O(N^3+LN) \tag{32}$$

因此，IDMSVD 的时间复杂度为式(30)－式(32)的总和，如

$$(4L^2N+8LN^2+9N^3+LN+N^3)(1+\frac{M}{L-N}) \tag{33}$$

近似值为：

$$O(MLN) \tag{34}$$

而 $M\gg L>N$。由此可知 IDMSVD-LSE 运算所花费的时间将比 SVD-LSE 节省许多。

4 并行迭代式分割与合并算法

通过 CUDA 技术，迭代式分割与合并最小平方估计法可利用 GPU 来实现并行运算。同样将输入矩阵拆分为数个子矩阵，每个子矩阵分别交由一个 thread 执行，因为每个子矩阵是互相独立的，彼此之间不需要互相传递数据，所以每个 block 只包含一个 thread。由于迭代式分割与合并算法可能执行数个阶层，因此每一个阶层则会执行一次 GPU 的程序(kernel)。以下将详细介绍如何以 CUDA 来实现平行化迭代式分割与合并算法。

以第 j 层为例，输入矩阵为 $X(j)$和 $Y(j)$，$X(j)$的大小为 $M(j)\times N$，$Y(j)$的大小为 $M(j)\times 1$，计算目前子矩阵数量 s_j，$s_j=\left\lceil\frac{M(j)}{L}\right\rceil$。接着将输入矩阵 $X(j)$与 $Y(j)$从 host 复制到 device，在 device 端的输入矩阵称为 $Xd(j)$和 $Yd(j)$。每个子矩阵通过式(16)会得到 3 个矩阵，分别是 U、Σ 和 V，通过式(19)与式(20)会得到输出矩阵 $Xd(j+1)$和 $Yd(j+1)$，因此 host 需先宣告新的矩阵空间在 device 的内存中，其中 Ud 的大小为 $(L\times N)\times s_j$，Σd 与 Vd 的大小为 $(N\times N)\times s_j$，$Xd(j+1)$的大小为 $(N\times N)\times s_j$，$Yd(j+1)$的大小为 $(N\times 1)\times s_j$，在此假设每个子矩阵的 rank 等于 N。在调用 kernel function 时，需要给定 grid 与 block 的大小，这里将 block 与 thread 都宣告为一维的形式。grid 的大小为 g_j，$g_j=s_j$，block 的大小为 b，$b=1$。Thread 的总数为 $g_j\times b$。block 中包含一个 thread，处理一个子矩阵，而该如何分配哪一个 thread 处理哪一个子矩阵？在 kernel 中可使用 *blockIdx* 与 *threadIdx* 等参数，这些参数是 CUDA 内建的参数。*blockIdx.x* 可以取得目前的 block 的索引，$0\leqslant blockIdx.x<g_j$；*threadIdx.x* 可以取得目前 thread 的索引，因为 block 大小为 1，故 *threadIdx.x* $=0$。令 $idx=blockIdx.x$，有了索引后，就可控制哪一个 thread 处理哪一个子矩阵。利用 SVD 将子矩阵 $X_{j,idx}$ 分解，见式(16)，再通过式(19)与式(20)，每个 thread 最后都会得到 $Yd_{idx}(j)$与 $Xd_{idx}(j)$，而 $Yd(j+1)=\bigcup_{idx=0}^{g_j-1}Yd_{idx}(j)$，$Xd(j+1)=\bigcup_{idx=0}^{g_j-1}Xd_{idx}(j)$。接着回到 host 端，将 device 端的输出矩阵 $Xd(j+1)$和 $Yd(j+1)$复制回 host 端，得到 $X(j+1)$和 $Y(j+1)$。最后，需将 device 中的矩阵删除，即可进入下一个阶层。重复以上的动作，当 $M(j)<L$ 时，则停止迭代，通过式(6)与式(24)即可求出最小平方解。完整的步骤与流程见图 1。其中 kernel function 则表示于算法 2 中。

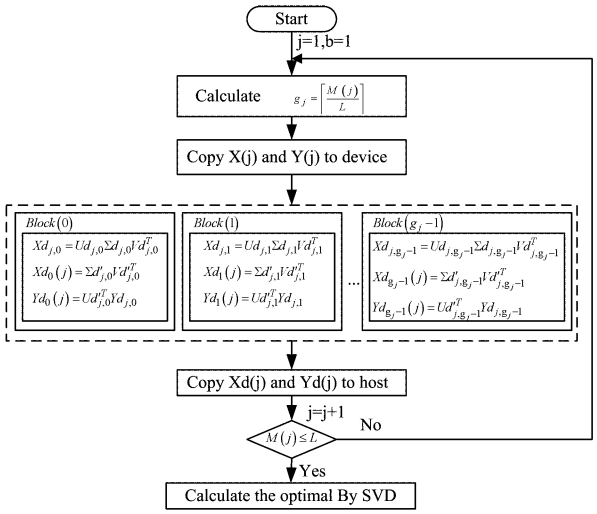

图 1 并行迭代式分割与合并算法流程图

算法 2

Input：Xd(j)，Yd(j)

Output：Xd(j+1)，Yd(j+1)

1. Set idx=blockIdx.x；
2. Decompose $Xd_{j,idx}$ into three matrices $Ud_{j,idx}$，$\Sigma d_{j,idx}$，and $Vd_{j,idx}$；
3. Obtain $Yd_{idx}(j)=Ud'^T_{j,idx}Yd_{j,idx}$；
4. Obtain $Xd_{idx}(j)=\Sigma d'_{j,idx}Vd'^T_{j,idx}$；
5. $Yd(j+1)=\bigcup_{idx=0}^{g_j-1}Yd_{idx}(j)$；
6. $Xd(j+1)=\bigcup_{idx=0}^{g_j-1}Xd_{idx}(j)$.

5 实验分析

利用 CUDA 实现并行迭代式分割与合并算法，并将其与 IDMSVD 及其他方法比较。为了方便，在此将奇异值分解最小平方估计法简称为 SVD-LSE，将迭代式分割与合并奇异值分解最小平方估计法简称为 IDMSVD-LSE，将以 CUDA 实现的并行迭代式分割与合并最小平方估计法简称为 PIDMSVD-LSE。

5.1 实验数据及环境

实验是以随机数的方式来产生不同大小的矩阵，如表 1 和表 2 所列，共有 8 组不同大小的数据，分别编号为 GDS1，GDS2，…，GDS8。其中 M 表示训练样本的数目，N 表示训练样本的维度加 1。

表 1 GPU 实验数据 a

Dataset	GDS1	GDS2	GDS3	GDS4
M	1000	10000	100000	500000
N	11	11	11	11

表 2　GPU 实验数据 b

Dataset	GDS5	GDS6	GDS7	GDS8
M	1000	10000	100000	500000
N	26	26	26	26

实验使用 ASUS 服务器，Intel® Xeon™ X3330 2.66Hz 处理器，8G 内存。操作系统为 Win XP 64bit。GPU 为 NVIDIA GeForce GTX 570，有 480 个 CUDA 处理核心，内存带宽 152GB/秒，1280MB 内存。使用 MATLAB2010 执行所有实验的方法。

5.2　GPU 实验

首先，比较 PIDMSVD-LSE 与 IDMSVD-LSE 和 SVD-LSE 的执行时间，如表 3 和表 4 所列。其中，符号“—”表示内存不足无法执行完成。表 3 和表 4 中 IDMSVD-LSE 与 PIDMSVD-LSE 的运行时间都明显比 SVD-LSE 的运行时间少。以数据集 GDS2 和 GDS6 为例，IDMSVD-LSE 比 SVD-LSE 的运行速度快了 50 倍左右。但在执行 GDS5 数据集时，PIDMSVD-LSE 所需时间却比其他方法都长，这是因为数据量太小的关系，PIDMSVD-LSE 没法发挥 GPU 的优势，也就是并行处理的子矩阵太少，数据在 host 与 device 之间复制的时间大于并行处理所节省的时间。当 M 足够大时，PIDMSVD-LSE 将比 IDMSVD-LSE 的运行时间还少，此时平行处理所节省的时间大于数据在 host 与 device 之间复制所花费的时间。如数据集 GDS8，IDMSVD-LSE 需要 27204 毫秒，而 PIDMSVD-LSE 只需要 5452 毫秒，少了将近 4/5 的运行时间，而 SVD-LSE 则无法执行完成。SVD-LSE 内存不足的原因是 SVD 分解产生的矩阵 U 的大小是 $M\times M$，矩阵 U 太大，导致 MATLAB 回传内存不足。

表 3　GPU 实验：不同方法执行时间比较 a(单位：毫秒)

Method(L=200)	GDS1	GDS2	GDS3	GDS4
SVD-LSE	56	5255	—	—
IDMSVD-LSE	9	98	1093	8225
PIDMSVD-LSE	33	69	261	971

表 4　GPU 实验：不同方法执行时间比较 b(单位：毫秒)

Method(L=200)	GDS5	GDS6	GDS7	GDS8
SVD-LSE	120	11827	—	—
IDMSVD-LSE	23	231	2861	27204
PIDMSVD-LSE	187	390	1417	5452

PIDMSVD-LSE 与 IDMSVD-LSE 不同 L 的运行时间比较见图 2－图 5，其分别表示了数据集 GDS1、GDS3、GDS5 与 GDS7 的情况。从图 3 与图 5 可明显看出 L 对于 IDMSVD-LSE 的运行时间影响较大，例如图 3 的 GDS3 数据集，当 $L=40$ 时，IDMSVD-LSE 需 3621 毫秒，PIDMSVD-LSE 需 305 毫秒，而当 $L=100$ 时，IDMSVD-LSE 需 965 毫秒，PIDMSVD-LSE 需 241 毫秒，IDMSVD-LSE 在设定不同 L 时，会明显影响执行的时间，这是因为 L 设定太小时，要计算的子矩阵太多，且 IDMSVD-LSE 是顺序执行的，导致需要较长的运行时间；而 PIDMSVD-LSE 是并行地处理子矩阵，因此 L 变化时对于运行时间的影响不大。

由图 2 与图 4 可以看出，PIDMSVD-LSE 设定不同 L 时产生的曲线是呈现类似锯齿状的，原因是在设定不同 L 时，会导致 PIDMSVD-LSE 的总阶层数不同，以及子矩阵的个数不同，这两种原因使得曲线呈现锯齿状的趋势。例如图 2 的 GDS1，对于不同 L 的阶层数，当 $L=30$ 时，需 26 毫秒，阶层数为 5，当 $L=40$ 时，需 24 毫秒，阶层数为 4，因为阶层数少了一层，所以时间减少了 2 毫秒。当 $L=50$ 时，需 29 毫秒，因为 L 变大，子矩阵的大小变大且子矩阵的数目变少，分散的程度变小而阶层数是相同的，所以运行时间比 $L=40$ 多了 5 毫秒。而当 $L=60$ 时，需 23 毫秒，阶层数为 3，比 $L=50$ 时少了一层，运行时间减少 6 毫秒，以此类推。因此不同 L 的运行时间画出的曲线呈现锯齿状。

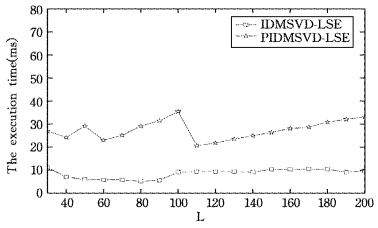

图 2　GPU 实验：不同 L 的执行时间比较(GDS1)

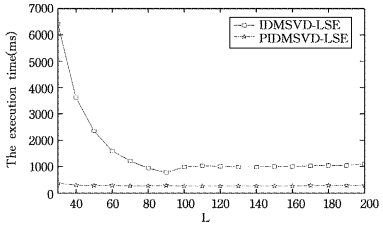

图 3　GPU 实验：不同 L 的执行时间比较(GDS3)

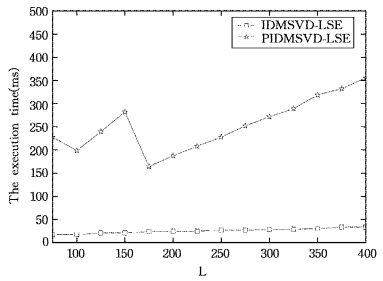

图 4　GPU 实验：不同 L 的执行时间比较(GDS5)

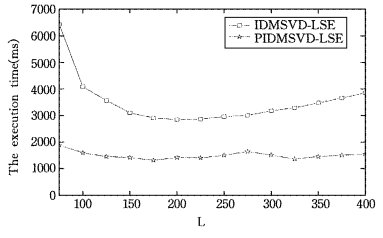

图 5　GPU 实验：不同 L 的执行时间比较(GDS7)

结束语　本文中提出的迭代式分割与合并奇异值分解最小平方估计法包含数个阶层，在每一个阶层，将输入矩阵细分为数个子矩阵，子矩阵的大小是可以动态决定的，再分别利用奇异值分解来缩减子矩阵的大小，将结果合并后作为下一个阶层的输入，以同样的方式执行数个阶层，直到将矩阵缩减到够小之后，再利用奇异值分解最小平方估计法求得最小平方解。实验结果显示，我们提出的方法可以有效地解决一般奇异值分解耗费时间与内存空间的问题。迭代式分割与合并奇异值分解最小平方估计法在每个阶层中分解每个子矩阵时，它们之间是互相独立的，因此可以同时执行。利用 GPU 来进一步改善 IDMSVD-LSE 的运行时间。本文提出的并行迭

代式分割与合并奇异值分解最小平方估计法的实验结果显示，利用 GPU 可有效地改善 IDMSVD-LSE 的运行时间，虽然对于较小的数据集，使用 GPU 没有办法得到改善，但执行的时间与单机执行的时间是相近的。

参 考 文 献

[1] Montgomery D C, Peck E A, Vining G G. Introduction to linear regression analysis(4th ed)[M]. Hoboken, N. J. , USA: Wiley-Interscience, 2006: 68-113

[2] Myers R H, Montgomery D C, Vining G G, et al. Generalized linear models: with applications in engineering and the sciences (2nd ed)[M]. Hoboken, N. J. , USA: Wiley-Interscience, 2010: 217-339

[3] Lee S-J, Ouyang C-S. A neuro-fuzzy system modeling with self-constructing rule generation and hybrid SVD-based learning [J]. IEEE Transactions on Fuzzy Systems, 2003, 11(3): 341-363

[4] Foster L V. Solving rank-deficient and ill-posed problems using UTV and QR factorizations [J]. SIAM Journal on Matrix Analysis and Applications, 2003, 26(2): 682-600

[5] Moler C B. Numerical computing with matlab[M]. Philadelphia, PA, USA: Society for Industrial Mathematics, 2004: 71-165

[6] Hari V. Accelerating the SVD block-jacobi method [J]. Computing, 2006, 76(1): 27-63

[7] Yamamoto Y, Fukaya T, Uneyama T, et al. Accelerating the singular value decomposition of rectangular matrices with the CSX600 and the integrable SVD[J]. Lecture Notes in Computer Science, 2007, 46(7): 340-346

[8] Kondaa T, Nakamura Y. A new algorithm for singular value decomposition and its parallelization [J]. Parallel Computing, 2009, 36(6): 331-344

[9] Bečka M, Okša G, Vajteršic M, et al. On iterative QR pre-processing in the parallel block-jacobi SVD algorithm[J]. Parallel Computing, 2009, 36(6): 297-307

[10] Ltaief H, Kurzak J, Dongarra J. Parallel two-sided matrix reduction to band bidiagonal form on multicore architectures [J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(4): 417-423

(上接第 30 页)

为评估频谱空间大小对 STS 算法时间性能的影响，本文将不同频谱空间大小、不同频点密集度在表 2 中的仿真参数(频谱空间大小除外，另外从站数为 15)下分别运行 50 次，得到 STS 的平均性能，如图 7 所示。

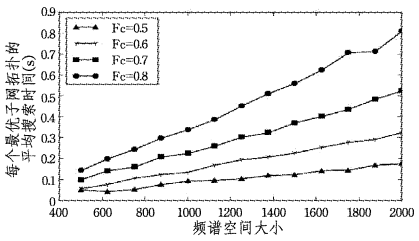

图 7 STS 算法在不同频谱空间大小下的时间性能

结束语 本文描述了树形认知无线电网络多簇子网模型；针对其频谱决策问题，根据 QoS 需求建立了基于优先级的多目标优化模型；提出了可以实时获得最优子网拓扑的 STS 算法。STS 算法根据 LLEF 策略及标记已访问的子网第 1 个簇拓扑构造不重复的搜索空间，以当前最优解更新的搜索步长为启发式条件，贪心搜索增长率更高的子网拓扑。引入动态门限筛选簇拓扑，从而排除在最优子网拓扑中不存在的簇拓扑。在仿真实验中，对算法的时间性能作了全面的分析，实验结果表明 STS 算法最坏情况下在 0.37 秒内生成最优子网拓扑，满足实时性需求，其在较大频谱空间下时间性能表现依然良好。

参 考 文 献

[1] European Radio communications Committee (ERC). European table of frequency allocations and utilizations frequency range 9kHz to 275GHz[R]. ERC Report 25, January 2002

[2] Natasha D, Mai Vu, Vahid T. Cognitive Radio Networks [J]. IEEE Signal Processing Magazine, 2008, 25(6): 12-23

[3] Akyildiz I F, Lee W Y, Vuran M C, et al. A survey on spectrum management in cognitive radio networks [J]. Communications Magazine, IEEE, 2008, 46(4): 40-48

[4] Talat S T, Wang L C. Load-Balancing Spectrum Decision for Cognitive Radio Networks with Unequal-Width Channels[C]// Vehicular Technology Conference (VTC Fall), 2012 IEEE. IEEE, 2012: 1-5

[5] Liao M X, He X X, Jiang X H. Optimal Algorithm for Cognitive Spectrum Decision Making [C]//COCORA 2012, The Second International Conference on Advances in Cognitive Radio. 2012: 50-56

[6] 杨云，章国安，邱恭安. 认知无线 Mesh 网络中基于概率的贪婪频谱决策技术研究[J]. 计算机科学，2012，39(B06)：163-165

[7] Tsagkaris K, Katidiotis A, Demestichas P. Neural network-based learning schemes for cognitive radio systems[J]. Computer Communications, 2008, 31(14): 3394-3404

[8] 张北伟，胡琨元，朱云龙. 基于博弈论的认知无线电频谱分配[J]. 计算机应用，2012，32(9)：2408-2411

[9] Chung S T, Kim S J, Lee J, et al. A game-theoretic approach to power allocation in frequency-selective Gaussian interference channels[C]// Proc. IEEE International Symposium on Inform. Theory. Pacifico, 2003

[10] Zhang W. Handover decision using fuzzy MADM in heterogeneous networks[C]// Wireless Communications and Networking Conference. WCNC 2004 IEEE. IEEE, 2004, 2: 653-658

[11] 瞿越，鲜永菊，徐昌彪. 基于用户需求的图着色论频谱分配算法[J]. 计算机应用，2011，31(3)：602-605

[12] 吴非，陈劼，廖楚林，等. 认知无线电网络中基于需求的多小区频谱分配算法[J]. 计算机应用，2008，28(1)：14-16

[13] 张北伟，朱云龙，胡琨元. 基于粒子群算法的认知无线电频谱分配算法[J]. 计算机应用，2011，31(12)：3184-3186

[14] Fan Zhong-ji, Liao Ming-xue, He Xiao-xin, et al. Efficient Algorithm for Extreme Maximal Biclique Mining in Cognitive Frequency Decision Making[C]// 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN). IEEE, 2011: 25-30

[15] Ji Pan-pan, Liao Ming-xue, He Xiao-xin, et al. Extreme Maximal Weighted Frequent Itemset Mining for Cognitive Frequency Decision Making[C]// 2011 International Conference on Computer Science and Network Technology (ICCSNT). IEEE, 2011, 1: 267-271