Software project 'miraculix': Efficient computations with large genomic datasets

M. Schlather[1*], A. Freudenberg[1], G. Moerkotte[2], T. Pook[3] and J. Vandenplas[3]

[1]*Institute for Mathematics, University of Mannheim, B6, 26, D-68159 Mannheim, Germany*

[2]*Institute for Computer Sciences, University of Mannheim, B6, 26, D-68159 Mannheim, Germany*

[3]*Animal Breeding and Genomics, Wageningen UR, P.O. Box 338, 6700 AH, The Netherlands*

martin.schlather@uni-mannheim.de

Abstract

We present mathematical approaches for CPU accelerations to calculate matrix multiplications between a Single Nucleotide Polymorphism (SNP) matrix and another SNP matrix or a real-valued matrix. These accelerations are important in crucial time-relevant calculations of single-step evaluations and other methods in genetics. The presented algorithms are much faster than previous algorithms. The C-code is released as part of the software project 'miraculix', which has been integrated into existing software such as MoBPS and MiXBLUP. We also discuss precision problems and missing SNP genotypes.

Key words: CPU, fast calculation, matrix multiplication, SIMD, SSE

Introduction

Many free and commercial software packages offer a broad range of methods in quantitative genetics, such as PLINK (Chang et al., 2015) and GCTA (Yang et al., 2011) to name a few. Others deal only with specific aspects, e.g., MiXBLUP (Vandenplas et al., 2022) with breeding value estimation or MoBPS (Pook, 2020) with breeding program simulation. In many of these applications, the most time-consuming steps are related to the Single Nucleotide Polymorphism SNP-matrix $Z \in \{0,1,2\}^{n\times s}$, which is multiplied to its transposed or a real-valued matrix. Here, n is the number of individuals and s the number of SNPs per individual. Many packages uncompress the 2-bit-packed SNP-matrix in some way before further calculations. Here, some approaches for CPUs are presented that avoid this unpacking partially or fully.

We will deal with matrix products of the form $Z^\top Z$ and $ZZ^\top$, which is the so-called unweighted genomic relationship matrix (GRM), up to a factor (Fragomeni et al., 2017). Afterwards, we will deal with products of the form $Z^\top V$ where $V \in R^{n\times p}$. As matrix products boil down mathematically to a collection of scalar products, we consider here scalar products, only. We first assume that missing values are absent. Afterwards, SNP matrices with missing values are considered together with certain precision considerations and centring of SNP matrices, since all three problems have similar mathematical foundations. We refer to Freudenberg et al. (2023a, 2023b) for benchmarks, including GPU solutions, and to Schlather (2020) for related and former methods.

Materials and Methods

For simplicity and clarity, we will primarily refer to commands of the Intel SSE instruction set family (128 bits). We comment on AVX2 and AVX512 explicitly when extensions of SSE are not obvious or when SSE is not enough for the given instructions. Note that most SSE commands can be easily transferred to the NEON instruction set through the header file sse2neon.h, for instance, in contrast to AVX commands.

Notations

In the subsequent pseudo-codes, &, |, and >> denote bitwise and, bitwise or, and shift to the

right, respectively. The signs '+' and '-' denote addition and subtraction in the decimal system. They can be interpreted as parallel operations on k-bit pieces if it is guaranteed that no k-bit overflow or underflow appears. We will use this fact several times, for $k = 2,4,6,8$ bits.

In case a register is filled by a repeated sequence s of bits we write (s)*. For instance, (01)* means that zeros and ones are alternated. The variable 'sum' refers to some register that accumulates summands; in case partial sums must be calculated first, sum is further added up in a variable called `total'.

Variables in the code pieces refer to Single Instruction Multiple Data (SIMD) registers, if not indicated differently; a and b indicate SNP values with a certain compressed coding. Finally, indexing assumes little endian.

Mini Lookup Tables

The SSE command _mm_shuffle_epi8 offers a lookup table with 16 entries of 1 Byte. AVX implementations realize only more parallel lookups, while the size of the lookup table does not change. The lower 4 bits of each byte in the SIMD register are used to realize 16 lookups at once at a cheap prize of at most 1 clock cycle.

Such mini lookup tables have a broad field of applications. For instance, they can be used for data transformation, adding-up neighboured 2-bit values, and to implement population counts (i.e., the number of bits in a register that equal 1) on systems without genuine popcnt command (Mula et al., 2016). We define

shuffle(x) :=

_mm_shuffle_epi8(x & (00001111)*, table) + _mm_shuffle_epi8((x>>4) & (00001111)*, table)

where the values of the table depend on the context and can always be obtained by simple calculations. For instance, the popcnt table is {0,1,1,2, 1,2,2,3, 1,2,2,3, 2,3,3,4}. Since 'sum' may not exceed the value 255, regular clearance of 'sum' is necessary. In case of popcnt this must happen after 31 iterations, the latest.

Large Lookup Tables

A lookup table with more than 16 entries can still be accessed in a reasonable time if the table fits well into the L1 cache. Hence, lookup tables for AVX registers should be addressed by at most 8-bit, and ALU registers by at most 14 bits.

Strassen algorithm

An important algorithm for calculating a matrix product between large matrices is the Strassen algorithm (Strassen, 1969). For a quadratic matrix $Z \in R^{n \times n}$ the standard costs for the product $ZZ^\top$ are of order n^3, whereas the costs of the Strassen algorithm are of order $n^{2.807}$. Indeed, in a standard set-up of a double-precision matrix Z, the Strassen algorithm is faster than the standard algorithm if n is larger than about 10^3. Numerical experiments suggest that the Strassen algorithm will be beaten in a SNP-SNP matrix multiplication by the best algorithms presented below up to $n \approx 10^6$. Note that the Strassen algorithm performs best in case of quadratic matrices. Otherwise, the smallest edge length is decisive for its performance. Hence the Strassen algorithm will never be an option for calculating $Z^\top V$ in a single step framework, where V is a vector or a small matrix. A further disadvantage of the Strassen algorithm is that its numerical errors are larger than those of the approaches presented here. Since the fast multiplication of matrices is still an active area of research, the limit $n \approx 10^6$ may change in future.

SNP-SNP scalar products by integer product

An immediate way of calculating the scalar product from a compressed 2-bit representation is to extract the first two bits of each of the two vectors a and b, and to continue with integer arithmetic. Then, the next two bits are extracted using shifting, and so on. Clearly, this procedure can be vectorized. Of advantage here is the SSE command _mm_madd_epi16, which multiplies and adds two consecutive 16-bit integers so that only 7 shifts are necessary. This method is based on the 2-bit standard binary coding of $\{0,1,2\}$; in case of PLINK 1 binary coding, a preceding transformation is necessary to the standard 2-bit binary coding.

The speed can be improved by the following consideration. Let $a_1, a_2, b_1, b_2 \in \{0,1,2\}$ be 4 SNP numbers. The two products $a_1 b_1$ and $a_2 b_2$ can be calculated in a single multiplication through

$$(a_1 + 2^c a_2)(b_1 + 2^c b_2) =$$
$$a_1 b_1 + 2^c (a_2 b_1 + a_1 b_2) + 2^{2c} a_2 b_2$$

provided the result is identifiable, i.e., the three summands on the right-hand side occupy different bits in the binary representation of the above value. This is the case when $c > 3$. Hence, convenient choices for c are $c = 4,6$ or 8. For instance, choosing $c = 8$ reduces the number of calls of _mm_madd_epi16 to 4 and the number of shifts to 3 by the following code:

```
for (i=0; i<8; i+=2)
  sum += _mm_madd_epi16((a >> i) &
        (00000011)*, (b >> i) & (00000011)*)
```

Clearance of the variable 'sum' is necessary after 7 iterations,

```
total += ((char *) sum)[0] + ((char *) sum)[2]
```

The analogue AVX512 command is _mm512_dpbusd_epi32, which sums up 4 products of adjacent 8-bit integers into a 32-bit integer. Hence, $c = 4$ and $c = 8$ are not possible and $c = 6$ leads to 3 calls of _mm512_dpbusd_epi32.

SNP-SNP scalar products by lookup tables

The following algorithm relies on data with PLINK 1 binary format, where the coding $00_p = 0_d$, $10_p = 1_d$ and $11_p = 2_d$ is used. Here, the index p and d denote PLINK 1 binary coding and decimal coding, respectively,

```
c:= a xor b
d:= ~(c >> 1) & c & (01)*
sum += shuffle( (a & b) - d )
```

Note that $d = 01_b$, if the decimal result is 2, and $d = 00_b$ otherwise.

SNP-SNP scalar product for chromosome data

If data are available per chromosome, we have two matrices $Z_{11}, Z_{12} \in \{0,1\}^{n \times s}$ where the value 1 indicates a deviation from the reference allele and $Z_{11} + Z_{12}$ equals the SNP matrix Z. Then, the non-centred relationship matrix is given by

$$(Z_{11} + Z_{12})(Z_{11}^\top + Z_{12}^\top) =$$
$$Z_{11} Z_{11}^\top + Z_{11} Z_{12} + Z_{12} Z_{11}^\top + Z_{12} Z_{12}^\top$$

Note that all scalar products on the right-hand side are between binary data, so that the multiplication step can be realized by the bitwise & and the adding-up by popcnt. Obviously, this algorithm can be used also for genomic data after a preprocessing step, where the genome data are artificially split into data per chromosome.

SNP-SNP scalar product based on the Hamming Distance

An interesting algorithm has been introduced in PLINK (Purcell et al., 2007; Chang et al., 2015) and has been based on the idea that a value can be represented by the number of bits that equal 1 in a 4-bit representation. The values of the vectors a and b must be coded asymmetrically by two mappings f and g, say, as a coding by a single mapping is not possible. Then, the bitwise &-operator is applied before popcnt is applied. Table 1 gives a possible realisation.

Table 1. Values for the Hamming distance method.

$f(\cdot) \wedge g(\cdot)$	$g(0)=0000_b$	$g(1)=0011_b$	$g(2)=1111_b$
$f(0)=0000_b$	000_b	0000_b	0000_b
$f(1)=0110_b$	0000_b	0010_b	0110_b
$f(2)=1111_b$	0000_b	0011_b	1111_b

Overview over SNP-SNP algorithms

Tables 2-6 give an overview over some properties of the divers approaches.

Table 2. Amount of additional cache/memory.

Method	Cache/memory needs
Integer product	Space for partial sums
Mini lookup table	Space for partial sums
Per chromosome	No extra needs for AVX512
Hamming distance	Each SNP needs 8 bits instead of 2

Table 3. Rough speed of the algorithm; the speed depends on the hardware and the specific coding.

Method	Speed
Integer product	Highly hardware dependent; fast on AVX512 & GPU
Mini lookup table	Intermediate
Per chromosome	High on AVX512
Hamming distance	High on AVX512

Table 4. Generality of the algorithm with respect to the hardware. Note that AVX512 has a lot more commands available and that the available set of commands differs between CPU and GPU.

Method	Hardware generality
Integer product	Any; currently, hardware is being developed in favour of this algorithm
Mini lookup table	All SIMD variants
Per chromosome	Well adapted to GPU & AVX512; modifications work for all SIMD variants
Hamming distance	All SIMD variants

Table 5. Number of registers needed for the calculations.

Method	Register need
Integer product	Several
Mini lookup table	Many
Per chromosome	Few
Hamming distance	Few

Table 6. Generality of the algorithm with respect to the coding of a SNP. If the algorithm is not general, much more memory is needed as a preceding re-coding is necessary.

Method	SNP coding generality
Integer product	Standard binary coding needed; re-coding on the fly possible
Mini lookup table	Principle suits any 2-bit coding; adaptions necessary
Per chromosome	Inherent coding; ideal for information per chromosome
Hamming distance	Inherent coding

SNP-double scalar products

In contrast to the bunch of algorithms for SNP-SNP scalar products, the spectrum of possible approaches to perform SNP-double scalar products is narrower and the algorithms simpler.

SNP-double scalar products can be performed by preceding conversion to double, essentially in the same way as for the integer product, except that the obtained, intermediate integer value is transformed into a double-precision value before being multiplied.

Since a SNP can take only the three values 0, 1 and 2, the implementation by addition is another, ensnaring approach. There are at least two variants of this idea. First, GPUs and AVX512 allow a conditional addition by indirect or direct masking, e.g., _mm512_mask_add_pd in AVX512, without loss of speed in comparison to a simple add command. Second, the if-condition is a moderately expensive command provided it does not lead to a far jump. Hence, the multiplication can be implemented by two nested if-conditions.

The last approach given here is more intriguing and mathematically more complex. It is called 5codes (Freudenberg et al., 2023b). For convenience, we repeat the algorithm here. Let $Y = Z^\top V$ and start with the well-known fact, that for fixed, real-valued values $V_j \in R$, the product $Z_{i,j}^\top V_j$ takes only 3 different values for arbitrary $Z_{i,j}^\top \in \{0,1,2\}$. Hence, a partial scalar product $Z_{i,j}^\top V_j + \cdots + Z_{i,j+k-1}^\top V_{j+k-1}$ can take at most 3^k different values. So, by creating a lookup table $H_{j,k}$, we can replace

```
for (j=0; j<nrow(Z); j+=k)
  Y[i] += Z[ j,i]*V[j]+…+Z[ j+k-1,i]*V[j+k-1]
```

by

```
for (j=0; j<nrow(Z); j+=k)
  Y[i] += H_{j,k}(Z[ j,i], … Z[ j + k - 1,i]) .
```

Since $3^5 = 243$, we can use $k = 5$ SNP values to index H by a single byte. Hence, a lookup table of doubles has less than 2000 Bytes. Now, m tables may fit into the L1 cache, so that the final pseudo-code reads

```
for (j=0; j<nrow(Z); j += m * k)
  Y[i] += H_{j,k}(Z[ j,i], …, Z[ j + k - 1,i]) + … +
    H_{j+(m-1)k,k}(Z[j + (m-1) k, i], …, Z[j + m k-1, I])
```

Overview over SNP-double algorithms

Tables 7-11 give a comparative overview of the properties of the different approaches.

Table 7. Amount of additional cache/memory.

Method	Cache/mem need
Conversion to double	Space for converted values
Conditional adding (mask)	None
Conditional adding (if)	None
5-codes	Lookup table in L1

Table 8. Rough speed of the algorithm; the speed depends on the hardware and the specific coding.

Method	Speed
Conversion to double	Intermediate
Conditional adding (mask)	Very high
Conditional adding (if)	Very dependent on the implementation
5-code	High

Table 9. Generality of the algorithm with respect to the hardware. Note that AVX512 has a lot more commands available and that the available set of commands differs between CPU and GPU.

Method	Hardware generality
Conversion to double	Any
Conditional adding (mask)	AVX512 & GPU
Conditional adding (if)	Any
5-codes	Any

Table 10. Number of registers needed for the calculation.

Method	Register need
Conversion to double	Few extra registers
Conditional adding (mask)	Few extra registers
Conditional adding (if)	Extra ALU registers
5-codes	Extra ALU registers

Table 11. Generality of the algorithm with respect to the coding of a SNP. If the algorithm is not general, much more memory is needed as a preceding re-coding is necessary.

Method	SNP coding generality
Conversion to double	General; adaptions necessary
Cond. adding (mask)	Adaptions necessary
Conditional adding (if)	General; adaptions necessary
5-codes	Inherent coding

Centring, missing values and precision

The above sections have considered the scalar product for the non-centred GRM, only. There, it has also been assumed that no missing values are present. In this section, we extend the above results to centred GRM and allow for missing values. We assume, however, that the portion of missing values is small.

A typical situation in genetics is that the phenotype V is non-negative. Hence, all products in $Z^\top V$ are non-negative, so that the calculation of the scalar product cannot profit from cancellations. A simple measure for an increased precision is to centre V and/or Z before calculation. Of course, further action to increase precision can be taken, e.g., using higher precision formats such as long double.

Below, we choose an approach that includes considerations for calculating both GRM and LD, in a rather general set-up.

Centred GRM

Schlather (2020) has shown that centred and normalized GRM (VanRaden, 2008; Wals and Lynch, 2018) can be calculated without loss of performance. Indeed, first the non-centred GRM can be calculated as above. Afterwards, the result can be corrected at low costs. To this end, let I_k be the vector of length k whose components are all equal to 1. The centred and normalized GRM G is defined as

$$G = (Z - Q)\,(Z - Q)^\top/\sigma^2$$

where

$$Q = 2I_n p_s^\top$$

$$\sigma^2 = 2\sum_{i=1}^{s} p_{s,i}(1 - p_{s,i})$$

and the $p_{s,i}$ are the allele frequencies. Then,

$$\sigma^2 G = ZZ^\top - I_n(2Zp_s)^\top - (2Zp_s)I_n^\top + 4I_n({p_s}^T p_s){I_n}^\top.$$

Obviously, the matrix $\sigma^2 G$ can be calculated from $ZZ^\top$ at low computational costs of order $n[s + n]$. As the calculation of σ^2 has costs of order s, the total computational costs for retroactive centring are some magnitudes smaller than the costs for calculating the cross-product $ZZ^\top$.

If there are no missing values and p_s equals the empirical allele frequency $n^{-1}Z^\top I_n/2$, the value $2n^2\sigma^2$ and the matrix $n^2\sigma^2 G$ are integer-valued and hence can be calculated exactly, so that the numerical errors in G can be reduced to a minimum. The costs for calculating $2n^2\sigma^2$ and

$n^2\sigma^2 G$ from $ZZ^\top$ are also of order $n[s+n]$, see Schlather (2020) for details. Note that the components of $ZZ^\top$ are unsigned 32-bit integers in standard applications, whereas $2n^2\sigma^2$ and $n^2\sigma^2 G$ need a 64-bit integer representation.

Allele frequencies in presence of missing values

Let $N \in R^{s\times s}$ and $S \in R^{n\times n}$ be the diagonal matrices whose diagonal elements equal to n (respectively s) minus the number of missing values in the respective row (column) of $Z^\top$. Then, the vector of empirical allele frequencies might be defined as

$$f_s := \frac{1}{2} N^{-1} Z^\top I_n$$

Let

$$g_n := \frac{1}{2} S^{-1} Z I_s$$

be the analogue mean taken in the direction of the SNPs, which appears in LD calculations.

Numerical centring

While the centring of GRM should always be performed retroactively, a preceding centring of Z and/or V in $Z^\top V_n$ or ZV_s increases the precision of the result. A retroactive correction of this numerical centring is of low cost. Let

$$B = Z - 2cI_n p_s^\top,$$

where $p_s \in R^s$ is any arbitrary vector. It is close to f_s in standard practical applications. For an advantageous centring of V_n, we aim to minimize

$$\| B(V_n - \mu_n e_n) \| = \min_{\mu_n}!, V_n \in R^n$$

for some fixed vectors $e_n \in R^n$, which may depend on Z. The minimization problem has the solution

$$\mu_n = \frac{e_n^\top B^\top B}{e_n^\top B^\top B e_n} V_n,$$

where

$$e_n^\top B^\top B =$$

$$e_n^\top ZZ^\top - 2m_n c q_n^\top \left[\left[c - \frac{e_n^\top Z I_s}{m_n}\right] I_{n\times n} - \frac{S}{s}\right]$$

with $I_{n\times n}$ the identity matrix and $m_n = 2se_n^\top q_n$.

If there are only a few missing values, i.e. $S/s \approx I_{n\times n}$, we have

$$e_n^\top B^\top B \approx e_n^\top ZZ^\top - 2m_n c(c-2) g_n^\top.$$

If we further choose $e_n = I_n$, then

$$I_n^\top B^\top B \approx 2n f_s^\top Z^\top + 2m_n c(c-2) g_n^\top$$

and

$$I_n^\top B^\top B I_n \approx I_n^\top ZZ^\top I_n + c(c-2)\frac{m^2}{s}$$

where $m = I_n^\top Z I_s$. Analogous formulae hold for an advantageous centring of V_s.

Genetic centring

In genetics, the centred matrices

$$Z - 2I_n p_s^\top \text{ and } Z^\top - 2p_s I_n^\top, z \in \{0,1\}.$$

are of interest, where p_s is the or any given allele frequency. Here, we combine these centred matrices with the numerical centring above in a rather general way. To this end, let $z \in \{0,1\}$ denote whether genetically motivated centring is of interest, i.e., we consider

$$Z - 2zI_n p_s^\top \text{ or } Z^\top - 2zp_s I_n^\top, z \in \{0,1\}.$$

Let $c, \nu \in \{0,1\}$ denote whether numerical centring of Z and V, respectively should be performed. Then we get for arbitrary $\mu_n, \mu_s \in R$, $p_s \in R^s$, and $q_n \in R^n$, that

$$(Z^\top - 2zp_s I_n^\top)V_n =$$
$$(Z^T - 2cI_s q_n^\top)(V_n - \nu I_n \mu_n)$$
$$- 2z(I_n^\top V_n)p_s + 2c(q_n^\top V_n)I_s + \nu\mu_n Z^\top I_n$$
$$- c\nu\mu_n(q_n^\top I_n)I_s$$

and

$$(Z - 2zI_n p_s^\top)V_s =$$
$$(Z - 2cI_n p_s^\top)(V_s - \nu I_s \mu_s) + \nu\mu_s Z I_s +$$
$$2(c-z)(p_s^\top V_s)I_n - c\nu\mu_s(p_s^\top I_s)I_n$$

so that the first term on each right side is critical concerning computational costs, and the remaining summands can be considered as correction terms. Note that $Z^\top I_n$ needs to be calculated only once in for every genotype matrix

Z. More generally, let $\zeta \in \{0,1\}$ indicate the centring in SNP direction. Then formulae for

$$(Z^\top - 2zp_s I_n^\top - 2\zeta I_s q_n^\top)V_n$$

and

$$(Z - 2zI_n p_s^\top - 2\zeta q_n I_s^\top)V_s$$

can easily be derived from the above equations. Note that all the above formulae hold independent of the values of p_s and q_n. We have only assumed that the numerical centring of the matrices Z and $Z^\top$ uses the same vectors.

Missing values

Assume we aim to calculate

$$x = (Z^\top - 2zp_s I_n^\top - 2\zeta I_s q_n^\top)V_n$$

or

$$y = (Z - 2zI_n p_s^\top - 2\zeta q_n I_s^\top)V_s$$

for arbitrary vectors $p_s \in R^s$ and $q_n \in R^n$ with a missing value in the position (j_k, i_k) of the matrix Z for $k = 1, \dots, \ell$. We define $Z_{j_k,i_k} := 0$ for all k and let I be the set of coordinates of all ℓ positions. Then, we have for $z, c, \nu \in \{0,1\}$ and arbitrary $\mu_n, \mu_s \in R$, $p_s \in R^s$, and $q_n \in R^n$, that

$$x_\alpha = \sum_{\beta,(\alpha,\beta)\notin I} (Z^\top - 2zp_s I_n^\top - 2\zeta I_s q_n^\top)_{\alpha\beta}\,(V_n)_\beta$$

$$= ((Z^\top - 2zp_s I_n^\top - 2\zeta I_s q_n^\top)V_n)_\alpha + 2\sum_{\beta,(\alpha,\beta)\in I} (z(p_s)_\alpha + \zeta(q_n)_\beta)(V_n)_\beta$$

$$y_\alpha = ((Z - 2zI_n p_s^\top - 2\zeta q_n I_s^\top)V_s)_\alpha + 2\sum_{\beta,(\beta,\alpha)\in I} (\zeta(q_n)_\alpha + z(p_s)_\beta)(V_s)_\beta.$$

This shows that matrix multiplications can be corrected for missing values retroactively even in very general set-ups. The corrections terms, i.e., the second summands in the above two equations, cause total computational costs proportional to the number of missing values in Z. The proportionality constant is large, however, because of cache misses, the outage of SIMD commands and the outage of tiling, at least in simple implementations.

Implementation of the numerical centring

While the centring of V_n and V_s is simple, the centring of Z and $Z^\top$ comes with extra computational costs for the conditional adding algorithms. Both the conversion to doubles and the 5-code algorithm do not lose speed and the implementation of the numerical centring is simple.

Conclusion

Algorithms for compressed SNP data can differ largely from simple approaches, such as decompression. Fast algorithms are hardware dependent and so change over time. Centring and missing values do not need to be considered in fast algorithms provided the number of missing values is small. Some increase in precision is possible without loss of speed, but with additional programming effort and use of special coding, e.g., 5-codes.

References

Chang, C.C., Chow, C.C., Tellier, L.C.A.M., Vattikuti, S., Purcell, S.M. and Lee, J.J. 2015. Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigasci. 4.* https://doi.org/10.1186/s13742-015-0047-8.

Fragomeni, B.O., Lourenco, D.A.L., Masuda, A., Legarra, Y. and Misztal, I. 2017. Incorporation of causative quantitative trait nucleotides in single-step GPLUB. *Genet. Sel. Evol. 49*, 59. https://doi.org/10.1186/s12711-017-0335-0.

Freudenberg, A., Schlather, M., Vandenplas, J., Pook, T. and Evans, R. 2023a. Accelerating single-step evaluations through GPU offloading. *Interbull Bulletin 59*, 23-32

Freudenberg, A., Vandenplas, J., Schlather, M., Pook, T., Evans, R. and ten Napel, J. 2023b. Accelerated matrix-vector multiplications for

matrices involving genotype co-variates with applications in genomic prediction. *Front. Genet. 14*, 1220408. https://doi.org/10.3389/fgene.2023.1220408.

Mula, W., Kurz, N. and Lemire, D. 2018. Faster population counts using AVX2 instructions. *Comput. J., 6*, 111-120. https://doi.org/10.1093/comjnl/bxx046.

Pook. T. 2020. MoBPS: Modular Breeding Program Simulator. R package version 1.4.15. https://github.com/tpook92/mobps.

Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M.A.R., Bender, D., Maller, J., Sklar, P., de Bakker, P.I.W., Daly, M.J. and Sham, P.C. 2007. A toolset for whole-genome association and population-based linkage analysis. *Am. J. Hum. Genet. 81*, 559–575. https://doi.org/10.1086/519795.

Schlather, M. 2020. Efficient calculation of the genomic relationship matrix. *BioRxiv*, https://doi.org/10.1101/2020.01.12.903146.

Strassen, V. 1969. Gaussian elimination is not optimal. *Num. Math. 13,* 354–356. https://doi.org/10.1007/BF02165411.

Vandenplas, J. Veerkamp, R.F., Calus, M.P.L., Lidauer, M.H., Strandén, I., Taskinen, M., Schrauf, M.F.S.G., and ten Napel. J. 2022. MiXBLUP 3.0 – software for large genomic evaluations in animal breeding programs. In *Proceedings of 12th World Congress on Genetics Applied to Livestock Production (WCGALP)*. Wageningen Academic Publishers, Wageningen, pp. 1498–1501. https://doi.org/10.3920/978-90-8686-940-4_358.

VanRaden, P.M. 2008. Efficient methods to compute genomic predictions. *J. Dairy Sci. 91*, 4414–4423. https://doi.org/10.3168/jds.2007-0980.

Walsh, B. and Lynch, M., 2018. Evolution and Selection of Quantitative Traits. Oxford University Press, Oxford. https://doi.org/10.1093/oso/9780198830870.001.0001.

Yang, J., Lee, S.H., Goddard, M.E. and Visscher, P.M. 2011. GCTA: A tool for genome-wide complex trait analysis. *Am. J. Hum. Genet. 88*, 76–82. https://doi.org/10.1016/j.ajhg.2010.11.011.