frontiers in Genetics

ORIGINAL RESEARCH
published: 28 July 2021
doi: 10.3389/fgene.2021.655638

Alternative Ways of Computing the Numerator Relationship Matrix

Mohammad Ali Nilforooshan[1]*, *Dorian Garrick*[2] and *Bevin Harris*[1]

[1] Livestock Improvement Corporation, Hamilton, New Zealand, [2] AL Rae Centre for Genetics and Breeding, School of Agriculture, Massey University, Palmerston North, New Zealand

Pedigree relationships between every pair of individuals forms the elements of the additive genetic relationship matrix (**A**). Calculation of $\mathbf{A}^{-1}$ does not require forming and inverting **A**, and it is faster and easier than the calculation of **A**. Although $\mathbf{A}^{-1}$ is used in best linear unbiased prediction of genetic merit, **A** is used in population studies and post-evaluation procedures, such as breeding programs and controlling the rate of inbreeding. Three pedigrees with 20,000 animals (20K) and different (1, 2, 4) litter sizes, and a pedigree with 180,000 animals (180K) and litter size 2 were simulated. Aiming to reduce the computation time for calculating **A**, new methods [Array-Tabular method, $(\mathbf{T}^{-1})^{-1}$ instead of **T** in Thompson's method, iterative updating of **D** in Thompson's method, and iteration by generation] were developed and compared with some existing methods. The methods were coded in the R programming language to demonstrate the algorithms, aiming for minimizing the computational time. Among 20K, computational time decreased with increasing litter size for most of the methods. Methods deriving **A** from $\mathbf{A}^{-1}$ were relatively slow. The other methods were either using only pedigree information or both the pedigree and inbreeding coefficients. Calculating inbreeding coefficients was extremely fast (< 0.2 s for 180K). Parallel computing (15 cores) was adopted for methods that were based on solving $\mathbf{A}^{-1}$ for columns of **A**, as those methods allowed implicit parallelism. Optimizing the code for one of the earliest methods enabled **A** to be built in 13 s (faster than the 31 s for calculating $\mathbf{A}^{-1}$) for 20K and 17 min 3 s for 180K. Memory is a bottleneck for large pedigrees but attempts to reduce the memory usage increased the computational time. To reduce disk space usage, memory usage, and computational time, relationship coefficients of old animals in the pedigree can be archived and relationship coefficients for parents of the next generation can be saved in an external file for successive updates to the pedigree and the **A** matrix.

Keywords: pedigree, numeric relationship matrix, inverse, Cholesky decomposition, conjugate gradient, inbreeding coefficient, parallel computing

OPEN ACCESS

Edited by:
Steven J. Schrodi, University of Wisconsin-Madison, United States

Reviewed by:
Wei-Min Chen, University of Virginia, United States
Radovan Kasarda, Slovak University of Agriculture, Slovakia

***Correspondence:**
Mohammad Ali Nilforooshan
mohammad.nilforooshan@lic.co.nz

Specialty section:
This article was submitted to Statistical Genetics and Methodology, a section of the journal Frontiers in Genetics

Received: *19 January 2021*
Accepted: *17 June 2021*
Published: *28 July 2021*

Citation:
Nilforooshan MA, Garrick D and Harris B (2021) Alternative Ways of Computing the Numerator Relationship Matrix. Front. Genet. 12:655638. doi: 10.3389/fgene.2021.655638

1. INTRODUCTION

Co-ancestry or kinship is a fundamental characteristic in population and quantitative genetics. Co-ancestry is reflected by the coefficient of relationship, which is defined as the probability that two individuals share identical alleles by descent (Falconer and Mackay, 1996). The additive genetic relationship between two individuals is twice their co-ancestry. Wright (1922) computed relationships based on path coefficients (sum over all the paths of the products of the coefficients in a path that connects two individuals through common ancestors). Working with large pedigrees, it

is hard to keep track of all the paths and computationally efficient methods are required. Additive genetic relationships between individuals in a pedigree form the elements of the additive genetic relationship matrix, also called the numerator relationship matrix (**A**) and are the basis of the variance–covariance of breeding values for animals in the pedigree.

Since the development of best linear unbiased prediction (BLUP) methodology (Henderson et al., 1959) up to recent advanced methods for genomic evaluation (Fernando et al., 2016), the $\mathbf{A}^{-1}$ matrix has been a critical element used in computations for predicting genetic merit of individuals. It allows algorithms to exploit information on relatives for predicting genetic merit accounting for the genetic (co)variance structure among individuals in the population. More details about BLUP is given by Henderson (1975a), where it is shown how unbiasedness and minimum prediction error of predictors are met in populations under selection. Henderson (1976) and Quaas (1976) presented methods of directly deriving $\mathbf{A}^{-1}$ from the pedigree without the need to form **A**. Matrix $\mathbf{A}^{-1}$ has rows and columns corresponding to animals in the pedigree and it is very sparse. Many studies have been performed to improve the computational efficiency in constructing $\mathbf{A}^{-1}$ and in particular the diagonal elements of **A**, which are required for the construction of $\mathbf{A}^{-1}$ (Tier, 1990; Meuwissen and Luo, 1992; Colleau, 2002; Sargolzaei and Iwaisaki, 2005; Sargolzaei et al., 2005). However, there has been relatively little attention to the efficient calculation of off-diagonal elements of **A**. This is particularly because there is less demand for elements of **A** compared to elements of $\mathbf{A}^{-1}$. Also, the calculation of **A** is typically computationally more demanding. $\mathbf{A}^{-1}$ is sparser than **A** and calculations for each individual only rely on the individual and its parents. In **A**, the relationship coefficient between individuals i and j equals to the average of the relationship coefficients between the parents of j with i (0 for unknown parents). Although **A** is not needed for routine genetic evaluations, there is need for numerical values of some or all of its elements in population and gene flow studies, conservation programs, controlling the rate of inbreeding and in designing breeding programs. The aim of this study is to review existing methods and introduce new algorithms for calculating **A**. Runtime comparisons of the approaches are provided.

2. MATERIALS

Three pedigree structures (PED1, PED2, and PED3) with litter sizes of 1, 2, or 4 were simulated and tested. The pedigree simulation was performed using the R package "pedSimulate" (Nilforooshan, 2021) following the steps below:

1. Starting with 200 base population individuals, representing an equal number of foundation males and females.
2. Those foundation individuals were randomly mated (no selection or pre-mating mortality) to produce the next generation.
3. Pre-mating mortality rate was 3% in subsequent generations.
4. There was no overlap of generations for females, but there was one generation overlap for males.
5. Note that 80% of mature females and 20% of mature males were selected as parents of the next generation, then randomly mated.
6. Data were simulated generation after generation until reaching a population size of 20,000 individuals.
7. After the pedigree was simulated, 10% of dams and 20% of sires were randomly set to missing in non-base generations.

Another pedigree structure (PED4) was simulated similar to PED2, but with a population size of 180,000 individuals. PED4 was specifically created to be tested on the fastest methods based on the results from the three other pedigrees, and to find out about computational limitations and hardware requirements for a larger pedigree. The pedigree files are provided in the data repository (Data Availability Statement).

R statistical software (R Core Team, 2019) was used throughout this study. The R package "pedigreemm" (Vazquez et al., 2010) was used for calculating inbreeding coefficients and the $\mathbf{A}^{-1}$ matrix, and R packages "doParallel" (Wallig et al., 2020b) and "foreach" (Wallig et al., 2020a) were used for parallel processing, when the main task could be split into independent threads. Libraries LAPACK and SuiteSparse were exploited by using the R package "Matrix" (Bates and Maechler, 2019) for operations on sparse matrices. Data, Shell scripts, R code, and R functions used in this study are available in the data repository (Data Availability Statement).

Computations and benchmarking were performed on an Ubuntu server 20.04 LTS Amazon machine image. A few types of cloud machines from the R5, C5, and T2 families of the Amazon elastic compute cloud were used for the computations. Specifications of the cloud machine families are given in the **Appendix**, and specifications of each machine (number of virtual CPUs and read-only memory (RAM) capacity) are given in section 5.

3. METHODS

3.1. Tabular Method

The history of the tabular method goes back to early works done by Emik and Terrill (1949) and Cruden (1949). This method is based on simple rules. First, individuals are ordered in the pedigree to ensure parents appear before their progeny. Considering s and d being sire and dam of j, and individual i appeared in the pedigree before j, $A_{ij} = (A_{si} + A_{di})/2$. If a parent (e.g., s) is unknown, $A_{si} = 0$. If both parents s and d are known, $A_{jj} = 1 + F_j = 1 + A_{sd}/2$. The original method loops over rows, and columns within a row, to sequentially calculate matrix elements. Because **A** is symmetric, $n(n+1)/2$ of n^2 elements need to be calculated. This method is coded in function `buildA` of R package "ggroups" (Nilforooshan and Saavedra-Jiménez, 2020). Using array programming techniques (vectorized calculations), we optimized this method (Array-Tabular or method 3.1.2) to loop only over columns, which is expected to speed up the calculations. The jth array is calculated from the sth (for known s) and the dth array (for known d). This method is coded in R function `tabularA` (Data Availability Statement), which requires a pedigree data frame as input.

3.2. Thompson's Method

Applying the LDL factorization, $\mathbf{A} = \mathbf{TDT}'$, where $\mathbf{T}$ is a lower triangle matrix, and $\mathbf{D}$ is a diagonal matrix. In this method (Thompson, 1977), $\mathbf{T}$ and $\mathbf{D}$ matrices required to calculate $\mathbf{A}$ are derived indirectly (*i.e.*, no direct factorization or decomposition). For individual j, off-diagonal elements that were initialized with 0 are changed to $T_{ij} = (T_{si} + T_{di})/2$ if both parents s and d are known, and to $T_{ij} = T_{si}/2$ if one parent (e.g., s) is known. The diagonal elements of $\mathbf{D}$ that were initiated with 1 change to $D_{jj} = (2 - F_s - F_d)/4$ if both parents are known, and to $D_{jj} = (3 - F_s)/4$ if one parent (e.g., s) is known. This method is coded in R function `TDTp` (Data Availability Statement). This function requires the pedigree data frame and a corresponding vector of inbreeding coefficients as input.

Alternatively, $(\mathbf{T}^{-1})^{-1}$ may be used instead of $\mathbf{T}$ (method 3.2.2), where $\mathbf{T}^{-1} = \mathbf{I} - \mathbf{J}$ and $\mathbf{J}$ is the lower triangle parent incidence matrix, with coefficients 0 and 0.5. R function `TDTp2` (Data Availability Statement) calculates $\mathbf{TDT}'$ via $\mathbf{T}^{-1}$.

3.3. Iterative Updating of D in Thompson's Method

Using Thompson's Method, $\mathbf{A} = \mathbf{TDT}'$. The initial $\mathbf{D}$ ($\mathbf{D}_0$) is calculated ignoring inbreeding coefficients in the population ($D_{jj} = 1 - p_j/4$, where p_j is the number of known parents for individual j). Using sire and dam design matrices ($\mathbf{J}_s$ and $\mathbf{J}_d$) with coefficients 0 and 0.5, matrix $\mathbf{D}$ and consequently $\mathbf{A}$ are updated in each iteration until diag($\mathbf{A}$) remains unchanged.

In iteration i, $\mathbf{A}_i = \mathbf{TD}_i\mathbf{T}'$, diag($\mathbf{A}_{i+1}$) and $\mathbf{D}_{i+1}$ are calculated. Note that diag($\mathbf{A}_{i+1}$) is calculated prior to calculating off-diagonal elements of $\mathbf{A}_{i+1}$, where $\text{diag}(\mathbf{A}_{i+1}) = 1 + 2 \times \text{diag}(\mathbf{J}_s\mathbf{A}_i\mathbf{J}'_d)$. For individual j with sire s and dam d that meets the three conditions of (1) $A_{jj_i} > \text{diag}(\mathbf{A}_{i+1})_j$, (2) $A_{ss_i} = \text{diag}(\mathbf{A}_{i+1})_s$ and (3) $A_{dd_i} = \text{diag}(\mathbf{A}_{i+1})_d$, $D_{jj_{(i+1)}}$ is calculated as $1 - (A_{ss_i} + A_{dd_i})/4$. If a parent (e.g., s) is unknown, then $A_{ss_i} = 0$. The iterations continue until there is no individual j left with $A_{jj_i} > \text{diag}(\mathbf{A}_{i+1})_j + \epsilon$, where ϵ is a small positive (tolerance) value, which was set to 1e–5 in this study. This method is coded in R function `AiterD` (Data Availability Statement), and it requires the pedigree data frame and the tolerance value as input.

3.4. Henderson's Method

Using the concepts of Cholesky decomposition, $\mathbf{A}$ can be written as $\mathbf{A} = \mathbf{LL}'$, where $\mathbf{L}$ is a lower triangle matrix with non-zero diagonal elements. Henderson (1975b) introduced a method for deriving $\mathbf{L}$ indirectly to calculate $\mathbf{A}$. If the definition of s and d change from sire and dam of j to parents of j, where $s < d$, calculation of $\mathbf{L}$ can be formulated as:

1. $\mathbf{L} = \mathbf{I}$
2. If only one parent (e.g., s) is known:
   a. $L_{ij|i \le s} = L_{si}/2$
   b. $L_{jj} = \left(1 - \sum_{i=1}^{s} L_{ij}^2\right)^{\frac{1}{2}}$
3. If both parents are known:
   a. $L_{ij|i \le s} = (L_{si} + L_{di})/2$
   b. $L_{ij|s<i \le d} = L_{di}/2$
   c. $L_{jj} = \left(1 + \frac{1}{2}\sum_{i=1}^{s} L_{si}L_{di} - \sum_{i=1}^{s} L_{ij}^2\right)^{\frac{1}{2}}$

Calculation of $\mathbf{L}'$ is coded in R function `getLp` (Data Availability Statement), and it requires a pedigree data frame as input.

3.5. Iteration by Generation

The previous methods iterate over individuals with at least one known parent. Number of loops can increase the computational time. Thus, a method that minimizes loops (e.g., generations rather than individuals) could be beneficial. In this method, the $\mathbf{A}$ matrix is successively enlarged and completed generation by generation. The base (0) generation, irrespective of the date of birth, includes individuals with both parents unknown, and $\mathbf{A}_0 = \mathbf{I}$. Individuals in generation i ($i > 0$) have a known parent in generation i – 1 and the other parent either unknown or in generation $< i$. For generation i,

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{I} \\ \mathbf{J}_i \end{bmatrix} \mathbf{A}_{i-1} \begin{bmatrix} \mathbf{I} \\ \mathbf{J}_i \end{bmatrix}' + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \text{diag}(\mathbf{o}_i) \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{A}_{i-1} & \mathbf{A}_{i-1}\mathbf{J}'_i \\ \mathbf{J}_i\mathbf{A}_{i-1} & \mathbf{J}_i\mathbf{A}_{i-1}\mathbf{J}'_i + \text{diag}(\mathbf{o}_i) \end{bmatrix},$$

where $\mathbf{o}_i = 1 + 2 \times \text{diag}(\mathbf{J}_{s_i}\mathbf{A}_{i-1}\mathbf{J}'_{d_i}) - \text{diag}(\mathbf{J}_i\mathbf{A}_{i-1}\mathbf{J}'_i)$, $\mathbf{J}_i$ is the parent incidence matrix with elements of 0.5 and 0, rows corresponding to individuals in generation i and columns corresponding to individuals in generations $< i$. $\mathbf{J}_{s_i}$ and $\mathbf{J}_{d_i}$ are sire and dam incidence matrices with elements of 0.5 and 0, and $\mathbf{J}_i = \mathbf{J}_{s_i} + \mathbf{J}_{d_i}$. If the definition of s and d change from sire and dam to parents, where $s < d$, $\mathbf{A}_{i-1}$ can be replaced with its upper triangle without diagonal elements.

For example, consider the following pedigree with columns corresponding to individual, sire and dam ID:

```
1  0  0
2  0  0
3  1  0
4  1  2
5  3  4
6  1  4
7  5  6
```

Individuals 1 and 2 are assigned to the base generation, individuals 3 and 4 to generation 1, individuals 5 and 6 to generation 2, and individual 7 to generation 3. The matrices involved in the calculation of $\mathbf{A}$ are presented below:

$$\mathbf{A}_0 = \mathbf{I}_2,$$
$$\mathbf{J}_{s_1} = \begin{bmatrix} 0.5 & 0 \\ 0.5 & 0 \end{bmatrix}, \mathbf{J}_{d_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0.5 \end{bmatrix},$$
$$\mathbf{J}_{s_2} = \begin{bmatrix} 0 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 \end{bmatrix}, \mathbf{J}_{d_2} = \begin{bmatrix} 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix},$$
$$\mathbf{J}_{s_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.5 & 0 \end{bmatrix}, \mathbf{J}_{d_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

This method is coded in R function `iterGen` (Data Availability Statement), and it requires a pedigree data frame as input. This function is written in a way that there is no need to form $\mathbf{J}_{s_i}$, $\mathbf{J}_{d_i}$ or $\mathbf{J}_i$, nor to do any matrix multiplications.

3.6. Direct Solving of $\mathbf{A}^{-1}$

Calculation of $\mathbf{A}^{-1}$ is typically computationally less extensive than the calculation of $\mathbf{A}$, and there are fast algorithms for computing inbreeding coefficients (Tier, 1990; Meuwissen and Luo, 1992; Colleau, 2002; Sargolzaei and Iwaisaki, 2005; Sargolzaei et al., 2005), which are required for the derivation of $\mathbf{A}^{-1}$. One way of obtaining $\mathbf{A}$ is via solving equations involving $\mathbf{A}^{-1}$. In this approach, $\mathbf{A}^{-1}$ was directly solved to form $\mathbf{A}$, using the `solve` function in R.

3.7. Solving $\mathbf{A}^{-1}$ for Columns of A

Evidently, $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. This method is based on solving the equation system $\mathbf{A}^{-1}\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ is the ith column of $\mathbf{A}$ and $\mathbf{b}$ is the vector of $\mathbf{0}$ with the ith element (corresponding to the ith column of $\mathbf{A}$) set to 1. This method is coded in R function `solveAi` (Data Availability Statement), and it requires matrix $\mathbf{A}^{-1}$ as input.

3.8. Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of A

This method is similar to Solving $\mathbf{A}^{-1}$ for Columns of $\mathbf{A}$, but the pre-conditioned conjugate gradient (PCG) algorithm is used to solve the equation system. In PCG, a pre-conditioner matrix ($\mathbf{M}^{-1}$) is used for conditioning the linear equation system, and it exploits the symmetric positive definite structure of the coefficient matrix. The PCG algorithm is coded in R function `pcg`. Function `pcgAi` is a wrapper of function `pcg` over the columns of $\mathbf{A}$. Both functions are available in the data repository (Data Availability Statement). For each column of $\mathbf{A}$, the equation system $\mathbf{M}^{-1}\mathbf{A}^{-1}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ is iteratively solved, where $\mathbf{x}$ is the ith column of $\mathbf{A}$. Vector $\mathbf{b}$ (as defined in Solving $\mathbf{A}^{-1}$ for Columns of $\mathbf{A}$) was used as prior for $\mathbf{x}$. One may use previously solved columns of $\mathbf{A}$ as prior for the current $\mathbf{x}$ by copying the previously solved columns of $\mathbf{A}$ to the corresponding rows. However, it creates dependency between solving each column of $\mathbf{A}$ with its previous columns. In order to take advantage of parallel processing, we chose not to use this option. The PCG convergence criterion was set to 1e–5. We assumed that the resulting $\mathbf{A}$ matrix is symmetric. If not, there would be additional computational cost for making it symmetric [i.e., $\mathbf{A} = (\mathbf{A} + \mathbf{A}')/2$]. Matrices $\mathbf{A}^{-1}$ and $\mathbf{M}^{-1}$ are input to function `pcgAi`. If matrix $\mathbf{M}^{-1}$ is not defined with the argument `Minv`, or `Minv = NULL`, the conjugate gradient method is used instead of PCG, which makes no use of a pre-conditioner matrix. A few alternatives were tested as $\mathbf{M}^{-1}$:

3.8.1. $\mathbf{M}^{-1} = \text{diag}(1/\text{diag}(\mathbf{A}^{-1}))$

3.8.2. $\mathbf{M}^{-1} = \mathbf{D}$

Matrix $\mathbf{D}$ is as defined in Thompson's Method. R function `getD` (Data Availability Statement) calculates diag($\mathbf{D}$) from the pedigree data frame and the corresponding vector of inbreeding coefficients.

3.8.3. $\mathbf{M}^{-1} = \mathbf{D}_0$

Matrix $\mathbf{D}_0$ is as defined in Iterative Updating of $\mathbf{D}$ in Thompson's Method. R function `getD0` (Data Availability Statement) calculates diag($\mathbf{D}_0$) from the pedigree data frame.

4. METHOD FEATURES

4.1. Tabular Method

The tabular method is straightforward based on very simple rules. The *cons* of this method are as follows: (1) It loops over rows of $\mathbf{A}$ and over columns 1 to i in row i. Loops can be skipped for animals with both parents unknown. There is an additional cost of calculating an inbreeding coefficient in every loop on a row, if both parents of the animal are known. (2) Due to dependencies (progeny on parents) between calculations, computations cannot easily be parallelized. (3) Because the method involves updating $\mathbf{A}$ in every loop, the method could not take advantage of libraries for sparse matrices. Such libraries reduce memory usage and computational complexity of matrix operations.

We used R package "Matrix" (Bates and Maechler, 2019) for sparse matrices. However, iteratively forming or updating elements of a matrix initially built in a sparse format slowed down the procedure considerably. We think a possible reason was that every time the matrix was getting updated the memory had to be re-allocated. Therefore, we used R package "Matrix" (equipped with libraries LAPACK and SuiteSparse) for whole matrix operations rather than for updating elements of the matrix or appending rows/columns to it.

The benefit of the Array-Tabular Method over the Tabular Method was reducing the number of loops with replacing inner (on columns of $\mathbf{A}$) loops with vectorized operations. Here, we chose to loop over columns than rows, because operations on columns are faster than operation on rows in modern programming languages such as R and Julia.

4.2. Thompson's Method

Matrix $\mathbf{T}$ and vector $\mathbf{d}$ ($\mathbf{d}$ = diag($\mathbf{D}$)) had to be created iteratively with an iteration over every animal with a known parent. As a result, the libraries for sparse matrices (LAPACK and SuiteSparse) were not used at this (updating) stage. The cost of making $\mathbf{T}$ and $\mathbf{d}$ was slightly lower than the cost of directly making $\mathbf{A}$ via the Tabular Method. The main benefit of this method was that after creating $\mathbf{T}$ and $\mathbf{d}$, it was possible to take advantage of libraries for sparse matrices. Matrix $\mathbf{T}$ was converted to a sparse format and vector $\mathbf{d}$ was converted to the diagonal sparsely formatted $\mathbf{D}$. Similar to the Array-Tabular Method, inner loops were replaced by vectorized operations for making $\mathbf{T}$. There were additional costs involved with storing $\mathbf{T}$ and $\mathbf{D}$ in the memory and matrix multiplications ($\mathbf{TDT}'$), however those were considerably reduced by making use of the libraries for sparse matrices (LAPACK and SuiteSparse).

We tested $(\mathbf{T}^{-1})^{-1}$ instead of $\mathbf{T}$ and the reason was that $\mathbf{T}^{-1}$ is much sparser than $\mathbf{T}$ (which has off-diagonal elements of -0.5 in each row for each known parent of the individual and a diagonal element of 1), but it can be built immediately, because there are no dependencies between rows of $\mathbf{T}^{-1}$. On the other hand, there is a computational cost involved for inverting the lower triangle sparse matrix $\mathbf{T}^{-1}$.

4.3. Iterative Updating of D in Thompson's Method

The aim of this method was to remove the dependency on inbreeding coefficients that existed in Thompson's Method. The one-off computational cost for the calculation of $\mathbf{D}$ is replaced with an iterative procedure for updating $\mathbf{D}_i$ in iteration i, starting with $\mathbf{D}_0$ with 3 possible values depending on the number of known parents of the individual. There are matrix multiplication costs ($\mathbf{TD}_i\mathbf{T}'$) involved in each iteration. The number of iterations to update and stabilize $\mathbf{D}_i$ is much lower than the number of iterations to form $\mathbf{D}$ from inbreeding coefficients. However, given the cost of updating $\mathbf{D}_i$, this method might only be justifiable over Thompson's Method if the inbreeding coefficients are not available or the number of iterations to update $\mathbf{D}_i$ is very low. Using $(\mathbf{T}^{-1})^{-1}$ rather than $\mathbf{T}$, this method does not require ordering animals in the pedigree.

4.4. Henderson's Method

Compared to Thompson's Method, this method requires less matrix multiplications ($\mathbf{LL}'$ vs. $\mathbf{TDT}'$). However, calculating $\mathbf{L}$ is computationally more demanding than calculating $\mathbf{T}$. Matrices $\mathbf{L}$ and $\mathbf{T}$ have the same sparsity. Matrix $\mathbf{L}$ is created iteratively, and then sparsely formatted to reduce the matrix multiplication cost using the libraries for sparse matrices, provided by R package "Matrix."

4.5. Iteration by Generation

This method is independent from inbreeding coefficients in the population. Most of the previous methods iterate over rows of $\mathbf{A}$ and even columns 1 to i within row i (e.g., Tabular Method). This method on the other hand iterates over generations. In each generation i, a parent incidence matrix ($\mathbf{J}_i$) needs to be created and there are matrix multiplications and summations involved. However, in the code, these matrix operations were omitted by detecting parents of the current generation and looping over parents within the generation. As a result, the number of loop iterations are reduced considerably compared to the methods looping over individuals, especially if there are many progeny per parent per generation. Calculations for progeny per parent per generation were done using vectorized operations. Due to matrix updating, libraries for sparse matrices were not used.

4.6. Direct Solving of $\mathbf{A}^{-1}$

Calculation of $\mathbf{A}^{-1}$ is typically very fast and computationally less demanding (operations and memory usage) than the calculation of $\mathbf{A}$. Solving $\mathbf{A}$ from $\mathbf{A}^{-1}$ does not involve matrix updating and $\mathbf{A}^{-1}$ is very sparse. Therefore, there is great opportunity in using libraries for sparse matrices. Another advantage in using $\mathbf{A}^{-1}$ to derive $\mathbf{A}$ is the lack of calculation dependencies. As a result, the solving procedure can be done at once and in parallel. The first method of solving $\mathbf{A}$ from $\mathbf{A}^{-1}$ is of course direct solving of $\mathbf{A}^{-1}$. Explicit parallelism was an option. However, in this study we limited the use of parallel processing to implicit parallelism.

4.7. Solving $\mathbf{A}^{-1}$ for Columns of A

Quantitative fields of science deal with solving systems of linear equations. Equation systems can be presented in a matrix form, where the coefficient matrix of predictions (here, $\mathbf{A}^{-1}$) multiplied by the vector of predictions (here, the ith column of $\mathbf{A}$) equals to the right-hand-side vector (here, a vector of zeros with the ith element equal to 1). The vector of predictions is to be predicted by solving the equation system. Solving for columns of $\mathbf{A}$ was parallelized implicitly. Though, solving each column of $\mathbf{A}$ could be parallelized explicitly, it had no justification over explicit parallelism of Direct Solving of $\mathbf{A}^{-1}$. For this method, we took advantage of the libraries for sparse matrices via R package "Matrix."

TABLE 1 | Description of the simulated pedigrees.

Number of	PED1	PED2	PED3	PED4
Individuals	20,000	20,000	20,000	180,000
Litter size	1	2	4	2
Generations	15	9	6	12
Sires	3,507	2,084	1,449	21,985
Dams	6,271	5,012	4,176	51,890
Missing sires	4,160	4,160	4,160	36,160
Missing dams	2,180	2,180	2,180	18,180
Missing both parents	582	595	630	3,808
Inbred individuals	3,240	2,025	930	62,449

4.8. Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of A

The iterative PCG algorithm is commonly used in quantitative genetics to solve mixed model equations. It is preferable for solving large and sparse equation systems, where direct solving of the coefficient matrix is not feasible. Pre-conditioner matrices $\text{diag}(1/\text{diag}(\mathbf{A}^{-1}))$, $\mathbf{D}$ and $\mathbf{D}_0$ were tested. This method took advantage of implicit parallel processing for independently solving columns of $\mathbf{A}$ and libraries for sparse matrices.

5. RESULTS

5.1. Simulated Pedigrees

Table 1 provides information about the simulated pedigrees. With a constant pedigree size (PED1, PED2, and PED3), the number of generations and the total number of parents decreased with increasing litter size. The number of inbred individuals decreased with increasing litter size (**Table 1**) since fewer generations were represented. However, the average inbreeding coefficients among inbred individuals increased with litter size and were 0.008, 0.013, and 0.030 for PED1, PED2, and PED3, respectively. The maximum inbreeding coefficient was 0.25 for all the pedigrees.

5.2. Pedigrees With 20,000 Animals

Table 2 shows elapsed time for the calculation of $\mathbf{A}$, using different methods for PED1, PED2, and PED3. The elapsed times presented in **Table 2** were recorded on an "r5.large" cloud machine with 2 cores, and 16 Gb of RAM for the methods that were making no use of parallel processing, and a "c5.4xlarge" cloud machine with 16 cores and 32 Gb of RAM for the methods

TABLE 2 | Elapsed time (MM:SS) for the calculation of **A** matrix using different methods for different pedigree.

Method	PED1	PED2	PED3
3.1.1: Tabular Method	01:39	01:37	01:36
3.1.2: Array-Tabular Method	00:13	00:13	00:13
3.2.1: Thompson's Method	00:47	00:41	00:34
3.2.2: $(\mathbf{T}^{-1})^{-1}$ instead of **T** in Thompson's Method	01:03	00:58	00:51
3.3: Iterative Updating of **D** in Thompson's Method	04:11	02:05	01:01
3.4: Henderson's Method	00:42	00:37	00:32
3.5: Iteration by Generation	02:13	01:13	00:45
3.6: Direct Solving of $\mathbf{A}^{-1}$	04:08	02:32	01:43
3.7: Solving $\mathbf{A}^{-1}$ for Columns of $\mathbf{A}^{p}$	07:31	05:53	05:35
3.8: Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of $\mathbf{A}^{p}$			
3.8.1: $\mathbf{M}^{-1} = \text{diag}(1/\text{diag}(\mathbf{A}^{-1}))$	11:42	12:10	11:48
3.8.2: $\mathbf{M}^{-1} = \mathbf{D}$	15:13	16:42	18:45
3.8.3: $\mathbf{M}^{-1} = \mathbf{D}_0$	15:25	17:08	18:53

p, parallel processing using 15 computational cores; PED1, The pedigree of 20,000 individuals and litter size of 1; PED2, The pedigree of 20,000 individuals and litter size of 2; PED3, The pedigree of 20,000 individuals and litter size of 4; $\mathbf{M}^{-1}$ *is the pre-conditioner matrix for the pre-conditioned conjugate gradient algorithm;* **D** *is a diagonal matrix introduced in Thompson's Method;* $\mathbf{D}_0$ *is the starting* **D** *in Iterative Updating of* **D** *in Thompson's Method.*

that were making use of parallel processing (3.7 and 3.8). The new methods (3.1.2, 3.2.2, 3.3, and 3.5) were among the 7 (of 12) fastest methods. The five fastest methods were 3.1.2 being the fastest to 3.4, 3.2.1, 3.2.2, and 3.5. Except for the methods 3.8.1, 3.8.2, and 3.8.3, the runtime decreased from PED1 to PED3, as a result of a greater litter size, less number of generations and less number of parents in the same population size.

Compared to Thompson's Method, Iterative Updating of **D** in Thompson's Method was independent from inbreeding coefficients being known in the population. However, using fast algorithms for calculating inbreeding coefficients (Tier, 1990; Meuwissen and Luo, 1992; Colleau, 2002; Sargolzaei and Iwaisaki, 2005; Sargolzaei et al., 2005), these coefficients can be computed in a short time for large populations. Calculation of inbreeding coefficients took less than 0.01 s for PED1, PED2, and PED3, shorter with less number of generations (larger litter sizes). Consequently, calculating **D** based on known inbreeding coefficients is faster than starting with $\mathbf{D}_0$ with no knowledge on inbreeding coefficients in the population and then iteratively converging to **D**.

Calculation of $\mathbf{A}^{-1}$ was fast and efficient. It took 31.4, 31.3, and 31.2 s for PED1, PED2, and PED3 to calculate $\mathbf{A}^{-1}$. Unlike **A**, $\mathbf{A}^{-1}$ is sparse and calculations for each individual involves the individual and its parents rather than information on all relatives. Consequently, a possibility for the calculation of **A** is via $\mathbf{A}^{-1}$. Off-diagonal elements of $\mathbf{A}^{-1}$ showed 99.98% sparsity for all the pedigrees. On the other hand, off-diagonal elements of **A** showed 58.04%, 70.51%, and 85.14% sparsity for PED1, PED2, and PED3, respectively. The average of non-zero off-diagonal elements of **A** was 0.007, 0.011, and 0.023, and the maximum relatedness was 0.750, 0.750, and 0.812 for PED1, PED2, and PED3, respectively. Direct Solving of $\mathbf{A}^{-1}$ was faster than Solving $\mathbf{A}^{-1}$ for Columns of **A**, and Solving $\mathbf{A}^{-1}$ for Columns of **A** was faster than Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of **A**.

We made use of the parallel version of LAPACK, ScaLAPACK library for high-performance linear algebra routines for parallel distributed memory machines to do Solving $\mathbf{A}^{-1}$ for Columns of **A** on 15 cores of an "r5.4xlarge" machine with 16 cores and 128 Gb of RAM. However, the procedure was crushed due to the high memory usage.

Three diagonal pre-conditioner ($\mathbf{M}^{-1}$) matrices were tested for Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of **A**. We did not find any notable difference between **D** and $\mathbf{D}_0$ used as $\mathbf{M}^{-1}$ in time to calculate **A**. On the other hand, diag(1/diag($\mathbf{A}^{-1}$)) was a better $\mathbf{M}^{-1}$ and it further reduced the elapsed time considerably (**Table 2**). The differences between the elapsed time of the PCG-based methods were mainly due to the convergence of the PCG algorithm rather than the elapsed time to form $\mathbf{M}^{-1}$. Forming diag(1/diag($\mathbf{A}^{-1}$)) and $\mathbf{D}_0$ each took 0.02 s, and it took 1 s to form **D** for PED1, PED2, and PED3. Block pre-conditioner matrices $\mathbf{TT}'$ and $\mathbf{TD}_0\mathbf{T}'$ were also tested (results not shown). Besides being computationally more expensive to build, those matrices considerably increased the computational time likely due to increased matrix multiplication costs per PCG iteration per column of **A**.

Root of mean squared error (RMSE, excluding the upper diagonal elements of **A**) for the methods of Solving $\mathbf{A}^{-1}$ for Columns of **A** and Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of **A** are presented in **Table 3**. Even with the low strict PCG convergence criterion of 1e–5, RMSE values were very close to zero for most of the methods. The method of Solving $\mathbf{A}^{-1}$ for Columns of **A** produced considerably lower RMSE values than the method of Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of **A**. There was no difference between the PCG methods using **D** and $\mathbf{D}_0$ as $\mathbf{M}^{-1}$. Using diag(1/diag($\mathbf{A}^{-1}$)) as $\mathbf{M}^{-1}$ produced the highest RMSE values.

Wherever possible, libraries for sparse matrices should be used to reduce the memory usage and computational complexity for sparse matrices. **Table 4** shows the memory usage by matrices **A**, $\mathbf{A}^{-1}$, **T**, $\mathbf{T}^{-1}$, **L**, and **D** in sparse format. Those 20,000 × 20,000 matrices occupied more than 32e+8 bytes of memory when not stored in sparse format.

5.3. Pedigree With 180,000 Animals

The Array-Tabular Method and the non-optimized Tabular Method (loops over rows and columns within each row) were tested on a larger pedigree with 180,000 animals (PED4), and it took them 17 min 3 s and 2 h 20 min 34 s time to calculate **A**, respectively. Calculating **A** for such a pedigree required a much larger memory than for a pedigree with 20,000 animals. Therefore, an "r5.8xlarge" cloud machine was used, which has 32 cores and 256 Gb of RAM. Inbreeding coefficients were calculated in 0.13 s. Other methods were not tested for PED4, either because those performed slow for the pedigrees with 20,000 animals or they failed due to the shortage of memory. Calculation of $\mathbf{T}^{-1}$, $\mathbf{T}'$, $\mathbf{L}'$, **D**, and $\mathbf{D}_0$ took 2 min 16 s, 4 min 8 s, 5 min 2 s, 9.4 s, and 0.01 s, respectively. However, there was a shortage of memory for matrix multiplications to derive **A**. Also, the calculation of $\mathbf{A}^{-1}$

TABLE 3 | Root of mean squared error for the lower triangle elements of **A** for the methods of solving $\mathbf{A}^{-1}$ and $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for columns of **A**, where $\mathbf{M}^{-1}$ is the pre-conditioner matrix for the pre-conditioned conjugate gradient algorithm.

Method	PED1	PED2	PED3
3.7: Solving $\mathbf{A}^{-1}$ for Columns of **A**	4e–17	5e–17	4e–17
3.8: Iterative Solving $\mathbf{M}^{-1}\mathbf{A}^{-1}$ for Columns of **A**			
3.8.1: $\mathbf{M}^{-1} = \text{diag}(1/\text{diag}(\mathbf{A}^{-1}))$	1.2e–4	1.5e–4	1.7e–4
3.8.2: $\mathbf{M}^{-1} = \mathbf{D}$	7.5e–5	9e–5	1.2e–4
3.8.3: $\mathbf{M}^{-1} = \mathbf{D}_0$	7.5e–5	9e–5	1.2e–4

*PED1, The pedigree of 20,000 individuals and litter size of 1; PED2, The pedigree of 20,000 individuals and litter size of 2; PED3, The pedigree of 20,000 individuals and litter size of 4; **D** is a diagonal matrix introduced in Thompson's Method; $\mathbf{D}_0$ is the starting **D** in Iterative Updating of **D** in Thompson's Method.*

TABLE 4 | Memory usage (bytes) of matrix **A** and other matrices used to calculate **A**, when stored in sparse format, using R package "Matrix."

Matrix	PED1	PED2	PED3
A	1,007,355,312	708,069,624	356,891,384
$\mathbf{A}^{-1}$	3,456,376	3,397,608	3,361,008
T	6,680,136	4,618,272	2,995,416
$\mathbf{T}^{-1}$	725,424	725,424	725,424
L	6,680,136	4,618,272	2,995,416
D	161,240	161,240	161,240

*PED1, The pedigree of 20,000 individuals and litter size of 1; PED2, The pedigree of 20,000 individuals and litter size of 2; PED3, The pedigree of 20,000 individuals and litter size of 4; **T** is a lower triangle matrix introduced in Thompson's Method; **D** is a diagonal matrix introduced in Thompson's Method; $\mathbf{D}_0$ is the starting **D** in Iterative Updating of **D** in Thompson's Method.*

(using function `getInv` from R package "pedigreemm") failed due to the shortage of memory.

6. DISCUSSION

Calculating matrix **A** is typically computationally more demanding than the calculation of $\mathbf{A}^{-1}$, because except for individuals missing both parents, all the coefficients are to be calculated. Various methods of calculating **A**, from pedigree or from $\mathbf{A}^{-1}$, with/without knowledge on inbreeding coefficients, and with/without the possibility of implicit (independent) parallelism were tested and developed. Availability of information on inbreeding coefficients was not an issue, as inbreeding coefficients for large populations can be calculated in a very short period of time. Although the calculation of $\mathbf{A}^{-1}$ is straightforward and fast, the methods based on solving **A** from $\mathbf{A}^{-1}$ were relatively slow. These methods are widely and commonly used for solving linear equation systems, such as generalized linear models and linear mixed models. However, unlike **A** and $\mathbf{A}^{-1}$, coefficient matrix (left-hand-side matrix) of those models is complex and does not have any known properties, except that it is symmetric and it should be positive definite. Making use of the known properties of **A**, it can be calculated directly from pedigree with no additional information needed (inbreeding coefficients are required for Thompson's Method).

Direct Solving of $\mathbf{A}^{-1}$ as a big single task was faster than Solving $\mathbf{A}^{-1}$ for Columns of **A** with 20,000 smaller tasks (for pedigrees with 20,000 individuals), despite those were distributed over 15 computational cores rather than a single core. Further increasing the number of computational cores for the methods using parallel processing reduces their computational time, but the price per computational unit increases. Presumably, the sparsity of $\mathbf{A}^{-1}$ justified Direct Solving of $\mathbf{A}^{-1}$ over Solving $\mathbf{A}^{-1}$ for Columns of **A**. In the case of this study, because the treads of split task were independent from each other, parallelism overhead was minimized. Increased parallelism overhead minimizes the benefit from increased number of computational cores. Depending on the size and sparsity of the matrix, there are situations where direct inversion is preferred over distributed (e.g., Solving $\mathbf{A}^{-1}$ for Columns of **A**) and iterative procedures.

6.1. Other Ways of Solving A From $\mathbf{A}^{-1}$

There are other ways of solving **A** from $\mathbf{A}^{-1}$, which were not covered in this study. Examples are the conjugate gradient algorithm (i.e., PCG with $\mathbf{M}^{-1} = \mathbf{I}$) and inverse from Cholesky (or QR) decomposition of $\mathbf{A}^{-1}$. Rather than direct inversion, a symmetric positive-definite matrix like $\mathbf{A}^{-1}$ can be inverted from its Cholesky decomposition. Matrix $\mathbf{A}^{-1}$ can be decomposed into a product of $\mathbf{A}^{-1} = \mathbf{U}'\mathbf{U}$, where **U** is an upper triangle matrix with non-zero diagonal elements. Then, **A** is calculated from **U**, ($\mathbf{A} = \mathbf{U}^{-1}(\mathbf{U}^{-1})'$). Function `chol2inv` from R package "Matrix" (Bates and Maechler, 2019) inverts a matrix from its Cholesky decomposition. Also, R function `chol` computes the Cholesky decomposition of a matrix. Computational cost for direct inversion of a matrix is n^3, where n is the size of the matrix. Computational cost of Cholesky decomposition of a matrix (calculating **U**) is approximately $(2n^3 + 3n^2)/6$ (van de Geijn, 2011), and inverting a triangle matrix has a computational cost of n^2. Although the computational cost of multiplying two $n \times n$ matrices is n^3, considering one matrix being lower triangle and the other being upper triangle, the matrix multiplication cost can be reduced to $\sum_{i=1}^{n} i^2$. Using sparse matrix libraries (e.g., SuiteSparse, BLAS, and LAPACK) for sparse matrices, the computational costs would be less than those mentioned above.

Henderson (1976) presented a simple method for calculating $\mathbf{A}^{-1}$ by deriving its decomposition indirectly ($\mathbf{A}^{-1} = (\mathbf{T}^{-1})'\mathbf{D}^{-1}\mathbf{T}^{-1}$), where **D** is as defined in Thompson's Method, and $\mathbf{T}^{-1} = \mathbf{I} - \mathbf{J}$, where **J** is a lower triangle parent incidence matrix with coefficients 0.5 and 0. Unfortunately, $\mathbf{U}_2 = (\mathbf{T}^{-1})'\mathbf{D}^{-\frac{1}{2}}$ cannot serve as an indirect Cholesky decomposition of $\mathbf{A}^{-1}$, because with $\mathbf{U}_1$ being the direct Cholesky decomposition of $\mathbf{A}^{-1}$, $\mathbf{A}^{-1} = \mathbf{U}_1'\mathbf{U}_1 = \mathbf{U}_2\mathbf{U}_2'$, $\mathbf{U}_1 \neq \mathbf{U}_2$, and $\mathbf{U}_1 \neq \mathbf{U}_2'$.

Another method is forward and backward substitution. This method consists of a backward substitution followed by a forward substitution, per column of **A**. In forward substitution, the equation system $\mathbf{Kc} = \mathbf{b}$ is solved, where **b** is a vector of **0** with the ith element (corresponding to the ith column of **A**) set to 1, and **K** is a lower triangle matrix with non-zero diagonal elements, from the Cholesky decomposition of $\mathbf{A}^{-1}$ (*i.e.*, $\mathbf{A}^{-1} = \mathbf{K}\mathbf{K}'$).

Then, in the backward substitution, the equation system $\mathbf{K}'\mathbf{x} = \mathbf{c}$ is solved, where $\mathbf{x}$ is the ith column of $\mathbf{A}$. This method is coded in R function `fbs` (Data Availability Statement), and it requires matrix $\mathbf{A}^{-1}$ as input. We tested this method on PED1, PED2, and PED3 (results not shown). The results were good in terms of accuracy (RMSE), but the performance was poor in terms of computational time.

Another algorithm is based on the decomposition of $\mathbf{A}^{-1}$ to $\mathbf{T}'^{-1}\mathbf{D}^{-1}\mathbf{T}^{-1}$ (Colleau, 2002). This algorithm is used to create the product of $\mathbf{A}$ with a vector $\mathbf{x}$, in linear time proportional to the number of animals. It can also be used to compute products of partitions of $\mathbf{A}$ multiplied by a vector (Misztal et al., 2009). For example, in genetic evaluations including pedigree and genomic information, the partition of $\mathbf{A}$ for genotyped animals is created by repeatedly applying the algorithm of Colleau (2002), where the multiplying vector is a vector of zeros containing a single 1 (Misztal et al., 2009), similar to vector $\mathbf{b}$ used in Solving $\mathbf{A}^{-1}$ for Columns of $\mathbf{A}$.

6.2. Calculating Numerator Relationships From Inbreeding Coefficients

Van Vleck (2007) introduced a method for computing numerator relationships between any pair of animals. This method is based on the fact that the relationship between two individuals is twice the inbreeding coefficient of their progeny. As many pairs of animals do not have progeny (including animals from the same sex), this method involves introducing dummy progeny to pedigree for pairs of animals, for which calculating relationship coefficients is of interest. We did not test the performance of this method. It is expected to be very efficient for a small subset of pedigree, but not for a whole large pedigree, mainly due to the very many dummy progeny to be appended to the pedigree.

6.3. Optimizations for Less Memory Usage

Memory usage and the required disk space for saving matrix $\mathbf{A}$ to an external file can be reduced by calculating $\mathbf{A}$ in a tabular-sparse (non-zero elements and their row and column indices) rather than a matrix format, which is implemented in function `tabA` from R package "ggroups" (Nilforooshan and Saavedra-Jiménez, 2020). Although this method is memory efficient, it is recommended when the memory is a bottleneck. It reduces the memory usage at the cost of increased computational time. However, nowadays, memory is a relatively cheap component compared to other hardware components. In a previous study (Nilforooshan and Saavedra-Jiménez, 2020), calculating $\mathbf{A}$ for a pedigree of 3,000 animals took 13 min 56 s vs. 2 s, time for `tabA` vs. `buildA` (Tabular Method). Where matrix $\mathbf{A}$ needs to be saved in tabular-sparse format (takes less disk space), it can be built using function `buildA` or function `tabularA` (Data Availability Statement), converted to tabular-sparse format using function `mat2tab` from R package "ggroups" (Nilforooshan and Saavedra-Jiménez, 2020) and written to an external file. Writing the file in a binary format can further reduce the disk space usage.

In order to create and store matrix $\mathbf{A}$ compactly, but in a dense format, it can be created and stored in a lower triangle packed storage format (https://en.wikipedia.org/wiki/Packed_storage_matrix). That means, the lower triangle of matrix $\mathbf{A}$ is created and stored row-wise in a vector, where element $A_{ij}(i > j)$ is located at the position $(i(i-1)/2) + j$ of the vector. This method is mainly adopted for storing symmetric dense matrices in an external file. For writing symmetric semi-sparse matrices like $\mathbf{A}$ in an external file, it has to be dense enough ($> 1/3$ of the upper/lower triangle elements) to choose this format over the tabular-sparse format. This method is coded in R function `getAvec` (Data Availability Statement). Calculating $\mathbf{A}$ using this method took less memory. However, the computational time increased to 24 min 6 s for PED4 on an "r5.4xlarge" machine with 128 Gb of RAM. Although the memory usage was reduced, 128 GB of RAM was still not enough and the procedure made use of a swap space of 12 Gb, which was partly responsible for the increased computational time. This method was tested for PED1, PED2, and PED3 on a "t2.medium" cloud machine with 2 cores and 4 Gb of RAM, and the calculation of $\mathbf{A}$ took 17 s for each pedigree. Repeating the same process on a "t2.small" machine with 1 core and 2 Gb of RAM, using a swap space for compensating the shortage of RAM, increased the elapsed time to 1 min 30 s, 1 min 5 s, and 1 min 6 s for PED1, PED2, and PED3, respectively. Therefore, using swap space for computations should be avoided as it noticeably slows down the procedure, unless only marginal excess memory above the provided RAM is required.

There might be situations, where the whole $\mathbf{A}$ is not needed, but a block of it. These situations may bring opportunities to reduce the computational complexity and memory demand. **Supplementary Table 1** provides examples for Thompson's Method and Henderson's Method to calculate different blocks of $\mathbf{A}$.

7. CONCLUDING REMARKS

Modifying the way of coding one of the oldest methods of calculating $\mathbf{A}$ (Tabular Method), this matrix was calculated in a shorter time than the time spent for calculating $\mathbf{A}^{-1}$. Computational time and memory usage are the main bottlenecks for calculating $\mathbf{A}$. There is a trade-off between these two. Fast computation can be achieved by providing sufficient memory, and memory bottleneck can be lifted if computational time is not a hard limit. Creating and storing matrix $\mathbf{A}$ in a dense vector (packed storage) or in the tabular-sparse format can reduce the memory requirement, but the computational time is compromised.

Opportunities to speed up the calculation $\mathbf{A}$ by parallel processing were limited to a few methods. For most of the methods, calculations are iterative and not independent from each other (progeny following parents and generation after generation). Explicit parallelism (parallelism of a concurrent computation) does not seem to be an efficient option, because calculation of $\mathbf{A}$ involves many small and simple computations rather than a few large and complicated computations, where explicit parallel processing might be helpful (e.g., direct inversion of $\mathbf{A}^{-1}$). Therefore, parallelism overhead (time required to coordinate parallel tasks among computational nodes) is expected to be higher than the gain from parallelism.

The choice of programming language influences the computational time. However, comparing methods, time rankings, and accuracy matter the most. Programming languages with slow loops penalize iterative procedures. There are various optimization techniques available that can be applied to different conditions and limitations. Some examples are as follows: (1) Iteratively creating matrices **T** and **L**, we could not take advantage of libraries for sparse matrices. Those libraries were used to create **A** after **T** or **L** became available. In the shortage of memory, those matrices can be built in tabular-sparse format (e.g., three growing vectors of matrix elements and their row/column indices, then making a sparse matrix at once given those vectors). (2) We used vectorized operations to calculate multiple elements in a matrix row simultaneously. In species with multiple progeny per mating, this can be extended to multiple rows for full-sib progeny. In the same manner, multiple elements of vector **d** can be calculated simultaneously. (3) To reduce memory demand and computational complexity, small elements (e.g., $<$ 1e−5) can be set to zero on the fly.

Finally, in the absence of pedigree corrections for animals that are already in the pedigree, matrix **A** can be updated for new animals added to the pedigree. Therefore, calculations are done only once for each individual. Evidently, for updating **A** for each new individual, only coefficients corresponding to its parents (if known) are to be retrieved. Similarly, for pedigree corrections, **A** is corrected from where the pedigree is corrected. If relationship coefficients are no longer needed for old/culled animals, only the partition of **A** for animals chosen to be the parents to the next generation can be kept and saved in an external file. Other relationship coefficients can be archived in a separate file. Besides saving disk space, computational time and memory usage for the upcoming update of the **A** matrix are reduced.

DATA AVAILABILITY STATEMENT

The data, scripts and functions that support the findings of this study are openly available in the figshare repository https://doi.org/10.6084/m9.figshare.13497939.

AUTHOR CONTRIBUTIONS

MN initiated the idea of the study, wrote the programs, run the analyses, drafted the manuscript, and revised it. DG helped MN with more ideas to test and revising the manuscript. BH was the project leader and he was involved in the revision of the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This study received financial support from the NZ Ministry of Primary Industries, SFF Futures Programme: Resilient Dairy–Innovative breeding for a sustainable dairy future [Grant number: PGP06-17006].

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fgene.2021.655638/full#supplementary-material

REFERENCES

Bates, D., and Maechler, M. (2019). *Matrix: Sparse and Dense Matrix Classes and Methods. Version 1.2-18*. Available online at: https://cran.r-project.org/package=Matrix (accessed January 18, 2021).

Colleau, J. J. (2002). An indirect approach to the extensive calculation of relationship coefficients. *Genet. Select. Evol.* 34, 409–421. doi: 10.1186/1297-9686-34-4-409

Cruden, D. (1949). The computation of inbreeding coefficients for closed populations. *J. Heredity* 40, 248–251. doi: 10.1093/oxfordjournals.jhered.a106039

Emik, L. O., and Terrill, C. E. (1949). Systematic procedures for calculating inbreeding coefficients. *J. Heredity* 40, 51–55. doi: 10.1093/oxfordjournals.jhered.a105986

Falconer, D. S., and Mackay, T. F. C. (1996). *Introduction to Quantitative Genetics, 4th Edn*. London: Longman.

Fernando, R. L., Cheng, H., Golden, B. L., and Garrick, D. J. (2016). Computational strategies for alternative single-step bayesian regression models with large numbers of genotyped and non-genotyped animals. *Genet. Select. Evol.* 48:96. doi: 10.1186/s12711-016-0273-2

Henderson, C. R. (1975a). Best linear unbiased estimation and prediction under a selection model. *Biometrics* 31, 423–447. doi: 10.2307/2529430

Henderson, C. R. (1975b). Rapid method for computing the inverse of a relationship matrix. *Biometrics* 58, 1727–1730. doi: 10.3168/jds.S0022-0302(75)84776-X

Henderson, C. R. (1976). A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* 32, 69–83. doi: 10.2307/2529339

Henderson, C. R., Kempthorne, O., Searle, S. R., and von Krosigk, C. M. (1959). The estimation of environmental and genetic trends from records subject to culling. *Biometrics* 15, 192–218. doi: 10.2307/2527669

Meuwissen, T. H. E., and Luo, Z. (1992). Computing inbreeding coefficients in large populations. *Genet. Select. Evol.* 24, 305–313. doi: 10.1186/1297-9686-24-4-305

Misztal, I., Legarra, A., and Aguilar, I. (2009). Computing procedures for genetic evaluation including phenotypic, full pedigree, and genomic information. *J. Dairy Sci.* 92, 4648–4655. doi: 10.3168/jds.2009-2064

Nilforooshan, M. A. (2021). *pedSimulate: Pedigree, Genetic Merit and Phenotype Simulation. Version 0.1.1*. Available online at: https://cran.r-project.org/package=pedSimulate (accessed January 18, 2021).

Nilforooshan, M. A., and Saavedra-Jiménez, L. A. (2020). ggroups: an R package for pedigree and genetic groups data. *Hereditas* 157:17. doi: 10.1186/s41065-020-00124-2

Quaas, R. L. (1976). Computing the diagonal elements and inverse of a large numerator relationship matrix. *Biometrics* 32, 949–953. doi: 10.2307/2529279

R Core Team (2019). *R: A Language and Environment for Statistical Computing. Version 3.6.2*. Vienna: R Foundation for Statistical Computing.

Sargolzaei, M., and Iwaisaki, H. (2005). Comparison of four direct algorithms for computing inbreeding coefficients. *Anim. Sci. J.* 76, 401–406. doi: 10.1111/j.1740-0929.2005.00282.x

Sargolzaei, M., Iwaisaki, H., and Colleau, J. J. (2005). A fast algorithm for computing inbreeding coefficients in large populations. *J. Anim. Breed. Genet.* 122, 325–331. doi: 10.1111/j.1439-0388.2005.00538.x

Thompson, R. (1977). The estimation of heritability with unbalanced data. II. Data available on more than two generations. *Biometrics* 33, 497–504. doi: 10.2307/2529364

Tier, B. (1990). Computing inbreeding coefficients quickly. *Genet. Select. Evol.* 22, 419–430. doi: 10.1186/1297-9686-22-4-419

van de Geijn, R. A. (2011). *Notes on Cholesky Factorization*. Available online at: https://www.cs.utexas.edu/users/flame/Notes/NotesOnCholReal.pdf (accessed January 18, 2021).

Van Vleck, L. D. (2007). Computing numerator relationships between any pair of animals. *Genet. Mol. Res.* 6, 685–690.

Vazquez, A. I., Bates, D. M., Rosa, G. J., Gianola, D., and Weigel, K. A. (2010). Technical note: an R package for fitting generalized linear mixed models in animal breeding. *J. Anim. Sci.* 88, 497–504. doi: 10.2527/jas.2009-1952

Wallig, M., Microsoft, and Weston, S. (2020a). *foreach: Provides Foreach Looping Construct. Version 1.5.1*. Available online at: https://cran.r-project.org/package=foreach (accessed April 07, 2021).

Wallig, M., Microsoft, Weston, S., and Tenenbaum, D. (2020b). *doParallel: Foreach Parallel Adaptor for the 'Parallel' Package. Version 1.0.16*. Available online at: https://cran.r-project.org/package=doParallel (accessed April 07, 2021).

Wright, S. (1922). Coefficients of inbreeding and relationship. *Am. Nat.* 56, 330–338. doi: 10.1086/279872

Conflict of Interest: MN and BH are employed at the company Livestock Improvement Corporation, Hamilton, New Zealand.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

APPENDIX

The R5 family instances of Amazon elastic compute cloud feature either the first- or second-generation Intel Xeon Platinum 8000 series processor with a sustained all core Turbo CPU clock speed of up to 3.1 GHz (https://aws.amazon.com/ec2/instance-types/r5/).

The C5 family instances of Amazon elastic compute cloud feature custom second-generation Intel Xeon Scalable Processors with a sustained all-core turbo frequency of 3.6 GHz and maximum single core turbo frequency of 3.9 GHz (https://aws.amazon.com/ec2/instance-types/c5/).

The T2 family instances of Amazon elastic compute cloud are low-cost general-purpose machines backed by Intel Xeon processors that provide a baseline level of CPU performance with the ability to burst to full core (3.3 GHz) performance (https://aws.amazon.com/ec2/instance-types/t2/).