Report of Rcpp Project

Haibo Jin
Student number: 014343698

Contents

1 Introduction

Our task of the project is to develop an C++ program with Rcpp. The program implements an algorithm to generate a genetic relatedness matrix between individuals from a genome-wide panel of SNPs stored in a GEN file.

The GEN file stores information of gene of different individuals in a one-line-per-SNP format. For every line of SNP, there are $5 + 3n$ entries, where the first five strings are useless in this project. The remaining data are sets of genotype probabilities of different individuals. Every set consists of three genotypes, namely AA, AB and BB.

Based on these genotype probabilities, we can calculate the genetic relatedness matrix R as follows

$$R_{n\times n} = \begin{bmatrix} r_{11} & r_{21} & \cdots & r_{n1} \\ r_{21} & r_{22} & \cdots & r_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{bmatrix}$$

r_{ij} in the matrix can be calculated as

$$r_{ij} = \frac{1}{|M_{ij}|} \sum_{l \in M_{ij}} \frac{(z_{li} - 2p_l)(z_{lj} - 2p_l)}{2p_l(1-p_l)}$$

The relatedness matrix indicates the genetic relatedness between two chosen people, which can be used to study genetics and some relevant research.

2 Computational Approach

The code can be divided into three parts: initialization, loop over every line of SNPs and update of matrix.

2.1 Initialization

Some pre-work, like declare variables, open GEN files and initialize matrices. The for loop of initialization can be replaced by memset() function, even though the improvement is not so obvious.

2.2 Loop over every line of SNPs

This is the main part of the project. On every iteration of all the SNPs, we should first read a line of loci, then calculate the expectation of individual's specific genotype, the normalized genotype probabilities, the estimated allele frequency of allele B of the loci, etc. During the iteration, we need to exclude the loci if it is in one of the three excpetions: the percentage of individuals with missing genotype data is above 75%; The minor allele genotype $min(p_l, 1-p_l)$ is less than 0.05; Or the measure I_v is less 0.25.

Additionally, we need to record the vector v_m and v_r which are used to calculate matrix M and R using ssyrk(). This ssyrk() is a subroutine interface of openBLAS, which can update matrix efficiently. Theoretically, this subroutine should be much faster than manually updating matrix in nested for-loops. ssyrk() is used in a form that: $C = \alpha * A * A^T + \beta * C$. Here matrix M and R can be seen in the position of C in the formula, and v_m and v_r can be seen in the position of A. α and β should be one here. v_m indicates whether the genotype of an individual is missing or not over one loci. v_r is the element-wise product of v_m and a vector calculated before. For every line of loci, v_m and v_r are vectors. There is another subroutine of openBLAS named ssyr(), which can only update a matrix over a vector. If we use this ssyr(), we need to update for p times(p is the number of lines), which is not so efficient. Instead, if we use

ssyrk(), by recording all the v_m and v_r of all locus, v_m and v_r become matrices and so that we just need to update matrix M and R once. The definition of the function ssyrk() is listed below:

void cblas_ssyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans,const blasint N, const blasint K, const float alpha, const float *A, const blasint lda, const float beta, float *C, const blasint ldc);

2.3 Update of Matrix

After traversing all the SNPs, we get v_m and v_r matrices. Then we use the two matrices to generate matrix M and R. Two simple calls of ssyrk() can finish the computation quickly. At last, use the matrix M to divide R element-wise to get the normalized final R matrix. The R matrix is an upper triangular matrix since the relatedness matrix is a symmetric matrix, thus there is no need to compute all the elements.

3 Progress Summary

Roughly, there are four versions of my project:

- First version: Programmed in C++ with Rcpp, no optimization.
- Second version: Based on the previous version, use openMP to parallel some part of for-loops. Also, optimize the structure of the code by memorizing some repeated computations.
- Third version: Based on the previous version, use dynamic array instead of numeric matrix in Rcpp.
- Fourth version: Based on the previous version, use a subroutine interface of openBLAS to update matrix M and R instead of three-nested for-loops. Moreover, initialize matrix M and R with memset() function instead of for-loops.

4 Results

The output of the program for all the training sets are correct. The running time of the four versions tested with training set 2(500 by 500), training set 3(5000 by 10000), training set 4(500 by 500), training set 5(1000 by 1000) and training set 6(5000 by 1000) are listed in the table 1.

Due to the bad performance of my laptop, the results do not seem so good. But based on the results in the table, it can be seen that the running time is reduced significantly by the improvement added every time. Especially the subroutine ssyrk() of openBLAS, it reduces much running time of updating matrix M and R instead of using three-nested for-loops.

Table 1: Comparison between different versions over running time(seconds)

	Traing set 2	Traing set 4	Traing set 5	Traing set 6	Traing set 3
first version	5	3.6	19	107	over 1000
second version	4.5	2.4	14.6	79	over 1000
third version	2.8	2	9.6	70	over 1000
fourth version	0.7	0.8	3	14	210

5 Discussion

Advantages of my approach:

- Using an openBLAS subroutine ssyrk() to update matrix instead of updating it manually, which improves the computational performance significantly.
- Using openMP to parallel the for-loops.
- Read one line of loci in every iteration so that the memory of the computer is big enough to store the information.
- Using dynamic array of float type instead of numeric matrix, which improves the computational performance.
- Using memset() function to initialize matrices instead of using nested for-loops.

About the disadvantages of my approach, first of all, I define the number of lines and individuals as constants at the beginning of the code, which means I need to know the exact number of lines and columns of the data before running it. Additionally, I think there must be some places of my code could be further improved, such as the structure of the code, the speed of reading GEN files, using some other techniques, etc.

6 Acknowledgement

I feel great to take this advanced course of statistical software tools since I learned a lot during the course.

Thanks to Christian Benner, I know how to use the integration of R and C++ with several great techniques to achieve high performance computing.

Thanks to Arto Nissinen, I solved a bug of my code and found a useful way to update my matrices.