Accelerated computations for iterative-solver techniques in single-step BLUP models

Alexander Freudenberg[*,1], Jeremie Vandenplas[2], Martin Schlather[1], Torsten Pook[2], Ross Evans[3], and Jan ten Napel[2]

[1]Chair of Applied Stochastics, University of Mannheim, Mannheim, Germany
[2]Animal Breeding and Genomics, Wageningen UR, Wageningen, The Netherlands
[3]Irish Cattle Breeding Federation, Ballincollig, Ireland

July 6, 2023

Abstract

In the last two decades, a number of so-called single-step models have been suggested to incorporate both pedigree and genomic data in statistical breeding value estimation for breeding programs. Yet, steadily growing population sizes and the suboptimal use of computational resources are pushing the practical application of these models to their limits. As an extension to the *miraculix* package, we have developed tailored solutions for the computation of genotype matrix multiplications, a critical bottleneck when iteratively solving equation systems associated with single-step models. Targeting modern Nvidia® GPUs as well as a broad range of CPU architectures, our implementation significantly accelerates the estimation of breeding values in large population sizes. *miraculix* is a C/CUDA library released under the Apache 2.0 license. It is freely available at https://github.com/alexfreudenberg/miraculix.

1 Introduction

Over the past few decades, the incorporation of genomic information has become essential for ensuring progress in breeding (Schaeffer, 2006). A routine task in animal breeding is the estimation of breeding values within a population, that is, the estimation of the average additive effects of the alleles that an individual receives from both parents for a particular trait. Though breeding values are

*Corresponding author, Chair of Applied Stochastics, University of Mannheim, B 6, 26, Mannheim, Germany, alexander.freudenberg@uni-mannheim.de

estimated most accurately through utilizing genomic data, it is usually too costly to genotype a whole population. This is particularly true when analyzing large populations, like national dairy evaluations with millions of animals (Misztal et al., 2022). This circumstance has sparked a debate on how to jointly analyze phenotypic and pedigree information of genotyped and ungenotyped animals with Single Polymorphism Nucleotide (SNP) genotypes of genotyped animals.

Due to their desirable statistical properties, the single-step genomic BLUP (ssGBLUP) model (Legarra et al., 2009) and variations thereof have gained popularity, which combine both the genomic relationship matrix (GRM) $\mathbf{G}$ and the pedigree-based relationship matrix $\mathbf{A}$ into a generalized relationship matrix. Large population sizes used in animal breeding programs, ranging from tens of thousands to millions of individuals, have motivated research in computational strategies to solve the associated mixed model equations (MME) efficiently through the use of high-performance computing (HPC) techniques. One of the proposed approaches is the algorithm for Proven and Young Animals (APY), which approximates the inverse of the GRM, $\mathbf{G}^{-1}$, through genomic recursion on a subset of core animals (Misztal et al., 2014). As an alternative concept, the original ssGBLUP model was reformulated to, first, allow the use of well-established numerical software (Legarra and Ducrocq, 2012) and, second, avoid the explicit construction and inversion of the GRM $\mathbf{G}$ using the Woodbury decomposition. The result models were coined single-step GT(A)BLUP models (Legarra et al., 2009; Mäntysaari et al., 2017). Additionally, single-step SNP BLUP (ssSNPBLUP) models were proposed to estimate SNP effects directly, similarly avoiding $\mathbf{G}$ and its inverse (Liu et al., 2014).

Considering practical implementation, the role of the pedigree-based relationship matrix $\mathbf{A} = \begin{pmatrix} \mathbf{A}_{nn} & \mathbf{A}_{ng} \\ \mathbf{A}_{gn} & \mathbf{A}_{gg} \end{pmatrix}$ (with n and g denoting non-genotyped and genotyped animals respectively, and the submatrices of $\mathbf{A}$ defined as the respective relationship matrices) in these models allows the use of sparse matrix operations, as its inverse,

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}^{nn} & \mathbf{A}^{ng} \\ \mathbf{A}^{gn} & \mathbf{A}^{gg} \end{bmatrix}, \tag{1}$$

is itself sparse (Mrode, 2014). Hence, the multiplication of the inverse of the pedigree-based relationship matrix among genotyped individuals $\mathbf{A}_{gg}^{-1}$ by a vector can be calculated efficiently using its Schur's complement (Strandén et al., 2017)

$$\mathbf{A}_{gg}^{-1} = \mathbf{A}^{gg} - \mathbf{A}^{gn}(\mathbf{A}^{nn})^{-1}\mathbf{A}^{ng}, \tag{2}$$

which only involves sparse matrices. Additionally, established iterative-solver algorithms (e.g., the preconditioned conjugate gradient (PCG)) can be employed in the resulting equation systems (Strandén and Lidauer, 1999; Misztal et al., 2009) to avoid constructing the full coefficient matrix.

Nevertheless, the computational load of the remaining mathematical operations and the slow PCG convergence have been somewhat prohibitive to the application of ssGTABLUP and ssSNPBLUP models in ultra-large-scale settings.

To mitigate this issue, a number of numerical advances have been proposed to improve convergence speed (Vandenplas et al., 2018). So far, however, improvements in accelerating the involved matrix arithmetics have been limited to the application of shared-memory parallel libraries, such as Intel® MKL library and PARDISO (Alappat et al., 2020), and unpacking the compressed SNP matrix $\mathbf{Z}$ into the CPU cache for the matrix-matrix product in the PCG iteration (Misztal et al., 2009; Vandenplas et al., 2020). Bit-level algorithms have been employed by, for instance, the popular PLINK software (Chang et al., 2015), in the efficient implementation of genome-wide association studies (GWAS), which rely on similar genotype matrix operations. Yet, these routines are not accessible for the use in single-step BLUP models. PLINK also implements BLAS-based routines for the calculation of $\mathbf{G}$ on a Nvidia® GPU in its version 2.0. However, this functionality works on uncompressed SNP data stored in single-precision floating-point values, thereby significantly limiting possible problem sizes. Other authors have used GPUs to accelerate model training for machine learning algorithms in genomic selection (Xu et al., 2021). In parallel to this work, efficient approaches for the multiplication of matrices of mixed input data types have been suggested for use in transformer machine learning models, in particular with sub-byte integer data formats (Kim et al., 2022).

In this study, we present tailored algorithms for the multiplication of a SNP matrix by a real-valued matrix for CPUs and investigate the benefits of offloading this bottleneck to Nvidia® GPUs. Extending the Nvidia® CUTLASS library (Thakkar et al., 2023), our GPU approach benefits from fast tile iterators in the matrix multiplication. We provide our implementation as part of the software package *miraculix* (https://github.com/alexfreudenberg/miraculix) with bindings for Fortran, R and Julia. Since our CPU code is optimized for all major instruction set architectures, it is expected to yield performance increases across a variety of instruction set architectures. However, we also provide specifications that explicitly take advantage of the AVX2 and AVX512 architectures. Through the modular structure of the *miraculix* library, which also supplies functions for the calculation of $\mathbf{G}$, the code should be easily modifiable by researchers and practitioners interested in accelerating computations in other BLUP models or other genomic analyses.

We demonstrate how these advances can drastically reduce the computation times of genotype matrix multiplications on CPUs and GPUs compared to double-precision matrix multiplication routines provided by the Intel® MKL library, while simultaneously reducing memory requirements. Additionally, using genomic data provided by the Irish Cattle Breeding Federation and the genetic evaluation software MiXBLUP (ten Napel et al., 2021), we show how our novel approach brings down total run times in the estimation of genetic effects by several hours, thereby paving the way to include even larger population sizes in genomic evaluations.

2 Method

To understand the role of an efficient genotype matrix multiplication in the estimation of ssGBLUP models, we start with an introduction to the equation systems associated with the ssSNPBLUP and ssGTABLUP models. Then, we explain how the multiplication can be decomposed to reduce the theoretical complexity involved. Lastly, we propose novel techniques for the multiplication of a SNP matrix with a double-precision matrix on CPUs and GPUs.

2.1 Equation systems for single-step models

First, we introduce a general notation to describe the ssSNPBLUP approach, proposed by Liu et al. (2014). Second, we outline the ssGTABLUP technique proposed by Mäntysaari et al. (2017). Both approaches can be extended to multivariate models and have similar computational costs per iteration for large datasets (Vandenplas et al., 2023).

A standard univariate mixed model for ssGBLUP can be written as:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \begin{bmatrix} \mathbf{W}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_g \end{bmatrix} \begin{bmatrix} \mathbf{u}_n \\ \mathbf{u}_g \end{bmatrix} + \mathbf{e}$$

where the subscripts g and n refer to n_g genotyped and n_n non-genotyped animals, respectively, $\mathbf{y}$ is the vector of records, $\mathbf{b}$ is the vector of n_{fixed} fixed effects, $\mathbf{u}_n$ is the vector of additive genetic effects for the non-genotyped animals, $\mathbf{u}_g$ is the vector of additive genetic effects for the genotyped animals, and $\mathbf{e}$ is the vector of residuals. The matrices $\mathbf{X}$, $\mathbf{W}_n$, and $\mathbf{W}_g$ are incidence matrices relating records in $\mathbf{y}$ to the corresponding effects. The random effects vector $\mathbf{u}_g$ can be decomposed into $\mathbf{u}_g = \mathbf{a}_g + \mathbf{Z}\mathbf{g}$, where $\mathbf{g}$ is the vector of SNP effects and $\mathbf{a}_g$ contains the residual polygenic effects. The matrix $\mathbf{Z}$ is the centered genotype matrix, i.e., $\mathbf{Z} = \mathbf{M} - 1_{n_\text{g}}\mathbf{p}'$ for the uncentered genotype matrix $\mathbf{M} \in \{0, 1, 2\}^{n_\text{g} \times n_\text{s}}$ containing the n_s SNP genotypes (coded as 0 for one homozygous genotype, 1 for the heterozygous genotype, or 2 for the alternate homozygous genotype) of n_g genotyped animals, centered by their observed means $\mathbf{p}' = 1'_{n_\text{g}}\mathbf{M}$.

The ssSNPBLUP system of equations following Liu et al. (2014) can be written as:

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'_n\mathbf{R}_n^{-1}\mathbf{W}_n & \mathbf{X}'_g\mathbf{R}_g^{-1}\mathbf{W}_g & \mathbf{0} \\ \mathbf{W}'_n\mathbf{R}_n^{-1}\mathbf{X}_n & \mathbf{W}'_n\mathbf{R}_n^{-1}\mathbf{W}_n + \mathbf{\Sigma}^{11} & \mathbf{\Sigma}^{12} & \mathbf{\Sigma}^{13} \\ \mathbf{W}'_g\mathbf{R}_g^{-1}\mathbf{X}_g & \mathbf{\Sigma}^{21} & \mathbf{W}'_g\mathbf{R}_g^{-1}\mathbf{W}_g + \mathbf{\Sigma}^{22} & \mathbf{\Sigma}^{23} \\ \mathbf{0} & \mathbf{\Sigma}^{31} & \mathbf{\Sigma}^{32} & \mathbf{\Sigma}^{33} \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{u}_n \\ \mathbf{u}_g \\ \mathbf{g} \end{bmatrix} = \tag{3}$$

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{W}'_n\mathbf{R}_n^{-1}\mathbf{y}_n \\ \mathbf{W}'_g\mathbf{R}_g^{-1}\mathbf{y}_g \\ \mathbf{0} \end{bmatrix},$$

where the matrix $\mathbf{R}^{-1} = \begin{bmatrix} \mathbf{R}_n^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_g^{-1} \end{bmatrix}$ is the inverse of the residual (co)variance structure matrix. The matrix $\mathbf{\Sigma}^{-1}$ is defined as

$$\begin{aligned}\mathbf{\Sigma}^{-1} &= \begin{bmatrix} \mathbf{\Sigma}^{11} & \mathbf{\Sigma}^{12} & \mathbf{\Sigma}^{13} \\ \mathbf{\Sigma}^{21} & \mathbf{\Sigma}^{22} & \mathbf{\Sigma}^{23} \\ \mathbf{\Sigma}^{31} & \mathbf{\Sigma}^{32} & \mathbf{\Sigma}^{33} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}^{nn} & \mathbf{A}^{ng} & \mathbf{0} \\ \mathbf{A}^{gn} & \mathbf{A}^{gg} + \left(\frac{1}{w} - 1\right)\mathbf{A}_{gg}^{-1} & -\frac{1}{w}\mathbf{A}_{gg}^{-1}\mathbf{Z} \\ \mathbf{0} & -\frac{1}{w}\mathbf{Z}'\mathbf{A}_{gg}^{-1} & \frac{1}{w}\mathbf{Z}'\mathbf{A}_{gg}^{-1}\mathbf{Z} + \frac{m}{1-w}\mathbf{I} \end{bmatrix}\sigma_u^{-2},\end{aligned}$$

where the submatrices $\mathbf{A}^{nn}, \mathbf{A}^{ng}, \mathbf{A}^{gn}, \mathbf{A}^{gg}, \mathbf{A}_{gg}^{-1}$ are defined as in Equations (1) and (2). The scalar σ_u^{-2} is the inverse of the additive genetic variance, $w \in (0, 1)$ is the proportion of variance due to additive genetic effects, which is considered as residual polygenic effects, and, finally, $m = 2\sum_{j=1}^{n_s} p_j\,(1 - p_j)$ with $\mathbf{p} = (p_1, ..., p_{n_s})'$ being the observed allele frequencies.

The component-wise ssGTABLUP system of equations (Mäntysaari et al., 2020; Vandenplas et al., 2023) can be written as:

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}_n'\mathbf{R}_n^{-1}\mathbf{W}_n & \mathbf{X}_g'\mathbf{R}_g^{-1}\mathbf{W}_g \\ \mathbf{W}_n'\mathbf{R}_n^{-1}\mathbf{X}_n & \mathbf{W}_n'\mathbf{R}_n^{-1}\mathbf{W}_n + \mathbf{\Sigma}^{11} & \mathbf{\Sigma}^{12} \\ \mathbf{W}_g'\mathbf{R}_g^{-1}\mathbf{X}_g & \mathbf{\Sigma}^{21} & \mathbf{W}_g'\mathbf{R}_g^{-1}\mathbf{W}_g + \Omega \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{u}_n \\ \mathbf{u}_g \end{bmatrix} = \tag{4}$$
$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{W}_n'\mathbf{R}_n^{-1}\mathbf{y}_n \\ \mathbf{W}_g'\mathbf{R}_g^{-1}\mathbf{y}_g \end{bmatrix},$$

for $\Omega = \mathbf{\Sigma}^{22} - \mathbf{\Sigma}^{23}\left(\mathbf{\Sigma}^{33}\right)^{-1}\mathbf{\Sigma}^{32}$.

2.2 Computational bottlenecks

Due to large population sizes in breeding programs, equation systems (3) and (4) need to be solved iteratively in practice, commonly through the PCG algorithm. To this end, each iteration requires multiplying the corresponding coefficient matrix with a candidate vector, in the case of univariate models, or a candidate matrix in the case of multivariate models.

To avoid the explicit construction of the ssSNPBLUP coefficient matrix when multiplying it by a matrix it can be decomposed into submatrices. Importantly, most of the submatrices in Equations (3) and (4) can be multiplied by a vector or a matrix at a relatively low cost by using iteration-on-data techniques and sparse matrix operations (Vandenplas et al., 2020). In contrast, the multiplication of $\mathbf{Z}$ by a matrix of low width $\mathbf{\Lambda}$ has been a computational bottleneck so far.

To address this problem, we first note that the operation can be reformulated into

$$\mathbf{Z\Lambda} = \mathbf{M\Lambda} - 1_{n_g}\mathbf{p}'\mathbf{\Lambda},$$

where, again, $\mathbf{M} \in \{0,1,2\}^{n_\mathrm{g} \times n_\mathrm{s}}$ is the uncentered SNP matrix, and $\mathbf{p}$ is the vector of allele frequencies. Since the subtraction of $1_{n_\mathrm{g}}\mathbf{p}'\mathbf{\Lambda}$ consists of a vector-matrix multiplication and subsequent rank-one updates, it is cheap computationally. The operation $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$ for a low-width matrix $\tilde{\mathbf{\Lambda}}$ is decomposed similarly.

Due to the low cost of genotyping, the matrix $\mathbf{M}$ can have extremely large dimensions, capturing the genomic information of millions of animals. Furthermore, whereas $\mathbf{\Lambda}$ is a regular matrix of double-precision, $\mathbf{M}$ is usually stored in compressed format to save memory. For instance, the PLINK 1 binary file format stores the genotypes of four individuals in one byte, utilizing that one entry of $\tilde{\mathbf{Z}}$ only requires 2 bits of storage. This compressed data format prevents naive calls to BLAS routines, and decompressing is inefficient and increases memory requirements. More generally, the problem of matrix multiplication of mixed input data types has only recently gained attention (Kim et al., 2022) and no off-the-shelf solution exists for compressed 2-bit integer data types. Vandenplas et al. (2020) propose a decompress-on-the-fly approach: Unpacking tiles of submatrices of $\mathbf{M}$ small enough to store the result in L1 cache and perform matrix multiplication on these tiles. Thanks to the fast access times of the L1 cache, SNP data is readily at hand when new tiles of $\mathbf{\Lambda}$ are loaded. Furthermore, data movement along the cache hierarchy is kept at a minimum.

2.3 Acceleration of the genotype matrix multiplication

2.3.1 The *5codes* algorithm

Though slightly more sophisticated, our novel approach for CPU computations aims to reduce data streams through the cache hierarchy. Since missing entries can be corrected for at low cost by imputation or during postprocessing, they are coded as 0 at this point. We view the problem of storing SNP data through the lens of combinatorics: Since there are 3 possible states $\{0,1,2\}$ a SNP can assume, a vector $z \in \{0,1,2\}^5$ of 5 contiguous SNPs can assume $3^5 = 243$ values. Hence, each realized vector can be stored in one 8-bit unsigned integer while preserving the order of the SNPs. During preprocessing, we convert the input data to this even more compressed format, which we coined *5codes*. At multiplication time, we load a vector $\lambda \in \mathbb{R}^5$ of 5 entries of $\mathbf{\Lambda}$ in double precision and store all possible results of the scalar product

$$\mathbf{x} = \mathbf{z}'\lambda, \quad \text{for } \mathbf{z} \in \{0,1,2\}^5$$

in a hash table in double precision. Since each hash table requires $3^5 = 243$ times 64 bits, at least 4 of them fit into the L1 cache, corresponding to a scalar product of vectors of length 20. Now we iterate over the first 5 columns of $\tilde{\mathbf{Z}}$, loading all realizations of $\mathbf{z}$ in $\tilde{\mathbf{Z}}$ into the CPU registers and consequently looking up the corresponding scalar product $\mathbf{x}$ in the hash table.

Since the different values of $\mathbf{x}$ only need to be computed once for every five-row tile of $\mathbf{\Lambda}$, we keep register instructions to a minimum. The computation is parallelized among the available processor cores by dividing $\tilde{\mathbf{Z}}$ and $\mathbf{\Lambda}$ into

chunks along the column axis and row axis respectively and using a single reduce-operation in the end.

As discussed above, genotype centering is no bottleneck thanks to its low complexity. Thanks to the structure of the *5codes* encoding, the centering can in fact be included in the hash table in this algorithm for $\tilde{\mathbf{Z}}\mathbf{\Lambda}$, which further reduces the number of memory accesses and decreases accumulation errors in precision.

2.3.2 GPU implementation

To leverage the powerful HPC capabilities of modern Nvidia® GPUs, we also implement a matrix multiplication routine in CUDA, in which we extend the CUTLASS library (Thakkar et al., 2023) for high-performance matrix operations in its version 2.10. Specifically, we employ the Single Instruction Multiple Threads (SIMT) subsection within CUTLASS's warp-level API to distribute scalar products of four-dimensional vectors to the cores of the GPU. We keep the established PLINK 1 byte-sized storage format for SNP four-dimensional vectors while introducing a new CUTLASS-compatible interleaved data type for double-precision vectors of size four. Furthermore, fitting the genotype matrix multiplication into the CUTLASS framework requires adding a new scalar-product microkernel for this specific combination of data types and adjusting the CUTLASS interfaces upstream accordingly. Through the highly efficient memory access iterators in CUTLASS, we are able to move data quickly from the device memory to the shared memory to the cores and back. Furthermore, in order to reduce memory allocations and data movement between the host and the device, we preallocate memory for the matrix $\mathbf{\Lambda}$ and transfer data objects which are required in every PCG iteration ($\mathbf{Z}$ and $\mathbf{p}$) at start-up time. To our knowledge, this is the first implementation of matrix multiplication of 2-bit integers with double-precision floating-point values, which we designed in parallel to the recent work of Kim et al. (2022) extending the CUTLASS library to include, among others, matrix multiplication of 4-bit integers with half-precision floating-point values.

2.3.3 Memory management

Since both the CPU and GPU approaches exploit a compressed storage format for the SNP matrix, the question arises of how to efficiently calculate the transposed matrix product $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$. A memory-efficient implementation would transpose chunks of $\tilde{\mathbf{Z}}$ of low dimension, e.g. 16 by 16 for the 2-bit compression, which are iterated over. Thanks to the compressed data storage, however, we are not memory-bound with the current number of genotyped animals and SNPs. For instance, the dataset of 2.61mn animals with 47k SNP markers we use in this article for testing purposes only requires about 57 gigabytes of random access memory. Therefore, we choose to transpose $\tilde{\mathbf{Z}}$ as a whole during start-up and store it separately to reduce computation time.

3 Results

Since our novel approaches take advantage of modern hardware architectures, it can be expected that they perform better compared to existing implementations. To examine this, we use simulated data to benchmark our concepts for genotype data of various sizes. Afterward, considering the ssSNPBLUP and ssGTABLUP systems of equations on a large population of Irish beef and dairy cattle, we compare the wall clock time to convergence required by the PCG implementation in the software MiXBLUP (ten Napel et al., 2021) when using our novel approach with the time required by the current implementation.

3.1 Computational improvements

Using the software suite PLINK (Chang et al., 2015), we simulated genotype data of various dimensions. Mimicking the population sizes of many breeding programs in practice, we generated genotype data of three distinct animal populations with a varying number of individuals: "Small" with 102k animals, "Medium" with 751k animals and "Large" with 3.1mn animals. For each population, 50,241 SNPs were simulated, resulting in memory requirements of approx. 1.2 GB in compressed storage format (38.2 GB in double-precision) for the small population, 8.8 GB (281.2 GB) for the medium population, and 36.3 GB (1160.6 GB) for the large population. Furthermore, we simulated a phenotype matrix of 10 independently, normally distributed traits. Using this data, we test our implementation of the *5codes* algorithm and the GPU implementation.

In Figure 1, we assess the performance of consecutively multiplying $\mathbf{Z\Lambda}$ and $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$, as is required in each iteration of a PCG solver. For compilation, we used the Intel® compiler in its version 2021.4.0 and employed compilation options to natively optimize our code to the available hardware. The efficiency of our proposed approaches is compared against the technique in Vandenplas et al. (2020) and a naive call to the Intel® MKL library. As for the latter, we first inflate the compressed genotype data to double precision, store the matrix together with its transpose and call the `dgemm` routine afterward. We ran our CPU benchmarks on a single AMD® Milan EPYC 7513 CPU and our GPU implementation on an Nvidia® H100. Since none of the evaluated ideas displayed considerable scalability advances beyond 20 cores (see Table 1), we refrain from testing them on more cores.

Evaluating our methodology against a crude call to the BLAS routine shows that computation times can be reduced by more than 99.7% (98.1%) for the small and medium population on the GPU (CPU). Unfortunately, the genotype data of the large population in double precision required approx. 2.27 TB, which could not be met in our hardware setup, hindering us from benchmarking the MKL library on this dataset. Similarly, the previous technique for multiplying compressed SNP matrices with phenotype data proposed by Vandenplas et al. (2020) took about 19 times (3 times) longer on all population sizes on the GPU (CPU).

To mitigate the impact of hardware-specific optimizations which might limit

reproducibility of our results, we further test our CPU code on Intel® and AMD® processors (Intel® Xeon Gold 6230 and AMD® Milan EPYC 7513) and our GPU code on three generations of Nvidia® datacenter GPUs (Volta V100, Ampere A100 and Hopper H100). Results displayed in Table 1 are the average computation times of 10 replicates.

As for the GPU implementation, our findings suggest that performance increases of ca. 25% (50%) can be achieved when using the recently introduced H100 compared to the A100 (V100). More importantly, the V100 ships with 16GB or 32GB of device RAM, limiting the potential problem sizes in our application. Yet, this issue could be mitigated by using multiple GPUs. Since both the transposed and the untransposed matrix are stored on the device, computation times for both multiplications do not differ substantially.

Despite using the Intel® icc compiler with Intel-specific optimizations, the *5codes* algorithm performs slightly better on the more powerful AMD® chip. We observe that our implementation scales reasonably well to 10 cores, decreasing computation times by at least a factor of 5. However, increasing the number of cores to 20 only yields mild performance improvements and even comes with a penalty in some cases.

Overall, our evaluations suggest that the GPU implementation significantly outperforms the *5codes* implementation, though practical applications should evaluate costs and benefits of adding a GPU to their genomic evaluations based on their respective compute time restraints.

3.2 Evaluation on cattle data

3.2.1 Dataset

We tested our methodology using data from the routine six-trait calving-difficulty evaluation for Irish dairy and beef cattle performed by Irish Cattle Breeding Federation (ICBF; Ireland) in March 2022, as in Vandenplas et al. (2023). The single-step genomic evaluations were based on the same multi-trait animal model and variance components as the current official routine breeding value evaluation described in more detail in Evans et al. (2019) and Vandenplas et al. (2023). Briefly, after extraction and editing, the data file included 16.59mn data records (across 6 traits), and the pedigree included 26.46mn animals. The genotypes of 2.61mn animals included 47,006 SNP markers from 29 bovine autosomes, with a minor allele frequency greater or equal to 0.01. For both single-step approaches, the genotype matrix was centered using observed allele frequencies. The genotype data was from a range of 30 different arrays ranging from 3k to 850k SNPs. Missing SNP genotypes were imputed using FImpute (Sargolzaei et al., 2014) to a 50k SNP set based on version 3 of the IDB chip.

3.2.2 Experiments

We solved the equation systems associated with the ssSNPBLUP and ssGTABLUP models with the program hpblup, a PCG-based solver used by the software

MiXBLUP 3.1 (ten Napel et al., 2021), which links against the *miraculix* library and toggles the use of our novel implementation through an option. The RPG effect was set to a proportion of $w = 0.20$. Experiments were performed with 180 GB of RAM on a single AMD® EPYC 7513 CPU for the *5codes* algorithm and a single Nvidia® A100 for the GPU implementation. For the CPU tests, we used 15 dedicated cores, as this is the recommended setting in MiXBLUP. The PCG was iterated until a relative error below 10^{-13} was achieved. To put our observed computation times into perspective, we also solved both models with the previous approach for multiplying genotype matrices implemented in MiXBLUP 3.1.

Results are displayed in Table 2. Due to the different nature of the effects estimated, the computing times should be evaluated separately for the two models. For the ssSNPBLUP equation system, we observe that the average wall clock time is reduced by approx. 72% (11%) for the multiplication $\mathbf{Z\Lambda}$ and by approx. 95% (76%) for $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$ on GPUs (CPUs) respectively. When solving the ssGTABLUP equation system, the average time for multiplying $\mathbf{Z\Lambda}$ was slightly higher in the *5codes* algorithm. Yet, both *5codes* and the GPU implementation significantly decreased the time for computing $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$.

As the genotype matrix multiplication constitutes a significant portion of the time per iteration in the PCG solver, the total time for solving the two models was reduced from approx. 6.47 hours (4.03 hours) to 2.49 hours (1.98 hours) on the GPU and to 4.22 hours (3.30 hours) on the CPU for the ssSNPBLUP (ssGTABLUP). A similar number of iterations was observed for solving all models, though the *5codes* algorithm required 27 iterations less in the ssSNPBLUP, which might be explained by its inherently higher precision.

4 Discussion

We have presented novel concepts for multiplying centered genotype matrices by real-valued matrices which are applicable both on CPUs as well as modern GPUs. Useful applications include single-step BLUP models that are used to compute breeding value estimates when only a subset of animals are genotyped and/or phenotyped. Yet, similar computational operations are employed in seemingly unrelated fields of modern genetics. For instance, genome-wide association studies could benefit from a fast genotype matrix multiplication at various computational bottlenecks: the multiplication of the SNP matrix by a phenotype vector is an essential part of the calculation of genotype-phenotype correlations (Yang et al., 2011). Additionally, many genome-wide association studies use a PCA of $\mathbf{G}$ for population stratification (Price et al., 2006; Meuwissen et al., 2017; Ødegård et al., 2018) (as realized, for example, in the software suite GCTA (Yang et al., 2011)) and hence are to gain from a fast genotype matrix multiplication as well.

Through our optimized algorithms we were able to achieve a speed-up of critical operations by a factor of ca. 3 compared to the methodology by Vandenplas et al. (2020) using CPUs, and a factor of ca. 20 using GPUs. Thanks to this

acceleration, we have shown how our software can be used by researchers and practitioners to estimate breeding values in a population of 26.46mn animals, 2.61mn of which were genotyped, in a reasonable time of approx. 2 hours.

Nevertheless, the growth of breeding populations as well as the steadily falling costs of genotyping will result in genomic datasets of ever-growing size. Therefore, there are several avenues for further research to utilize computing resources even more efficiently.

First, as indicated in Section 2, system memory requirements might be reduced by a factor of approximately 2 by transposing the compressed genotype matrix on-the-fly during matrix multiplication instead of storing the transpose separately. With Nvidia® GPUs currently limited to at most 94 GB device memory and most compute set-ups limited to hundreds of gigabytes of RAM, this would extend the dimensions of possible problem sizes addressable with our proposed vector-multiplication microkernel.

Second, though our GPU implementation uses highly efficient data access iterators provided by the CUTLASS library, a further reduction in computation time might be achieved by using warp-level-coordinated matrix operations, which have been added as hardware instructions on the latest generations of GPUs (see, e.g., the CUTLASS documentation for how this has been tackled in general tensor-tensor operations (Thakkar et al., 2023)). Similarly, we have seen that our *5codes* implementation suffers from scalability issues when extending the number of cores.

Notwithstanding these potential improvements, our software can be used in a variety of computational tasks in genomics, e.g., ssSNPBLUP, to reduce computation times.

Tables and Figures

Figure 1: Total time for consecutively calculating $\mathbf{Z\Lambda}$ and $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$ on a AMD® Milan EPYC 7513 processor with 20 dedicated cores and a single Nvidia® H100 GPU. No test was performed for the BLAS-routine in Intel® MKL with the large population size, as this would have required ca. 2.27 TB of RAM. Results are the average of 10 repeated calculations.

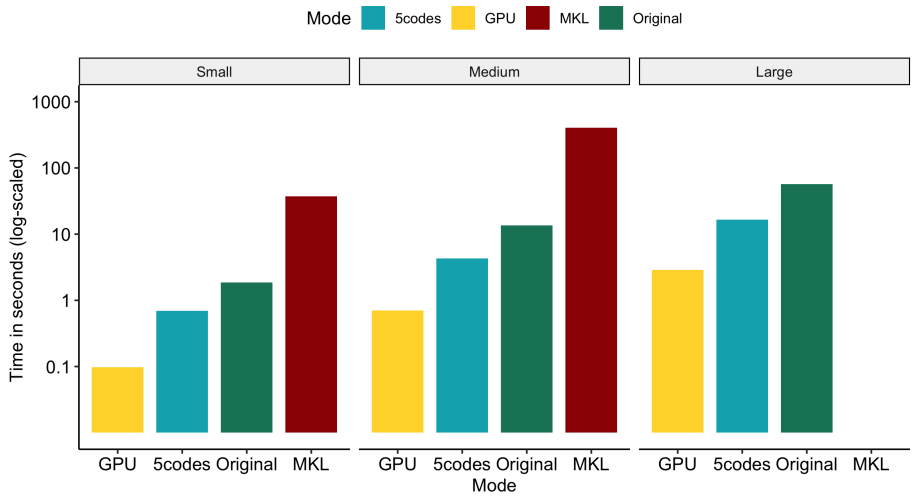

Table 1: Computation times in seconds of the GPU implementation and the *5codes* algorithm for the genotype matrix multiplication with simulated data on different hardware and varying shared-memory parallelism for CPUs (1 core, 10 cores and 20 cores). Dashes (-) indicate out-of-memory events. Results are the average of 10 repeated calculations.

Wall clock time (s)		**Nvidia®**			**Intel®**			**AMD®**		
		V100	**A100**	**H100**	**1c**	**10c**	**20c**	**1c**	**10c**	**20c**
Small	$\mathbf{Z\Lambda}$	0.10	0.06	0.05	2.24	0.36	0.35	1.71	0.38	0.41
population	$\mathbf{Z}'\tilde{\mathbf{\Lambda}}$	0.10	0.06	0.05	2.29	0.37	0.29	1.75	0.30	0.28
Medium	$\mathbf{Z\Lambda}$	0.69	0.45	0.35	16.03	2.61	2.51	12.53	2.96	2.94
population	$\mathbf{Z}'\tilde{\mathbf{\Lambda}}$	0.70	0.47	0.36	16.81	2.50	1.81	13.00	2.08	1.33
Large	$\mathbf{Z\Lambda}$	-	1.87	1.42	76.42	12.31	12.24	51.72	10.99	11.03
population	$\mathbf{Z}'\tilde{\mathbf{\Lambda}}$	-	1.94	1.47	83.44	12.73	10.04	53.68	8.36	5.55

Table 2: Computation times of single-step BLUP models on ICBF cattle data in seconds. All computations were performed on one AMD Milan EPYC 7513 CPU with 15 dedicated cores. GPU computations were performed on a single Nvidia® A100 GPU with 80 GB device memory. Both models were trained to a relative error below 10^{-13}. Total time contains preprocessing incl. set-up of the preconditioner matrix, solving of the MME, and postprocessing.

Wall clock time (s)	**ssSNPBLUP**			**ssGTABLUP**		
	Current	**5codes**	**GPU**	**Current**	**5codes**	**GPU**
Average time $\mathbf{Z\Lambda}$	5.41	4.81	1.50	8.76	10.38	1.53
Average time $\mathbf{Z}'\tilde{\mathbf{\Lambda}}$	29.68	7.05	1.62	28.66	6.83	1.58
Average time per iteration	51.43	32.32	14.16	50.38	35.21	16.25
Number of iterations	389	362	385	212	212	212
Preprocessing time	2953	3191	3186	3493	4068	3184
Total time	23275	15195	8961	14504	11863	7138

Conflict of Interest Statement

MiXBLUP is developed and marketed by the Animal Breeding and Genomics group at Wageningen UR Livestock Research, of which JV, TP and JtN are employees. The cattle data used in this study is proprietary and the intellectual property of the Irish Cattle Breeding Federation, at which RE is an employee.

Author Contributions

MS designed and implemented the *5codes* algorithm. AF designed and implemented the GPU solution. AF, JV and MS implemented interfaces to the algorithms. AF and JV wrote and ran analyses for this script. RE preprocessed and curated the ICBF cattle data. TP and JtN provided valuable suggestions for the experiment design and the focus of this study. All authors contributed to the article and approved the submitted version.

Funding

Analyses in this article were performed on the HPC systems bwUniCluster and Helix funded by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. AF acknowledges financial support from the Research Training Group "Statistical Modeling of Complex Systems" funded by the German Science Foundation. The publication of this article was funded by the University of Mannheim.

Data Availability Statement

The code in the *miraculix* package is released open-source under the Apache 2.0 license and is freely available on GitHub (https://github.com/alexfreudenberg/miraculix). The software *MiXBLUP* and the included solver *hpblup*, integrating the *miraculix* package, is commercially available at the MiXBLUP website (https://www.mixblup.eu/). Scripts for simulating the datasets used in this study can be found in the *miraculix* repository.

Individual genotype, pedigree and phenotype data used in this study are managed by the Irish Cattle Breeding Federation and cannot be made publicly available. Reasonable requests for the cattle data can be made to the Irish Cattle Breeding Federation (email: query@icbf.com; website: https://www.icbf.com/). Requests will be considered on a case-by-case basis, subject to approval by the data owner and compliance with any legal and ethical requirements related to data access and use.

References

Alappat, C., Basermann, A., Bishop, A. R., Fehske, H., Hager, G., Schenk, O., et al. (2020). A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. *ACM Trans. Parallel Comput.* 7. doi:10.1145/3399732

Chang, C. C., Chow, C. C., Tellier, L. C., Vattikuti, S., Purcell, S. M., and Lee, J. J. (2015). Second-generation PLINK: rising to the challenge of larger and richer datasets. *GigaScience* 4. doi:10.1186/s13742-015-0047-8. S13742-015-0047-8

Evans, R., Cromie, A., and Pabiou, T. (2019). Genetic evaluations for dam-type specific calving performance traits in a multi-breed population. In *Book of Abstracts of the 70th Annual Meeting of the European Federation of Animal Science*, ed. E. S. et al. 468

Kim, Y. J., Henry, R., Fahim, R., and Hassan, H. (2022). Who says elephants can't run: Bringing large scale MoE models into cloud scale production. In *Proceedings of The Third Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, ed. A. F. et al (Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics), 36–43

Legarra, A., Aguilar, I., and Misztal, I. (2009). A relationship matrix including full pedigree and genomic information. *Journal of Dairy Science* 92, 4656–4663. doi:10.3168/jds.2009-2061

Legarra, A. and Ducrocq, V. (2012). Computational strategies for national integration of phenotypic, genomic, and pedigree data in a single-step best linear unbiased prediction. *Journal of Dairy Science* 95, 4629–4645. doi:10.3168/jds.2011-4982

Liu, Z., Goddard, M. E., Reinhardt, F., and Reents, R. (2014). A single-step genomic model with direct estimation of marker effects. *Journal of Dairy Science* 97, 5833–5850. doi:10.3168/jds.2014-7924

Mäntysaari, E. A., Evans, R. D., and Strandén, I. (2017). Efficient single-step genomic evaluation for a multibreed beef cattle population having many genotyped animals. *Journal of Animal Science* 95, 4728–4737

Meuwissen, T., Indahl, U., and Ødegård, J. (2017). Variable selection models for genomic selection using whole-genome sequence data and singular value decomposition. *Genetics Selection Evolution* 49, 94. doi:10.1186/s12711-017-0369-3

Misztal, I., Legarra, A., and Aguilar, I. (2009). Computing procedures for genetic evaluation including phenotypic, full pedigree, and genomic information. *Journal of Dairy Science* 92, 4648–55. doi:10.3168/jds.2009-2064

Misztal, I., Legarra, A., and Aguilar, I. (2014). Using recursion to compute the inverse of the genomic relationship matrix. *Journal of Dairy Science* 97, 3943–3952. doi:10.3168/jds.2013-7752

Misztal, I., Lourenco, D., Tsuruta, S., Aguilar, I., Masuda, Y., Bermann, M., et al. (2022). 668. how ssgblup became suitable for national dairy cattle evaluations. In *Proceedings of 12th World Congress on Genetics Applied to Livestock Production (WCGALP)*, ed. R. F. V. et al (Wageningen Academic Publishers), 2757–2760

Mrode, R. A. (2014). *Linear models for the prediction of animal breeding values* (Cabi)

Mäntysaari, E., Koivula, M., and Strandén, I. (2020). Symposium review: Single-step genomic evaluations in dairy cattle. *Journal of Dairy Science* 103, 5314–5326. doi:10.3168/jds.2019-17754

Ødegård, J., Indahl, U., Strandén, I., and Meuwissen, T. H. (2018). Large-scale genomic prediction using singular value decomposition of the genotype matrix. *Genetics Selection Evolution* 50, 1–12

Price, A. L., Patterson, N. J., Plenge, R. M., Weinblatt, M. E., Shadick, N. A., and Reich, D. (2006). Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics* 38, 904–909

Sargolzaei, M., Chesnais, J. P., and Schenkel, F. S. (2014). A new approach for efficient genotype imputation using information from relatives. *BMC Genomics* 15, 478. doi:10.1186/1471-2164-15-478

Schaeffer, L. (2006). Strategy for applying genome-wide selection in dairy cattle. *Journal of Animal Breeding and Genetics* 123, 218–223

Strandén, I. and Lidauer, M. (1999). Solving large mixed linear models using preconditioned conjugate gradient iteration. *Journal of Dairy Science* 82, 2779–2787

Strandén, I., Matilainen, K., Aamand, G., and Mäntysaari, E. (2017). Solving efficiently large single-step genomic best linear unbiased prediction models. *Journal of Animal Breeding and Genetics* 134, 264–274. doi:10.1111/jbg.12257

ten Napel, J., Vandenplas, J., Lidauer, M. H., Strandén, I., Taskinen, M., Mäntysaari, E. A., et al. (2021). *MiXBLUP 3.0.1 manual*. Animal Breeding and Genomics, Wageningen University & Research, Wageningen, the Netherlands, v3.0 edn. Last accessed 15th of January 2023

Thakkar, V., Ramani, P., Cecka, C., Shivam, A., Lu, H., Yan, E., et al. (2023). *CUTLASS*

Vandenplas, J., Eding, H., Bosmans, M., and Calus, M. P. (2020). Computational strategies for the preconditioned conjugate gradient method applied to ssSNPBLUP, with an application to a multivariate maternal model. *Genetics Selection Evolution* 52, 1–10

Vandenplas, J., Eding, H., Calus, M. P., and Vuik, C. (2018). Deflated preconditioned conjugate gradient method for solving single-step BLUP models efficiently. *Genetics Selection Evolution* 50, 1–17

Vandenplas, J., ten Napel, J., Darbaghshahi, S. N., Evans, R., Calus, M. P., Veerkamp, R., et al. (2023). Efficient large-scale single-step evaluations and indirect genomic prediction of genotyped selection candidates. *Genetics Selection Evolution* , in press

Xu, Y., Laurie, J. D., and Wang, X. (2021). CropGBM: An ultra-efficient machine learning toolbox for genomic selection-assisted breeding in crops. In *Springer Protocols Handbooks* (Springer Protocols Handbooks). 133–150. doi:10.1007/978-1-0716-1526-3_5

Yang, J., Lee, S., Goddard, M., and Visscher, P. (2011). GCTA: a tool for genome-wide complex trait analysis. *American Journal of Human Genetics* 88, 76–82