Strandén, I., Evans, R., Taskinen, M. (2026) Parallel computing for efficient inbreeding coefficient calculation, *Proceedings 13th World Congress on Genetics Applied to Livestock Production* 23491: https://doi.org/10.31274/wcgalp.23491
©2026 Strandén, et al. CC BY-NC-ND

Parallel computing for efficient inbreeding coefficient calculation

I. Strandén[1*], R. Evans[2], and M. Taskinen[1]

[1]Natural Resources Institute Finland (Luke), 31600 Jokioinen, Finland; [2]Irish Cattle Breeding Federation (ICBF), Link Road, Ballincollig, Ireland; [*]Corresponding author: ismo.stranden@luke.fi

Introduction

Inbreeding coefficients are widely used statistics for quantifying genetic relatedness and population diversity. Moreover, when adopting the single-step genomic model (Legarra *et al.*, 2009; Christensen and Lund, 2009), the use of inbreeding coefficients is required to preserve mathematical consistency between the two pedigree-based relationship matrices, one for all pedigree individuals and the other for the genotyped individuals. Consequently, efficient computation of inbreeding coefficients is required for large pedigrees such as those in national livestock genetic evaluations.

Several algorithms have been presented for computing inbreeding coefficients for educational and computational reasons. Sargolzaei and Iwaisaki (2005) compared four methods and showed that their SI algorithm outperformed the other methods in terms of computational efficiency. The SI algorithm relies on parent numbers to precede their progeny and in grouping ancestors of each individual chronologically using longest ancestral path. This ordering allows the use of an efficient vector-based data structure for the list of ancestors when computing inbreeding coefficient of an individual. Since the 2000s the advent of parallel computing has allowed an increase in computing power, and parallelization has been regularly used in genetic evaluations. Nevertheless, the computation of inbreeding coefficients typically relies on single processor algorithms.

In this study, we applied OpenMP-based parallel computing in both the recursive tabular method and the SI algorithm for computing inbreeding coefficients in a large beef cattle pedigree. OpenMP supports parallel computing in the currently common shared-memory computers having many processing units with multiple cores. We evaluated the parallel performance of these methods under different numbers of CPU threads. We report the total computing time and peak memory use.

Materials & Methods

Tabular method. Inbreeding coefficient of individual i can be computed as $F_i = \mathbf{A}_{ii} - 1$, where $\mathbf{A}_{ii}$ is the diagonal element of the additive relationship matrix $\mathbf{A}$ for individual i. Computation using the classical tabular method requires pedigree records to be ordered such that parents precede their offspring and requires to store the full $\mathbf{A}$ matrix. For large pedigrees, storing the full $\mathbf{A}$ matrix is unfeasible. A recursive formulation can be more efficient (Aguilar and Misztal, 2008):

$$A_{ij} = \begin{cases} 0 & if\, i = 0 \vee j = 0, \\ 1 + F_i & if\, i = j, \\ 0.5\left(A_{sj} + A_{dj}\right) & otherwise, \end{cases} \quad (1)$$

where individual i is younger than j, and s and d are the sire and dam of i, respectively. Aguilar and Misztal (2008) provided Fortran code of the recursive Eq.1. Their code can be easily adapted to use multiple threads by OpenMP (Fig. 1). In the implemented parallel version of the tabular method, each CPU thread calls a function to compute the recursive formulate for an individual whose inbreeding coefficient has not been computed.

```
Subroutine FbyRecur(Finbr, Pedig, Last)
integer Pedig(2,*)
real    Finbr(0:Last)

  $OMP PARALLEL PRIVATE(i, Sire, Dam)
  do i = 1 to Last
    if (Not computed Finbr(i)) then
      $OMP ATOMIC WRITE
      Finbr(i) = 0.5 * Rel_cffa(Finbr, Pedig, Sire, Dam)
    endif
  enddo
  $OMP END PARALLEL

end subroutine FbyRecur


Recursive Function Rel_cffa(Finbr, Pedig, a1, a2)

  if ((a1 <= 0) OR (a2 <= 0)) then
    return(0)
  else if (a1 == a2) then
    if (Not computed Finbr(a1)) then
      $OMP ATOMIC WRITE
      Finbr(a1)= 0.5*Relcffa(Finbr,Pedig,Pedig(1,a1),Pedig(2,a1))
    endif
    return(1+Finbr(a1))
  else
    if (a1 < a2) then
      return(0.5*Relcffa(Finbr, Pedig, a1, Pedig(1,a2)) + &
             0.5*Relcffa(Finbr, Pedig, a1, Pedig(2,a2)) )
  else
      return(0.5*Relcffa(Finbr, Pedig, a2, Pedig(1,a1)) + &
             0.5*Relcffa(Finbr, Pedig, a2, Pedig(2,a1)) )
  endif

end function Rel_cffa
```

Figure 1. Pseudo-code of the recursive tabular method when using OpenMP.

SI method. As presented in Sargolzaei and Iwaisaki (2004, 2005) following Henderson (1976), the additive relationship matrix can be expressed as **A**=**LBL**' where **L** is a lower triangular matrix and **B** is a diagonal matrix. Diagonal element *i* of **A** is

$$A_{ii} = \sum_{j=1}^{i} L_{ij}^2 B_{jj}. \quad (2)$$

Several algorithms that follow Eq.2 have been presented as listed in Sargolzaei and Iwaisaki (2005). The main principle in these approaches is that the non-zero elements of the **L** matrix are computed row-by-row. Each row considers ancestors of an animal and can be done in a memory efficient manner without storing the full matrix. Differences in these algorithms affect computational efficiency, with the SI method being the most efficient for large pedigrees.

The SI algorithm presented in Sargolzaei and Iwaisaki (2004) uses an ordered list of the ancestors in order to have fast access to ancestral paths. The ordering can be made when calculating the longest ancestral path (LAP) for each individual. Then, individuals are divided into groups according to their LAP number and processed from the longest LAP number to the shortest. The method can be easily parallelized by processing in parallel the individuals with the same LAP group number (Fig. 2). Thus, the main difference in our parallel SI approach to the original SI method is to divide the loop processing all individuals into a nested loop with the outer loop processing the LAP groups and the inner loop processing (in parallel) individuals with the same LAP number.

Study design and statistics. A pedigree from an ICBF beef cattle evaluation having about 9.5 million individuals was used to evaluate the parallel performance of the tabular and SI algorithms. Baseline computations used no parallel computing. Parallel computations used 2, 5, 10, and 15 CPU threads for both methods. We report the elapsed wall clock time (in minutes)

and the peak RAM (in GB) for each case. Parallel efficiency is measured by speed-up defined as the ratio of the computing time using a single thread to multiple threads. Thus, for a single-thread case, the speed-up is one, and a higher speed-up number indicates higher performance benefit from parallel computing.

```
integer SI(0:t)
real    B(n)
! Parallel OpenMP section with thread private MI, L, and Anc vectors:
$OMP PARALLEL PRIVATE(lloop,ilap,i,S,D,j,k,L,Anc,MI) FIRSTPRIVATE(t)
allocate(L(n), Anc(0:n), MI(0:t)) ! Allocate local vectors for each thread
MI = SI

do lloop = 0 to m - 1                ! Loop over LAP groups 0 to m - 1
  ! Parallel loop over LAP group lloop individuals.
  $OMP DO   ! Threads wait in the end of this loop before continuing to next LAP group.
  do ilap = 1 to nlaps(lloop)        ! Individuals within LAP group
    i = order(SI(lloop) + ilap)
    GET_PARENTS(i,S,D)               ! Parents of individual i
    B(i) = 0.5 - 0.25 * (F(S) + F(D)) ! Calculate diagonal B
    if ((S == 0) .OR. (D == 0)) then  ! Missing parents or one parent individuals are non-inbred
      F(i) = 0; L(i) = 0; cycle
    endif
    F(i) = -1; L(i) =  1    ! Initialize F and L for the individual i
    t = LAP(i);  PUSH(i,t)  ! Add individual to the LAP group t of its ancestors

    do while (t > -1)    ! Loop from the largest LAP group t to zero
      POP(j,t)           ! Get next ancestor from LAP group t
      GET_PARENTS(j,S,D) ! Add parents to ancestors if not already there, contribute to parents
      ADD_PARENT(j,S)
      ADD_PARENT(j,D)
      F(i) = F(i) + L(j) * L(j) * B(j)
      L(j) = 0
      if (MI(t) == SI(t)) t = t - 1 ! Move to next LAP group if all ancestors in group t are evaluated
    enddo
  enddo
  $OMP END DO
enddo
$OMP END PARALLEL
```

Figure 2. Pseudo-code of the kernel of the SI method subroutine when using OpenMP.

Results and Discussion

The SI method was significantly faster than the recursive tabular method in all cases and required less than 1000th of the computing time (Table 1). Despite large differences in computing times, both approaches achieved similar speed-up numbers and showed excellent parallel scalability (Fig. 3). For the tabular (SI in parenthesis) approach, the use of 2, 5, 10, and 15 CPU threads gave speed-ups of 2.0 (1.9), 4.7 (4.3), 8.9 (8.1), and 14.2 (13.0), respectively.

Table 1. Computing time (Time, min) and peak RAM memory needed (Peak RAM, GB) for the Tabular and SI approaches by number of CPU threads used.

Threads[1]	Time (min)		Peak RAM (GB)	
	Tabular	SI	Tabular	SI
1	1911	1.083	0.176	0.503
2	966	0.583	0.315	0.700
5	404	0.250	0.536	1.290
10	214	0.133	0.839	2.273
15	134	0.083	1.208	3.264

[1] Number of CPU threads

The SI method had a higher peak memory use than the tabular method in all cases, requiring about 2.5 times more memory regardless of the number of cores used (Table 1). Both approaches showed a similar trend in the increase of RAM use by the number of CPU threads. The increase was approximately linear. When fitting a linear regression of peak RAM use on the number of CPU

threads using the data in Table 1, the slope term was about 0.40. This indicates that the peak RAM use increases by about 40% with an additional CPU thread when using our data. Because the single-threaded run required some hundreds of megabytes of RAM (176 MB for the tabular, and 503 MB for the SI), the RAM amounts were manageable for low numbers of CPU threads. Furthermore, as the SI method needed less than a minute computing time already with two CPU threads, computation of inbreeding coefficients for large pedigrees can be computed very fast with manageable amount of RAM.

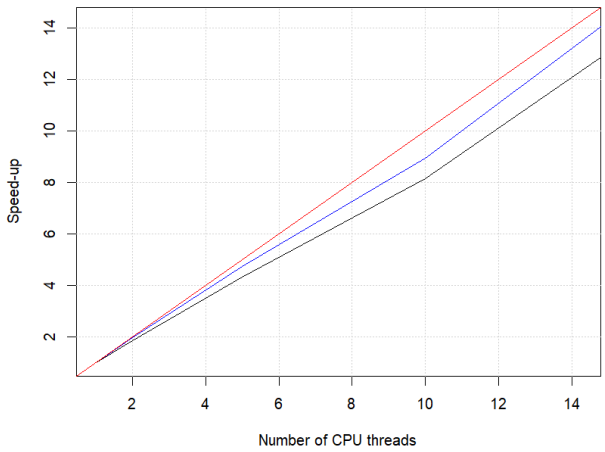

Figure 3. Speed-up by the number of CPU threads for the tabular method (blue) and the SI method (black) with the diagonal line (red) as a reference.

Conclusions

Both algorithms showed similar speed-ups as the number of CPU threads increased. However, the SI approach showed superior computational efficiency over the tabular method even with parallel computing. With 15 CPU threads, the SI method needed only 5 seconds and used 3.2 GB of memory, while the tabular method took over 2 hours and used 1.2 GB of memory. Increasing the number of CPU threads increased the peak memory needs of the two approaches at about the same pace. Due to the high speed-up with even low CPU thread numbers, it may be sufficient to use only a few CPU threads.

References

Aguilar I., and Misztal I. (2008) Journal of Dairy Science 91:1669-1672. doi.org/10.3168/jds.2007-0575

Aguilar I., Misztal I., Johnson D.L., *et al*. (2010) Journal of Dairy Science 93:743-752. doi.org/10.3168/jds.2009-2730

Christensen O.F., and Lund M.S. (2010) Genetics Selection Evolution. 42:2. doi.org/10.1186/1297-9686-42-2

Henderson C.R. (1976) Biometrics 32:69–83. doi.org/10.2307/2529339

Legarra A., Aguilar I., and Misztal I. (2009) Journal of Dairy Science 92:4656-4663. doi.org/10.3168/jds.2009-2061

Sargolzaei M., and Iwaisaki H. (2004) Japanese Journal of Biometrics 25:25-36. doi.org/10.5691/jjb.25.25

Sargolzaei M., and Iwaisaki H. (2005) Animal Science Journal 76:401-406. doi.org/10.1111/j.1740-0929.2005.00282.x