Parallel Principal Components Analysis

Hao Yu, Peng
Computer Science
National Yang Ming Chiao Tung University
haoyu0970624763@gmail.com

Yan Fu, Liu
Computer Science
National Yang Ming Chiao Tung University
sam901009@gmail.com

Chia Wen, Chang
Computer Science
National Yang Ming Chiao Tung University
a650993@gmail.com

Abstract

This project contributes to the implementation of parallel Principal Components Analysis. We use three different approaches to accomplish this project, such as Pthread, OpenMP, and CUDA. In Pthread and OpenMP, our implementation can reach up to 4x speedup. And for CUDA, although the speedup did not reach our expectation, it still has over 6x speedup in the matrix multiplication section.

Introduction

In many machine learning model with lots of features, we will do data preprocessing to reduce the input dimension. PCA is one of the dimension reduction methods, it uses the eigenvalues and eigenvectors to create a transformation matrix which transforms input data the lower dimension. Figure.1 shows the process of PCA. We can observe that there are lots of matrix operation in it. Therefore, it can be parallelized.

Figure 1: The process of PCA

We use Jacobi Algorithm to compute the eigenvalues and the eigenvectors of the input matrix. Figure.2 is the pseudo code of Jacobi Algorithm. In this project, we will focus on matrix multiplication and Jacobi Algorithm to reach the higher performance.

Initial eigenvectors matrix V

while not convergence:

Find the largest element a_{pq}

Construct orthonormal matrix U

Update V by U

Sort eigenvalues / eigenvectors

Figure 2: The pseudo code of Jacobi Algorithm

Proposed Solution

In Pthread version, we divide the whole matrix into few blocks. Each thread will handle a part of the matrix. Figure.3 show how we divide the matrix. After each thread finish the computation, it will store the result into output matrix.

In OpenMP version, the idea is same as Pthread version. We use "*#pragma omp parallel for*" to create threads to deal with each block.

Figure 3: Dividing matrix in pthread and OpenMP

In CUDA, we transfer matrix to GPU and let each thread calculate one pixel in matrix.

Experimental Methodology

We generate an input data to evaluate the performance of each method. The input data contains 256 data with 512 features. We test our code on machine with
"Intel i5-7500 CPU 3.40GHz" and "NVIDIA GeForce GTX 1060".

Experimental Results

We tested each method with input data 10 times and take the average execution time as the result. Figure.4 shows the comparison between Serial, Pthread, and OpenMP. We can observe that OpenMP has the best performance and the execution time of Pthread is a little bit longer than OpneMP. The reason why Pthread is lower might be the way we implement Pthread, we spent some time on comparison between each thread like getting the max value. However, we have "reduction pragma" in OpenMP. Besides, we found that

OpenMP also works faster than Pthread on matrix multiplication.

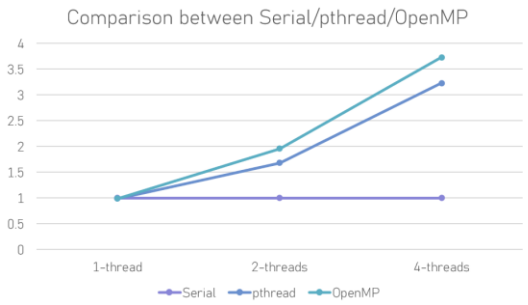
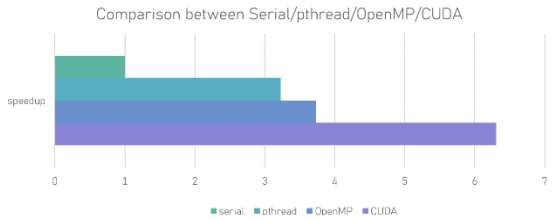


Figure 4: Comparison between Serial/Pthread/OpenMP

Figure.5 show the comparison between CUDA and Serial, Pthread, OpenMP in the matrix multiplication section. We can find that CUDA reaches the best performance (over 6x speedup). The reason why CUDA only has 6x speedup might be the matrix is not large enough. The overhead of transferring matrix from CPU to GPU is too large.



Conclusion

In this project, OpenMP version is the best version of Parallel Principal Analysis. It can reach up to 4x speedup overall. But we think CUDA might be able to work faster if we can find a way to deal with the overhead mentioned above. The other thing we can do is put our parallel PCA into a real machine learning model to see how it affect the accuracy.

Reference

[1] [筆記]主成分分析(PCA) - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天
from :https://ithelp.ithome.com.tw/articles/10211877

[2] C.F. Van Loan G. H. Golub. Matrix COmputations[M]. John Hopkins University Press, Baltimore and London, second edition, 1993.

[3]ccjou(2013) 主成分分析 | 線代啟示錄 form: https://ccjou.wordpress.com/2013/04/15/%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90/