Novel Techniques for Accelerating Statistical Operations on Compressed Genomic Data

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Alexander Freudenberg
aus München

Mannheim, 2023

Dekan:	Dr. Bernd Lübcke, Universität Mannheim
Referent:	Professor Dr. Martin Schlather, Universität Mannheim
Korreferent:	Professor Dr. Guido Moerkotte, Universität Mannheim

Tag der mündlichen Prüfung: 01.09.2023

Abstract

Over the last decades, the availability of genetic data has exploded and genomic information is widely used in a variety of fields today. While the cost of genotyping and sequence assembly has been steadily decreasing, software in quantitative genetics has been struggling to keep up with increasing computational demands. Many existing software solutions use strategies for shared-memory parallelism and instruction-level parallelism. However, partly due to a lack of suitable hardware instructions, the dissemination of software that utilizes accelerator hardware has been limited.

In this thesis, novel methods for the efficient processing of genomic data are presented. By utilizing low-precision integer instructions on modern NVIDIA® GPUs, the necessity to decompress SNP data for statistical evaluations is avoided. Due to the memory efficiency of compressed genomic storage formats, datasets of large populations with a high number of SNPs can be analyzed on a single datacenter GPU.

The benefits of these new techniques are demonstrated through examples of important quantities in quantitative genetics. First, it is shown that the analytical calculation of population statistics, such as the genomic relationship matrix or linkage disequilibrium, is significantly accelerated compared to existing methods. Second, the numerical evaluation of a single-step BLUP model is used to demonstrate that the use of accelerators can significantly reduce computing times required for estimating genetic values based on iterative-solver methods. Lastly, it is illustrated that the estimation of parameters for an important covariance model can be significantly improved.

Zusammenfassung

Über die letzten Jahrzehnten sind genetische Daten leicht verfügbar geworden und genetische Informationen werden heute in einer Vielzahl von Bereichen verwendet. Während die Kosten für Genotypisierung und Sequenzalignment stetig gesunken sind, hat die Software in der quantitativen Genetik Schwierigkeiten, mit den steigenden Rechenanforderungen Schritt zu halten. Viele bestehende Softwarelösungen nutzen Strategien für Shared-Memory-Parallelismus und Instruction-Level-Parallelismus. Teilweise aufgrund der fehlenden Hardware-Instruktionen ist die Verbreitung von Software, die Beschleuniger-Hardware nutzt, jedoch begrenzt.

In dieser Dissertation werden neue Methoden zur effizienten Verarbeitung von genomischen Daten vorgestellt. Durch die Nutzung von Low-Precision-Integer-Instruktionen auf modernen NVIDIA® GPUs wird die Notwendigkeit umgangen, SNP-Daten für statistische Auswertungen zu dekomprimieren. Aufgrund der Speichereffizienz von komprimierten genomischen Speicherformaten können dadurch Datensätze großer Populationen mit einer hohen Anzahl von SNPs auf einer einzelnen Datacenter-GPU analysiert werden.

Die Vorteile dieser neuen Techniken werden anhand von Beispielen wichtiger Statistiken in der quantitativen Genetik demonstriert. Erstens wird gezeigt, dass die analytis-

che Berechnung von Populationsstatistiken, wie der genomischen Verwandschaftsmatrix oder des Linkage Disequilibriums, im Vergleich zu bestehenden Methoden erheblich beschleunigt wird. Zweitens wird die numerische Auswertung eines Single-Step-BLUP-Modells verwendet, um zu demonstrieren, dass die Nutzung von Beschleunigern die erforderlichen Rechenzeiten für die Schätzung von genetischen Werten auf Basis iterativer Lösungsmethoden erheblich reduzieren kann. Zuletzt wird illustriert, dass die Schätzung von Parametern für ein wichtiges Kovarianzmodell signifikant verbessert werden kann.

Acknowledgements

During the work on this thesis, I had the pleasure of working with a number of great people to whom I wish to express my gratitude.

First of all, I would like to thank Martin Schlather for his supervision and guidance over the last few years. His passion for science and his critical questioning were a beacon of inspiration during this period and led me to my interest in efficient computations in the first place. Our collaboration in the development of the `miraculix` library led to the papers that form the foundation of this thesis.

Second, I am grateful to Leif Döring for setting me up toward a pathway in research early in my studies and for our collaboration on the statistical modeling of tumor cells in Venkataramani et al. (2022).

Furthermore, I thank the researchers with whom I had the privilege to collaborate: Guido Moerkotte, for deepening my knowledge in low-level programming and microprocessor architecture and for agreeing to assess this thesis. Martin Slowik, for dozens of fruitful discussions on statistical questions in a broad range of areas. Torsten Pook for answering my countless questions on genetic theory and allowing me to contribute to the scientific work in Pook et al. (2021). Jeremie Vandenplas for the myriad of valuable discussions on computational strategies in large-scale genomic evaluations and for his help in drafting one of the manuscripts which will be presented in this thesis. Varun Venkataramani and his colleagues from the University of Heidelberg, for giving me the opportunity to contribute to a fascinating project in cancer research, which seems to have struck a nerve in the community. A special thanks goes to Anja Gilliar for all the administrative support over the years.

In addition, I am grateful to my mother for her steady love and support in my life and for helping me with literature research. I thank my late grandmother, my grandfather, and my aunt who always encouraged and supported me.

I want to express my gratitude toward the full "Mensa Crew" for enjoyable extended lunch breaks and great times in the Café Soleil. Toward Minh Truong for having a sympathetic ear over all the years in Mannheim. And toward Ricarda Thilmann for her patience and appreciation.

Funding

I gratefully acknowledge financial support by the German National Academic Foundation and by the Research Training Group "Statistical Modeling of Complex Systems" funded by the German Science Foundation.
Analysis for this thesis was performed on the HoreKa compute cluster funded by the Ministry of Science, Research and the Arts Baden-Württemberg through bwHPC and the German Federal Ministry of Education and Research, on the Helix compute cluster funded by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, and on the bwUniCluster 2.0 compute cluster funded by the state of Baden-Württemberg through bwHPC.

Contents

Chapter 1

Introduction

Through technological advances, the field of genetics has seen unparalleled changes over the last decades. DNA sequencing has become dramatically cheaper, broadening the research and applications of the technology. Next-generation sequencing technologies have enabled genotyping at a reasonable price and even whole-genome sequencing is now regularly used in clinical studies (Schwarze et al., 2018). Yet, in large-scale genomic evaluations, genotyping is still mainly based on measuring the expression of single-nucleotide polymorphisms (SNPs) today, a common form of variation in the genome. SNPs describe a deviation at a single base pair of the genome. Commonly, only two possible alleles for the genotype of the base pair are considered in which case the SNP is called biallelic. Additionally, SNPs in which the less frequent allele falls short of a certain minimum frequency are excluded from the analysis. SNP arrays of different densities are available for genotyping, that is, the arrays detect a varying number of SNPs along the genome. In species with two sets of chromosomes, the SNP variation in an individual can be either on both chromosomes, on one chromosome or on none of the chromosomes. In quantitative genetics, SNPs are used to investigate associations between genes and phenotypes and should be thought of as an auxiliary tool for capturing the full genome (Mrode, 2014).

Augmenting the progress in genotyping technology, improvements in computational power have allowed researchers to process data faster and cheaper and to use available genomic data more effectively. On top of that, the developing field of bioinformatics has produced novel genomic technologies itself: For instance, it played a vital role in the development of the gene editing technology CRISPR-Cas (Alkhnbashi et al., 2020).

However, the development of efficient and scalable software solutions traditionally has not focused on the statistical analysis of genomic data outside of phylogenomics. Review studies have found that most endeavors to leverage the potential of modern computing architectures are concentrated on genome assembly tasks like sequence aligning. Graphics processing units (GPUs) have been used for tasks in sequence analysis (Taylor-Weiner et al., 2019; Krishna and Elisseev, 2021) and the hardware vendor NVIDIA® has developed proprietary software for some of these operations (O'Connell et al., 2023). On

the other hand, these review studies did not identify any software in quantitative and statistical genetics that is targeted at high-performance computing systems (Ocaña and de Oliveira, 2015; Shi and Wang, 2019). The de-facto standard software packages for association studies, PLINK (Chang et al., 2015) and GCTA (Yang et al., 2011), offer data parallelism and shared-memory parallelism strategies, but no scalability to multi-node architectures, despite their intended use in big data settings.
Partly due to a lack of hardware instructions, utilization of GPUs in statistical genomics has been scarce. In this thesis, some of the functionality of the software library `miraculix` is presented. The library comprises highly efficient algorithms for essential algebraic operations on compressed genomic data on both CPUs as well as GPUs. In particular, it avoids the need to decompress SNP data before performing mathematical operations on it by employing novel instructions on modern NVIDIA® GPUs. Due to the memory efficiency of compressed genomic storage formats, huge amounts of data can be processed on a single datacenter GPU. To the best of the author's knowledge, `miraculix` is the first software to perform operations on compressed genomic data on GPUs without a prior full decompression of the data. As will be shown later in this thesis, this functionality has the potential for a significant reduction in computing times in genomic evaluation pipelines, while simultaneously increasing energy efficiency significantly. Furthermore, the CPU implementation in `miraculix` substantially outperforms other popular libraries such as PLINK (Chang et al., 2015) and GCTA (Jiang and Reif, 2015).

1.1 Outline

This thesis is structured as follows. In Chapter 2, a summary of essential theory from statistical genetics is presented. It is mainly based on the foundational work by Falconer (1996) and Mrode (2014).
In Chapter 3, techniques for efficient multiplications of SNP matrices on GPUs are introduced. Approaches implemented in existing software solutions are discussed and the methodology for offloading this operation to NVIDIA® GPUs is presented. Its benefits for common statistical operations on genomic data are demonstrated. This chapter is based on the article

> Freudenberg, A., Schlather, M., Moerkotte, G., and Pook, T. (2023a). miraculix: Accelerated Computations for Genomic Analysis. *Manuscript submitted.* Freudenberg2023a

In Chapter 4, novel methods for multiplying SNP matrices with floating-point vectors are presented. Its advantages are discussed at the example of single-step models, a popular method for combining genomic data with pedigree data in large-scale genomic evaluations. The chapter is based on

> Freudenberg, A., Vandenplas, J., Schlather, M., Pook, T., Evans, R., and ten Napel, J. (2023c). Accelerated matrix-vector multiplications for matri-

ces involving genotype covariates with applications in genomic prediction. *Accepted for publication in Frontiers in Genetics.* Freudenberg2023c

and an extension to this manuscript is intended to be published in

Freudenberg, A., Schlather, M., Vandenplas, J., and Evans, R. (2023b). Accelerating single-step evaluations through GPU offloading. *Manuscript in preparation* Freudenberg2023b

Chapter 5 presents novel approaches for estimating parameters of the Whittle-Matérn covariance function which has been used to study polygenic effects in genomics. This chapter is based on unpublished work with Martin Schlather.

Chapter 2

Statistical models in genetics

Initially, some important concepts in the modeling of the relationship between genes and quantitative traits are described. This section is largely based on the foundational work by Falconer (1996). Though these approaches originated in animal breeding, they have swashed to other areas of quantitative genetics, e.g., heritability modeling in human genetics.

2.1 Quantitative trait modelling

The assemblage of genes in an individual is referred to as a genotype. Genetic theory assumes that the value of a quantitative phenotype $\mathbf{P}$ can be decomposed into the sum of the influence of the genotype $\mathbf{G}$ and an environmental deviation $\mathbf{E}$, i.e.,

$$\mathbf{P} = \mathbf{G} + \mathbf{E}.$$

The quantity $\mathbf{G}$ is coined genotypic value or genetic value. If it is assumed that only a single biallelic variation is causal for the trait, then the genotypic value is explained by two factors: The average effect $\mathbf{A}$ and the dominance deviation $\mathbf{D}$. In this case, the average effect is the theoretical average quantitative effect on the phenotypic trait of randomly substituting one allele at one chromosome set by the other possible allele. However, in practice, it is impossible to actually flip alleles without gene editing. Therefore, the concept of breeding values has been introduced: The breeding value is defined as twice the expected deviation in the phenotype of the offspring of an individual compared to the population average, where the multiplication by two is used to account for the fact that offspring only inherit half their genes from one parent. The breeding value for a single locus is then equal to the sum of the average effects of the two alleles that are present in the individual.
However, the realized genotypic value $\mathbf{G}$ might deviate from the sum of the average effects at this locus when the alleles on the two sets of chromosomes do not influence the trait independently but interact with each other. This is measured by the dominance

deviation term **D**. It is defined as the difference between the average effect and the actually realized impact on the phenotype.
If multiple causal variants are considered and it is assumed that alleles interact across loci when influencing a trait, then there is another effect: Interaction deviation **I** (also called epistatic deviation). Hence, the genotypic value **G** can be split into

$$\mathbf{G} = \mathbf{A} + \mathbf{D} + \mathbf{I}.$$

Due to computational challenges when considering the interactions between tens of thousands of SNPs, whose number grows quadratically with the number of variants, epistatic effects are often neglected in practice. Yet, they are sometimes studied when seeking to explain missing predictiveness of genotypic information (Hemani et al., 2013; Vojgani et al., 2021, 2023). In this thesis, the focus will lie on the analysis of additive effects.
When seeking to explain variation in a phenotypic value, its variance $V_{\mathbf{P}}$ is analyzed. To this end, it is commonly assumed that the variables **A**, **D**, **I** and **E** are uncorrelated, though genotype-environment interaction effects can be included in principle by assuming a non-zero covariance between **E** and **G**. The associated variances are termed additive variance $V_{\mathbf{A}}$, dominance variance $V_{\mathbf{D}}$, interaction variance $V_{\mathbf{I}}$ and environmental variance $V_{\mathbf{E}}$, while the sum of additive, dominance and interaction variances is called genotypic variance $V_{\mathbf{G}}$. These quantities are used to mathematically define the concept of heritability: Heritability in the broad sense is defined by the ratio of phenotypic variance explained by genotypic variance, i.e. $V_{\mathbf{G}}/V_{\mathbf{P}}$. Heritability in the narrow sense refers to the explained variance excluding dominance and interaction effects, $V_{\mathbf{A}}/V_{\mathbf{P}}$.

2.2 Statistical inference of genetic effects

Driven by the rising availability of genomic data over the last 20 years, a high volume of research has focused on improving the estimation of the aforementioned quantities in empirical settings. Additionally, the new area of genomic prediction has been established which focuses on predicting unobserved phenotypic values from genomic resemblance (Morota and Gianola, 2014). Chapter 11 in Mrode (2014) gives an overview of established models which are the basis of most statistical models used for inferring breeding values and genetic effects today. We give a brief summary of the relevant approaches.
Though whole-genome regression is becoming more popular, most quantitative genomic studies are based on the more cheaply available SNP arrays, which genotype genetic markers at different densities. In general, the genetic theory behind the use of SNPs assumes that they either directly influence phenotypic expression or are linked to other base pairs which are causal for this trait (Falconer, 1996; Mrode, 2014). Yet, the sub-par performance of genotypic data in explaining phenotypic variance has been attributed to imperfect linkage (Manolio et al., 2009; de los Campos et al., 2015). The theoretical concept of linkage between alleles during assemblage is specified through Linkage Disequilibrium (LD), which refers to the observed associations of the alleles at two base pairs. LD is measured in terms of haplotype covariances or haplotype correlations.

Due to the computational problems indicated above, it is common in animal breeding that only additive effects are of interest, while polygenic effects (i.e., effects caused by interactions between SNPs) are only included through an error term. Given a set of k SNP alleles $m_1, ..., m_k$, their effect on a quantitative phenotype y is modeled as

$$y = \mu + \sum_{i=1}^{k} z_i u_i + a + \varepsilon, \tag{2.1}$$

where μ is an intercept term, $z_i = m_i - 2p_i$ is the i-th SNP centered by its allele frequency p_i, u_i is the effect size of the i-th SNP, a is the residual polygenic effect and ε captures environmental deviation. The SNP values $m_1, ...m_k$ are coded as 0 for one homozygous genotype, 1 for the heterozygous genotype, or 2 for the alternate homozygous genotype. More specifically, if $m_i = 0$, then both sets of chromosomes in the individual carry the more frequent allele in the population, which is also referred to as the major allele. On the other hand, if $m_i = 2$, then both sets of chromosomes have the minor allele at the locus i, referring to the less frequent allele in the population. The value $p_i \in [0, 1]$ is the frequency of the minor allele across the population. In practical settings, the allele frequencies $p_1, ..., p_k$ are often not taken to be identical to the empirically observed frequencies in a population, as the non-random selection of individuals for genotyping would introduce selection bias. Rather, they are computed from a reference population or the empirical allele frequency is stratified to account for the population structure. While the error terms a and ε can be combined for the case where only genotyped individuals are considered, a separate treatment becomes important when also non-genotyped individuals are included.
Schreck (2018) and Schreck et al. (2019) investigate the caveats of different modeling approaches for the effect sizes $u_1, ..., u_p$. While genetic theory dictates that they are fixed and variation in genotypic values arises only from differences in the genotype, this postulation is difficult to account for in practice. If the number of phenotype records n is smaller than p or if SNPs are in perfect LD, the matrix $Z = (z_1, ..., z_k)$ does not have full rank and the best linear unbiased estimator $\hat{u}$ for the SNP effects $u = (u_1, ..., u_k)^T$ is not uniquely determined. Variable selection techniques are studied in genome-wide association studies (GWAS) to identify variants which are causal for the trait by themselves. However, when inferring genotypic values, these methods are not helpful, since evidence suggests that additive effects of quantitative trait loci (QTL) are infinitesimal for many phenotypes, i.e., the genotypic value is the sum of a high number of small effects (de los Campos et al., 2015).
Therefore, effect sizes are mostly modeled probabilistically in breeding value estimation today. In their seminal work, Meuwissen et al. (2001) suggest the use of a random effects model (REM) or, alternatively, one of two Bayesian models, BayesA and BayesB, which assume that effect sizes are drawn from a prior distribution. The REM and its augmentation, the Mixed Effects Model, which also models covariates with fixed effects, have been introduced into genetics by Fisher (1919). Both modeling approaches, the REM and Bayesian Models, are still actively used by researchers today. However, in

large-scale settings, the increased computational requirements of inferring parameters in Bayesian techniques make them impractical.
In contrast, the parameters β and u in the Mixed Effects Model can be estimated analytically: The standard Mixed Effects Model for the n-dimensional random vector y is given by

$$y = X\beta + Zu + \varepsilon, \tag{2.2}$$

where $X \in \mathbb{R}^{n\times l}, \beta \in \mathbb{R}^l, Z \in \mathbb{R}^{n\times k}$ and u, ε are k-dimensional and n-dimensional random vectors with existing covariance matrices $\sigma_u^2\Sigma_u, \sigma_\varepsilon^2\Sigma_\varepsilon$ for $\sigma_u^2, \sigma_\varepsilon^2 > 0$ respectively. The factors σ_u^2 and σ_ε^2 correspond to the additive variance and residual variance respectively and are coined variance components. If $\sigma_u^2\Sigma_u$ and $\sigma_\varepsilon^2\Sigma_\varepsilon$ are nonsingular, then the best linear unbiased estimator (BLUE) $\hat{\beta}$ for β and the best linear unbiased predictor (BLUP) $\hat{u}$ for u suffice the Mixed Model Equations (MME)

$$\begin{pmatrix} X^T\Sigma_\varepsilon^{-1}X & X^T\Sigma_\varepsilon^{-1}Z \\ Z^T\Sigma_\varepsilon^{-1}X & Z^T\Sigma_\varepsilon^{-1}Z + \frac{\sigma_\varepsilon^2}{\sigma_u^2}\Sigma_u^{-1} \end{pmatrix} \begin{pmatrix} \hat{\beta} \\ \hat{u} \end{pmatrix} = \begin{pmatrix} X^T\Sigma_\varepsilon^{-1}y \\ Z^T\Sigma_\varepsilon^{-1}y \end{pmatrix}, \tag{2.3}$$

as shown by Henderson (1949). At the same time, any vectors $\hat{\beta}, \hat{u}$ which suffice these equations are the BLUE and BLUP for β and u. Notice that this result holds without an assumption on the probability distribution of u and ε, though they are commonly assumed to be normally distributed in practice. The matrices Σ_u and Σ_ε are often assumed to be diagonal and the matrix on the left-hand side of the equation is commonly referred to as the coefficient matrix in the literature.
In genomics, $Z = (z_1, ..., z_k)$ holds the centered genotypes, and the intercept μ in Equation (2.1) has been replaced by the fixed effects term $X\beta$. The fixed effects are often used to explicitly model environmental effects on the phenotype. In some applications, only the so-called genomic values $g = Zu$ are of interest. In this case, the model becomes

$$y = X\beta + g + \varepsilon. \tag{2.4}$$

Formulation 2.2 is referred to as the SNP-BLUP model, whereas 2.4 is called the genomic BLUP (gBLUP) model. When Σ_ε and the covariance matrix $\sigma_g^2\Sigma_g$ of g are nonsingular, then the MME for 2.4 are

$$\begin{pmatrix} X^T\Sigma_\varepsilon^{-1}X & X^T\Sigma_\varepsilon^{-1} \\ \Sigma_\varepsilon^{-1}X & \Sigma_\varepsilon^{-1} + \frac{\sigma_\varepsilon^2}{\sigma_g^2}\Sigma_g^{-1} \end{pmatrix} \begin{pmatrix} \hat{\beta} \\ \hat{g} \end{pmatrix} = \begin{pmatrix} X^T\Sigma_\varepsilon^{-1}y \\ \Sigma_\varepsilon^{-1}y \end{pmatrix}. \tag{2.5}$$

The gBLUP model has the advantage that it can also be posed in the absence of genomic data. In this case, family relationships are used to construct the relationship matrix A, also called the numerator matrix. Starting from the oldest animals of consideration, this matrix is computed recursively by adding the degree of relationship between two animals to the off-diagonal, utilizing that a parent contributes half of their genes to

their offspring (see Chapter 2 in Mrode (2014)). This approximation of the genetic relationship between individuals in a population is then assumed to be the covariance matrix of g and replaces the genomic relationship matrix (GRM) G in the construction of the gBLUP model. The matrix A has the useful property that it is always symmetric positive definite and Henderson (1976) derived rules for constructing its inverse A^{-1} recursively as well. Starting from the oldest animals in the pedigree again, the matrix A^{-1} only has non-zero offdiagonal elements in the indices which correspond to the parents of an animal, though the values depend on the degree of inbreeding in the population. Hence, the inverse is highly sparse for large pedigrees.
In fact, this is the traditional approach which was popular before the broad availability of genomic data. VanRaden (2008) was the first to suggest the use of genomic data for approximating the number of shared alleles between individuals in the context of breeding value estimation. He reviewed methods for constructing the GRM G, two of which are still regularly used today. They assume that effect sizes are independent and normally distributed, implying that Σ_u is diagonal. The first method constructs the GRM G_1 as

$$G_1 = \frac{ZZ^T}{2p^T(1_k - p)},$$

where $p = (p_1, ..., p_k)^T$ is the vector of allele frequencies and $1_k = (1, ..., 1)$ is a vector of length k consisting of 1s. This method corresponds to Equation 2.2. Indeed, when using the new variance component $\sigma_g^2 = \frac{\sigma_u^2}{2p^T(1-p)}$, then

$$\text{Cov}(g) = \sigma_g^2 G_1.$$

The use of the constant $2p^T(1-p)$ was first suggested by Loiselle et al. (1995) as a way of scaling the covariance matrix ZZ^T uniformly to account for spatial genetic structure. VanRaden (2008) used this constant to scale the GRM to be "analogous to the numerator relationship matrix", which has generally been accepted in the literature.
The second method for constructing a GRM G_2 scales the columns of Z by their corresponding allele standard deviation before multiplication, i.e.,

$$G_2 = \left(ZD^{-1/2}\right)(ZD^{-1/2})^T,$$

where $D = \text{diag}(2k \cdot p_1 \cdot (1-p_1), ..., 2k \cdot p_k(1-p_k))$. This corresponds to a Mixed Effects Model, in which the genomic values are assumed as $g = ZD^{1/2}u$. In the case where $p_1 = ... = p_k$, both GRMs coincide, that is, $G_1 = G_2$. Though this method accounts for heteroscedasticity in the observed allele counts, it introduces numerical problems when the standard deviations are small and assumes that the theoretical variance of all alleles should be identical (Legarra et al., 2022).
Henderson (1963) showed that, assuming X is of full rank, the BLUE $\hat{\beta}$ and BLUP $\hat{u}$ are equal to

$$\hat{\beta} = \left(X^T\left(\sigma_\varepsilon^2\Sigma_\varepsilon + \sigma_u^2 Z\Sigma_u Z^T\right)^{-1}X\right)^{-1} X^T\left(\sigma_\varepsilon^2\Sigma_\varepsilon + \sigma_u^2 Z\Sigma_u Z^T\right)^{-1} y \qquad (2.6)$$

and

$$\hat{u} = \sigma_u^2 \Sigma_u Z^T \left(\sigma_\varepsilon^2 \Sigma_\varepsilon + \sigma_u^2 Z \Sigma_u Z^T\right)^{-1} (y - X\hat{\beta}) \tag{2.7}$$

respectively. Under the assumptions of the GRM G_1, the BLUP $\hat{g}$ for g after reordering of the variance components is given by

$$\hat{g} = G\left(G + \frac{\sigma_\varepsilon^2}{\sigma_g^2}\mathbb{1}_{n\times n}\right)^{-1}(y - X\hat{\beta}). \tag{2.8}$$

Similarly, the BLUE $\hat{\beta}$ can be expressed in terms of G through

$$\hat{\beta} = \left(X^T\left(G + \frac{\sigma_\varepsilon^2}{\sigma_g^2}\mathbb{1}_{n\times n}\right)^{-1}X\right)^{-1}X^T\left(G + \frac{\sigma_\varepsilon^2}{\sigma_g^2}\mathbb{1}_{n\times n}\right)^{-1}y. \tag{2.9}$$

For large-scale evaluations, which will be considered later in this thesis, the above model requires extensions to deal with incomplete information: First, although high-throughput phenotyping technologies have become available (Solberg et al., 2006), it is common that not all individuals in a population are phenotyped, e.g., in the case of sex-specific traits. Yet, they need to be included in breeding value estimation, since they are similarly subject to selection. In the mixed effects model, they are incorporated through an incidence matrix W, consisting of zeros and ones, which links each phenotype observation to its genomic value. The gBLUP model formulation then reads

$$y = X\beta + Wg + \varepsilon. \tag{2.10}$$

When SNP effects are of interest, it is possible to model residual polygenic effects a explicitly by replacing Wg by $W(Zu + a)$, where $a = (a_n^T, a_g^T)^T$ is a random vector with an existing covariance matrix.

Additionally, it is regularly the case that not all animals in a population are genotyped. Reasons for that include cost constraints, culling or slaughtering of animals or the inclusion of foreign animals, for which only pedigree data is available. Traditionally, a two-step approach has been used for breeding value estimation in this circumstance: First, the BLUP $\hat{g}$ is calculated for the genotyped subset of the population. From these predictors, the breeding values for non-genotyped animals was derived by using pedigree data. While this method is still occasionally used today, it has generally been accepted that it suffers from a number of drawbacks. For instance, accuracy errors in the first step of the estimation are amplified in the second step. Additionally, genotyping of animals is usually not performed at random, but rather a select group of fit animals are genotyped.

Therefore, the two-step approach leads to biased estimates and many evaluation centers have transitioned to a single-step approach introduced by Legarra et al. (2009) which combines the two relationship matrices A and G. They use genomic information when it is available and fall back to pedigree data otherwise, resulting in the matrix

$$\tilde{A} = \begin{pmatrix} A_{\mathrm{nn}} & A_{\mathrm{ng}} \\ A_{\mathrm{gn}} & G \end{pmatrix} = A + \begin{pmatrix} 0 & 0 \\ 0 & G - A_{\mathrm{nn}} \end{pmatrix}, \tag{2.11}$$

where $A_{\text{nn}}, A_{\text{ng}}, A_{\text{gn}}$ and A_{gg} denote the submatrices of A and where the indices non-genotyped and genotyped animals. However, this matrix can be indefinite and is therefore unsuitable for a covariance matrix.
Christensen and Lund (2010) enhanced this approach by assuming that the SNP values $m_1, ..., m_k$ are a realization of a normally distributed random variable with covariance matrix A. If the conditional probability distribution of the full genomic values g is additionally assumed to be of the form of a GRM, then the restricted conditional covariance matrix of the genomic values, where only the genotyped SNPs are known, is given by

$$\text{Cov}\left(\begin{pmatrix} g_n \\ g_g \end{pmatrix} \mid Z\right) = \sigma_g^2 \begin{pmatrix} A_{\text{gn}}A_{\text{gg}}^{-1}GA_{\text{gg}}^{-1}A_{\text{ng}} + A_{\text{nn}} - A_{\text{gn}}A_{\text{gg}}^{-1}A_{\text{ng}} & A_{\text{gn}}A_{\text{gg}}^{-1}G \\ GA_{\text{gg}}^{-1}A_{\text{ng}} & G \end{pmatrix} =: \sigma_g^2 \Sigma_g, \tag{2.12}$$

by using the formulae for the conditional probability distribution of the multivariate normal. By separately modeling residual polygenic effects a with mean zero and covariance matrix $\sigma_a^2 A$, they deduce the matrix H,

$$\begin{aligned} H =& \frac{1}{\sigma_h^2}\text{Cov}\left(\begin{pmatrix} g_n \\ g_g \end{pmatrix} + a \mid Z\right) \\ =& (1-w)\Sigma_g + wA \\ =& \begin{pmatrix} A_{\text{gn}}A_{\text{gg}}^{-1}\tilde{G}A_{\text{gg}}^{-1}A_{\text{ng}} + A_{\text{nn}} - A_{\text{gn}}A_{\text{gg}}^{-1}A_{\text{ng}} & A_{\text{gn}}A_{\text{gg}}^{-1}\tilde{G} \\ \tilde{G}A_{\text{gg}}^{-1}A_{\text{ng}} & \tilde{G} \end{pmatrix} \end{aligned} \tag{2.13}$$

for $\sigma_h^2 = \sigma_g^2 + \sigma_a^2$, $w = \sigma_a^2/(\sigma_a^2 + \sigma_g^2)$ and $\tilde{G} = (1-w)G + wA_{\text{gg}}$, which is non-singular. The factor w is interpreted as the proportion of the genomic variance that is explained by residual polygenic effects (see, e.g., Legarra et al. (2022); Vandenplas et al. (2018)). The inverse H^{-1} is computed using Schur's complement, deducing the neat representation:

$$H^{-1} = \begin{pmatrix} \tilde{G}^{-1} - A_{\text{gg}}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + A^{-1}.$$

Denoting the full genomic value by

$$\tilde{g} = \begin{pmatrix} g_n \\ g_g \end{pmatrix} + a,$$

the gBLUP model from Equation 2.10 now reads as

$$y = X\beta + W\tilde{g} + \varepsilon, \tag{2.14}$$

while the MME are constructed through the expression

$$\begin{pmatrix} X^T\Sigma_\varepsilon^{-1}X & X^T\Sigma_\varepsilon^{-1}W \\ W^T\Sigma_\varepsilon^{-1}X & W^T\Sigma_\varepsilon^{-1}W + \frac{\sigma_\varepsilon^2}{\sigma_h^2}H^{-1} \end{pmatrix} \begin{pmatrix} \hat{\beta} \\ \hat{g} \end{pmatrix} = \begin{pmatrix} X^T\Sigma_\varepsilon^{-1}y \\ W^T\Sigma_\varepsilon^{-1}y \end{pmatrix}. \tag{2.15}$$

Though technically only a sufficient and necessary condition for the BLUE $\hat{\beta}$ and BLUP $\hat{g}$, the equation system 2.15 is referred to as single-step gBLUP model (ssGBLUP). Similarly, equation systems that define a BLUP for the SNP effect vector u are called single-step SNP BLUP (ssSNPBLUP) models.
The authors of Liu et al. (2014) propose the use of the ssSNPBLUP model

$$\begin{pmatrix} X^T\Sigma_\varepsilon^{-1}X & X_n^T\left(\Sigma_\varepsilon^{(n)}\right)^{-1}W_n & X_g^T\left(\Sigma_\varepsilon^{(g)}\right)^{-1}W_g & 0 \\ W_n^T\left(\Sigma_\varepsilon^{(n)}\right)^{-1}X_n & W_n^T\left(\Sigma_\varepsilon^{(n)}\right)^{-1}W_n+\mathbf{\Sigma}^{11} & \mathbf{\Sigma}^{12} & \mathbf{\Sigma}^{13} \\ W_g^T\left(\Sigma_\varepsilon^{(g)}\right)^{-1}X_g & \mathbf{\Sigma}^{21} & W_g^T\left(\Sigma_\varepsilon^{(g)}\right)^{-1}W_g+\mathbf{\Sigma}^{22} & \mathbf{\Sigma}^{23} \\ 0 & \mathbf{\Sigma}^{31} & \mathbf{\Sigma}^{32} & \mathbf{\Sigma}^{33} \end{pmatrix} \begin{pmatrix} \hat{\beta} \\ \hat{g}_n \\ \hat{g}_g \\ \hat{u} \end{pmatrix}$$
$$= \begin{pmatrix} X^T\Sigma_\varepsilon^{-1}y \\ W_n^T\left(\Sigma_\varepsilon^{(n)}\right)^{-1}y_n \\ W_g^T\left(\Sigma_\varepsilon^{(g)}\right)^{-1}y_g \\ 0 \end{pmatrix},$$

where the matrix

$$\Sigma_\varepsilon^{-1} = \begin{pmatrix} \left(\Sigma_\varepsilon^{(n)}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \left(\Sigma_\varepsilon^{(g)}\right)^{-1} \end{pmatrix}$$

is the inverse of the residual covariance matrix and

$$\mathbf{\Sigma}^{-1} = \begin{pmatrix} A^{nn} & A^{ng} & 0 \\ A^{gn} & A^{gg}+\left(\frac{1}{w}-1\right)A_{gg}^{-1} & -\frac{1}{w}A_{gg}^{-1}Z \\ 0 & -\frac{1}{w}Z^TA_{gg}^{-1} & \frac{1}{w}Z^TA_{gg}^{-1}Z+\frac{2p^T(1-p)}{1-w}\mathbb{1}_{k\times k} \end{pmatrix}\sigma_u^{-2}. \tag{2.16}$$

A computational application of this model will be discussed in Chapter 4. However, this equation system is only one of many possible formulations. For instance, Mäntysaari and Strandén (2016) estimate the parameter vector $(\hat{\beta}^T, \hat{g}_g^T, \hat{a}_g^T, \hat{u}^T)^T$ instead.
Similarly, variations of the ssGBLUP model have been studied. For instance, Mäntysaari et al. (2020) propose the use of the decomposition

$$TT^T = \frac{1}{w}A_{gg}^{-1}Z\left(\frac{1}{w}Z^TA_{gg}^{-1}Z+\frac{2p^T(1-p)}{1-w}\mathbb{1}_{k\times k}\right)^{-1}Z^T\frac{1}{w}A_{gg}w^{-1},$$

so that the Woodbury matrix identity yields

$$\tilde{G}^{-1} = \frac{1}{w}A_{gg}^{-1} - TT^T.$$

Due to the matrix T, this model has been coined ssGTBLUP and holds the advantage that T only needs to be computed once. Furthermore, the right order of calculations reduces the computational burden if $k \ll n$. A slight variation of this model, the ssGTABLUP model, will be evaluated numerically in Chapter 4.
Practical applications often consider multiple correlated traits. In this case, the above equations are extended to consider y as a random matrix instead of a random vector. Furthermore, additional variance components are introduced which model the cross-correlations between traits which in turn are used to deduce the ratio of cross-correlation variance components in random effect vectors (Liu et al., 2014; ten Napel et al., 2021).

2.3 Computational considerations

The question of how to estimate breeding values empirically has always been part of the motivation in animal breeding research. Many of the most influential articles in the field dedicate a section to the computational implementation of methods (Meuwissen et al., 2001; VanRaden, 2008; Christensen and Lund, 2010). A general problem when solving MME of the form 2.3 or 2.5 lies in the presence of variance components in the coefficient matrix which are a priori unknown. Cunningham and Henderson (1968) suggested a procedure resembling the expectation-maximization algorithm which iteratively computes a candidate vector $\hat{\beta}$ based on estimates and then derives new estimates for the variance components from this candidate vector. Today, however, variance components are commonly estimated in a separate step. A popular method is the use of restricted maximum likelihood (REML) (Patterson and Thompson, 1971) which involves maximizing the likelihood of a linearly transformed Mixed Effects Model

$$K^T y = K^T Z u + K^T \varepsilon,$$

where $K \in \mathbb{R}^{m \times n}$ is chosen such that $K^T X = 0$. Broadly used software packages implementing this method include `rrBLUP` (Endelman, 2011), `sommer` (Giovanny, 2016) and `ASReml` (Gilmour et al., 2002).
Alternatively, estimates for variance components can be obtained from known or estimated heritability values, as the formula for heritability is structurally similar to the ratio of variance components in the coefficient matrix. For instance, genetic heritability in humans is often inferred from twin studies.
Computing the BLUE $\hat{\beta}$ and the BLUP $\hat{g}$ or $\hat{u}$ can be achieved in two ways: The simple method uses the explicit formulae 2.6, 2.7, 2.8 and/or 2.9. Special care is needed in the treatment of the involved matrix inverses. On current computers, only dense equation systems in the low hundreds of thousands can be solved through direct inversion, e.g., a Cholesky decomposition. Further issues are based on operations on the matrix X as the number of fixed effects can be in the millions. A classical technique for this purpose is called iteration-on-data which essentially refers to the use of out-of-core computations (Strandén and Lidauer, 1999; Schaeffer and Kennedy, 1986).

The alternative method employs iterative-solver algorithms, such as Gauß-Seidel or Preconditioned Conjugate Gradient (PCG), to solve the MME. With this approach breeding values for tens of millions of animals can be estimated at the moment. Software solutions based on iterative-solver techniques include `MiXBLUP` (ten Napel et al., 2021), `Mix99` (Vuori et al., 2006), `BLUPF90` (Misztal et al., 2014b) and `BOLT` (Garrick et al., 2018), most of which are proprietary and closed-source. While iterative solvers are employed in a variety of fields, their use in breeding value estimation has encountered a number of computational issues. First, if the number of individuals is large, then the GRM G is singular and, therefore, $\tilde{G}$ is also close to singularity for small values of w. In turn, this also leads to numerical problems with H^{-1}, though several approaches have been suggested to avoid its explicit construction (Mäntysaari et al., 2017, 2020). Second, variations of the ssSNPBLUP model infer not only one BLUP but a combination of $\hat{a}$, $\hat{g}_g$, $\hat{g}_n$ and $\hat{u}$. This increases the number of equations and thereby often leads to a slower convergence. Furthermore, the PCG often employs second-order derivatives of the minimization problem to be solved. However, in single-step models, this information is difficult to obtain due to the large dimensions of the coefficient matrix involved. Methods for choosing good preconditioners and improving convergence speed have been introduced by Taskinen et al. (2017) and Vandenplas et al. (2018, 2019, 2020, 2023).

Chapter 3

Techniques for an efficient SNP matrix multiplication

In the previous chapter, we discussed why genomic datasets used in empirical research are steadily growing in size. This growth has introduced challenges in the calculation of population statistics that are based on large parts of the genome. In other fields of research, similar computational challenges have been tackled with the help of GPUs. Within our work for the manuscript Freudenberg et al. (2023a), we have developed a range of algorithms for the calculation of essential SNP matrix operations widely used in empirical studies, which take advantage of modern NVIDIA® GPUs. We provide an implementation in the C library `miraculix`, together with exemplary interfaces in Julia and Fortran. To ease adaptation we also supply functions to calculate a number of derivatives, such as the genomic relationship matrix (GRM), linkage disequilibrium (LD) statistics, the genomic BLUP, and principal components analysis. Source code for `miraculix` is released under the Apache 2.0 licence and is freely available at https://github.com/alexfreudenberg/miraculix. The library is developed in C, C++ and CUDA and has supplementary bindings in Julia. An archived version of the repository is available at https://doi.org/10.6084/m9.figshare.23725977.v1.
This chapter is based on joint work with Martin Schlather, Guido Moerkotte and Torsten Pook.

3.1 Introduction

Through the emergence of high-throughput sequencing technology, the recent decades have seen the collection of massive genomic datasets, furthering the research in various fields in genetics such as human medicine or animal breeding and plant breeding. The consideration of large amounts of data helps to increase the accuracy of predictive models (Canela-Xandri et al., 2016; Zhao et al., 2021; Singh and Prasad, 2021) and some authors claim that big data can contribute towards the closing of the missing heritability gap (Kim et al., 2017; Pallares, 2019). For breeding purposes, the use of genomic in-

formation leads to more accurate breeding values at earlier life stages, thus allowing for earlier selection to both reduce housing cost and increase genetic gain (Schaeffer, 2006). However, the computational analysis of these datasets places a significant burden on researchers and practitioners. In this article, we present `miraculix`, a lightweight C library which offloads the computation of important genomic quantities to NVIDIA® GPUs.

A GRM describes the proportion of the genome that is shared between individuals in a population (Mrode, 2014) and is used in various selection methods such as genomic BLUP (VanRaden, 2008), single-step genomic BLUP (Misztal et al., 2009), extended genomic BLUP for modeling epistatic effects (Jiang and Reif, 2015) or (selective) epistatic random regression BLUP (Vojgani et al., 2021). Similarly, LD measures the statistical similarity of pairs of SNPs in a population. For instance, LD quantities are used in human genetic studies to infer information on disease causes or population history (Pritchard and Przeworski, 2001; Gazal et al., 2017).

Due to the large dimensions of modern genomic data sets, a naive calculation of the GRM, LD and their derivatives would inflict extraordinarily high computational demands, both in terms of memory requirements and calculation times. Since the SNP genotype of an individual is usually coded as 0 for one homozygous genotype, 1 for the heterozygous genotype, or 2 for the alternate homozygous genotype, each SNP value can be stored in 2 bits of memory. An example of this compressed storage format is the PLINK binary format. Usually, the reference allele is coded as 0. While many statistical objects in genomics can be calculated through the use of highly-optimized BLAS libraries, similar utilities are not available for these compressed storage formats.

There exist two main approaches to mitigate this problem. The first one decompresses SNP genotype data before further processing. For instance, the R packages `AGHmatrix` (Rampazo Amadeu et al., 2016), `qgg` (Rohde et al., 2019), `rrBLUP` (Endelman, 2011) and `snpReady` (Granato et al., 2018) use custom floating-point matrix operations for the calculation of the GRM or rely on BLAS libraries. The R package `SNPRelate` (Zheng et al., 2012) benefits from explicit SIMD instructions in the calculation of LD and the GRM. Standalone solutions for the calculation of LD include `HaploView` and `LDkit` (Barrett et al., 2004; Yao, 2020).

The calculation of the GRM and LD statistics is also implemented in the software packages `PLINK` and `GCTA` (Yang et al., 2011; Chang et al., 2015) which have popularized the second approach for processing compressed genotype data. They both utilize bit-compressed algorithms for an efficient calculation of the dot product of SNP vectors. Motivated by the remarkable speed improvements of these implementations, a number of tailored algorithms for the dot product have been developed for different instruction set architectures which are up to 48 times faster than a naive BLAS-based implementation (Schlather, 2020).

Additionally, some software solutions have studied the benefit of offloading genotype matrix operations to the GPU. `PLINK 2.0` provides a BLAS-based calculation of the GRM on GPUs. However, according to the documentation, this functionality is just provided as a proof-of-concept. The Julia package `SnpArrays.jl` (Zhou et al., 2020)

uses BLAS libraries on the GPU to accelerate the multiplication of SNP matrices by a floating-point vector.
Over the past few years, there has been a rising interest in low-precision arithmetics in the field of deep learning (Hubara et al., 2017), which has led to hardware improvements. For example, recent NVIDIA® architectures have introduced a number of new assembler instructions for this purpose. In deep learning, the method of quantization reduces the cardinality of possible values of a parameter by using low-precision integers and has been used in neural networks to increase the number of parameters (Gholami et al., 2021; Dettmers et al., 2022; Kim et al., 2022). This progress has opened new paths to explore for acceleration in genomic calculations.
We present the library `miraculix` which implements functions for the GPU-based multiplication of compressed SNP matrices by itself or floating-point matrices, which helps to accelerate the calculation of the GRM, LD and other essential quantities in genomics. In contrast to aforementioned software packages such as `PLINK`, the package `miraculix` offers only a narrow, highly fine-tuned functionality and is designed to allow a neat integration into genomic analysis pipelines. Furthermore, it differentiates itself from other GPU software solutions by using low-precision instructions available on NVIDIA® GPUs to operate on compressed SNP data. This technique reduces device memory requirements and is substantially faster than solutions in floating-point format. We provide interfaces which can be used by existing libraries for genomic analysis or in higher-level programming languages such as Julia (Bezanson et al., 2017).

3.2 Methods

For a diploid species, the SNP matrix M describes the genomic information of a set of genetic markers in the population. That is, $M \in \{0,1,2\}^{n\times k}$, where n is the number of individuals in the population and k is the number of SNPs. Due to the dramatic decrease in sequencing costs over the last decades, it is now possible to genotype millions of SNPs in vast populations or, alternatively, impute incompletely genotyped individuals. Therefore, researchers regularly deal with extraordinarily large data sets. For instance, the UK Biobank comprises broad genetic data of hundreds of thousands of human individuals (Bycroft et al., 2018).
The SNP matrix is used for a wide range of genomic analyses. In population analysis, for instance, the SNP matrix is used in the classical genomic relationship matrix (GRM) G, which is defined by

$$G = \frac{ZZ^T}{2p^T \cdot (1_k - p)}$$

with $1_k = (1, ..., 1)^T$ denoting a vector of length k consisting only of 1s, and p denoting the vector of allele frequencies. If p is assumed to be equal to the empirically observed frequencies, then the centered SNP matrix can be computed as

$$Z = PM, \quad \text{for} \quad P = \mathbb{1}_{n\times n} - 1/n \cdot 1_n 1_n^T.$$

Here, the matrix P scales M to have zero-centered alleles counts (Method 1 in VanRaden (2008)). In genome-wide analysis studies (GWAS), the SNP matrix Z is used to calculate correlations between traits and one or multiple SNPs (Jiang et al., 2019). In the analysis of linkage disequilibrium, the SNP matrix is used to approximate the correlation statistic r^2 through the computation of the correlation between allele counts in M. Due to the intrinsic properties of a SNP matrix, the efficient computation of MM^T or M^TM is in fact the problem of a $\{0, 1, 2\}$-matrix multiplication (Chang et al., 2015; Schlather, 2020).

Memory-efficient storage formats for M, such as the PLINK binary format, only use 2 bits per entry and the conceptual arrangement of these bits yields different multiplication approaches. A number of highly efficient SIMD-based algorithms for CPUs have been suggested (Schlather, 2020; Chang et al., 2015). Here, we rely on an allele-count encoding for our GPU implementation `MMAGPU`, which stores counts in unsigned 2-bit integers. This allows us to target the 4-bit matrix multiplication assembler instructions on modern NVIDIA® GPUs of compute capability 7.5 and higher. Through bit-masking and shift operations, we obtain a straightforward matrix multiplication microkernel. For fast data movement from global memory to shared memory to the cores and back, our library extends the CUTLASS library (NVIDIA, 2023) with 2-bit specializations, utilizing the available fast tile iterators. Since the resulting multiplication function is mainly bound by data transfers between the GPU and main memory, we divide the multiplication into blocks of rows and parallelize the multiplication of these rows into different threads and streams respectively.

Deviating from the above concepts, an efficient multiplication of the SNP matrix by a floating-point matrix is required for other essential operations in genomics, e.g., in GWAS. To our knowledge, `miraculix` is the first software library which offers a GPU-based implementation of optimized matrix multiplications on compressed genotype data. The R packages MoBPS (Pook et al., 2020) and EpiGP (Vojgani et al., 2023), as well as the proprietary software MiXBLUP (ten Napel et al., 2021), have integrated `miraculix`.

3.3 Results

Since multiplications of the SNP matrix are an essential operation in a number of computational tasks in genomics, `miraculix` can be used as the backend for various calculations. In this section, we describe four possible applications of our high-performance implementation and demonstrate how it enables the processing and analysis of datasets in previously unattainable computing times.

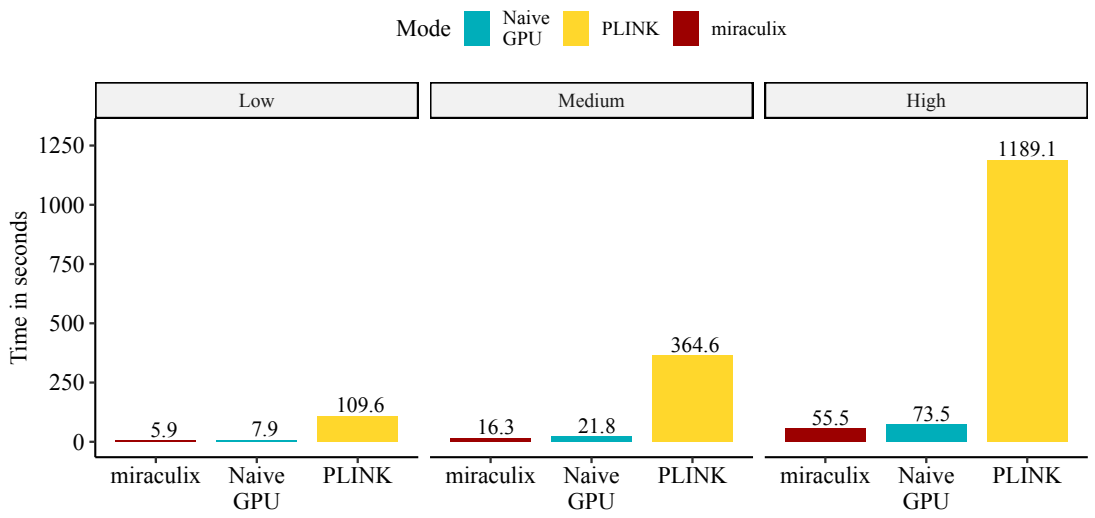


Figure 3.1: Wall clock times for the calculation of the genomic relationship matrix on three simulated sets of allele markers for 22,000 individuals. The genotype datasets comprise 50,241 ("low"), 250,000 ("medium") and 1,000,000 ("high") SNPs. Evaluations were performed on 64 cores of a dual-socket AMD® EPYC 7513 (2.6 GHz) and an NVIDIA® A100-80GB. Displayed is the median of 5 evaluations when using `PLINK`, a `cuBLAS`-based solution ("Naive GPU") and `miraculix`.

Table 3.1: Wall clock times for calculating the GRM for different SNP densities. Included operations: 1) Reading of SNP data in PLINK binary format, 2) Calculation of allele frequencies, 3) Conversion of PLINK format to miraculix-internal storage format, 4) Reformatting of the dataset from SNP-major to individual-major, 5) SNP-matrix multiplication, 6) Rank-one updates of the covariance matrix, 7) Scaling of covariance matrix by sum of allele variances, and 8) Writing the result to the disk. Evaluations were performed on a dual-socket Intel® Xeon Gold 6230 with 40 dedicated cores.

Wall clock time (s)	**Low**	**Medium**	**High**
Reading	0.20	0.61	1.79
Allele frequencies	0.05	0.27	1.07
Format conversion	0.01	0.05	0.24
Transpose formatting	4.59	19.69	91.97
Matrix multiplication	0.68	1.87	6.66
Rank-one updates	0.74	0.77	0.73
Scaling	0.0	0.0	0.0
Writing	2.27	2.61	2.28

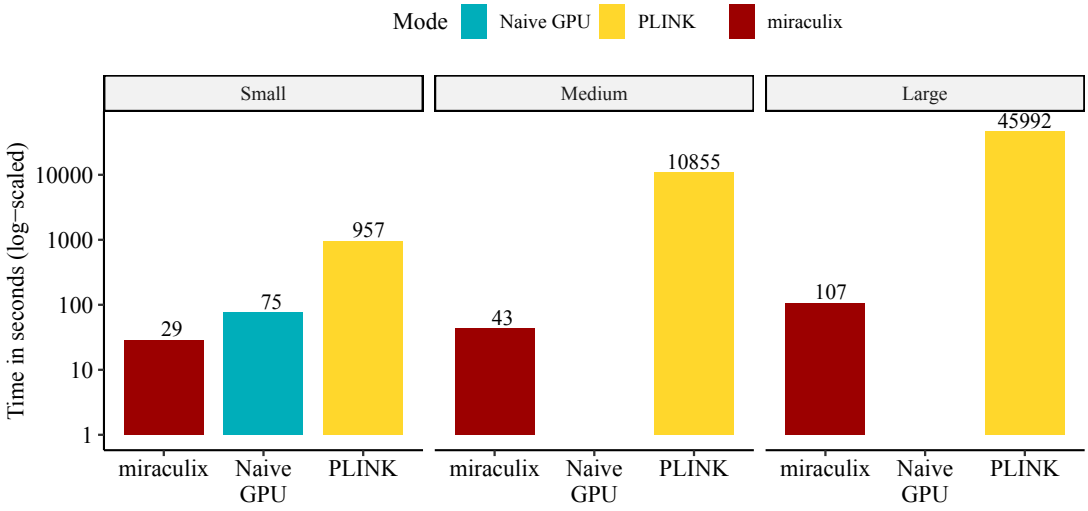


Figure 3.2: Wall clock times for the calculation of LD matrix R^2 for three populations of 50,241 SNPs each. Compared are the times required by `PLINK`, a `cuBLAS`-based solution ("Naive GPU") and `miraculix`. The "small", "medium" and "large" populations have 102,000, 751,000 and 3,101,000 individuals respectively. Evaluations were performed on 64 cores of a dual-socket AMD® EPYC 7513 (2.6 GHz) and an NVIDIA® A100-80GB. Displayed is the median of 5 evaluations, except for PLINK for which results are based on a single run.

Table 3.2: Wall clock times for calculating the LD matrix R^2 for different population sizes. Included operations: 1) Reading of SNP data in PLINK binary format, 2) Calculation of allele frequencies, 3) Conversion of PLINK format to miraculix-internal storage format, 4) Checking for missing values in SNP data, 5) SNP-matrix multiplication, 6) Rank-one updates of the covariance matrix, 7) Scaling of covariance matrix by reciprocal of diagonal elements of the covariance matrix, and 8) Writing the result to the disk. Evaluations were performed on a dual-socket Intel® Xeon Gold 6230 with 40 dedicated cores.

Wall clock time (s)	**Small**	**Medium**	**Large**
Reading	0.43	2.57	10.43
Allele frequencies	0.25	1.72	6.75
Format conversion	0.05	0.37	1.22
Validation of data	0.76	5.60	23.22
Matrix multiplication	4.23	21.20	63.47
Rank-one updates	10.97	10.48	10.19
Scaling	6.63	6.63	6.63
Writing	12.47	13.00	12.41

3.3.1 Genomic Relationship Matrix

For large dimensions of Z, a straightforward calculation of G becomes computationally prohibitive and a careful treatment of the involved operations is required. The decomposition

$$n^2 cG = n^2 MM^T - n1_n 1_n^T MM^T - nMM^T 1_n 1_n^T + 1_n 1_n^T MM^T 1_n 1_n^T,$$

with $c = 2p^T(1_k - p)$, reveals that the matrix G can be obtained from MM^T at relatively low computational costs of order $n^2 + nk$, whereas MM^T requires $\mathcal{O}(kn^2)$ calculations (Schlather, 2020). Moreover, $n^2 cG$ is integer-valued and can be computed without any numerical error.

In Figure 3.1, we compare the computation time of our GPU implementation with the CPU-targeted solution in PLINK. As these evaluations are performed on different hardware, we also benchmark a naive GPU implementation, which involves unpacking compressed genotype data into unsigned integers and uses the NVIDIA® `cuBLAS` library for multiplication. This approach resembles the implementation in `PLINK`, though we opted to store the input data in integers of 8 bits to save memory, while the proof-of-concept in `PLINK` uses single-precision floating-point values. We simulated three different sets of genotype markers for a population of 22,000 individuals with the simulation utility in PLINK: A low-density array with 50,241 markers ("Low"), a medium-density array with 250,000 markers ("Medium") and a high-density one with 1,000,000 markers ("High"). For reference, the UK Biobank currently comprises about 850,000 directly measured variants. We tested the GPU functions on an NVIDIA A100 with 80GB of device memory, while running PLINK on a dual-socket AMD® EPYC 7513 (2.6 GHz) with 32 dedicated cores each using the `PLINK` options `-make-rel square cov`. The results displayed are the median of 5 evaluations. Though direct conclusions on the efficiency of each solution are hard to draw due to the different hardware involved in the benchmarks, it can be observed that wall clock times in `miraculix` are smaller by a factor of at least 18 across the three test sets compared to PLINK. On the large dataset, the computation time was reduced from approximately 20 minutes to 56 seconds. Juxtaposing our solution to the simple `cuBLAS`-based solution, we see that significant speed gains can still be achieved by using our stack of microkernels for sub-byte integers and efficient memory management.

Yet, the significantly higher price tag of the A100 GPU has to be considered when evaluating these results: It is available at about 15,000 USD with a thermal power design (TDP) of 300W, while each of the two AMD® EPYC 7513 (2.6 GHz) CPUs has a recommended price of 2,840 USD with a TDP of 200W. Considering the power consumption of the evaluated methods, it is reasonable to assume that the GPU approaches are significantly more efficient than `PLINK`. Making a rough estimate based on the involved TDPs and computing times, a reduction in the magnitude of 20 can be presumed.

3.3.2 The gBLUP model

The genomic BLUP (gBLUP) model is widely used in population analysis to capture additive genetic effects (Misztal and Legarra, 2017) and is the basis for various extensions such as the extended gBLUP model, the single-step gBLUP model or the epistatic random regression BLUP model (Misztal et al., 2009; Jiang and Reif, 2015; Vojgani et al., 2021). In the gBLUP model, a quantitative trait y is assumed to be in a linear relationship with the genetic markers and environmental influences, captured in a matrix $X \in \mathbb{R}^{n \times l}$. The effects of SNPs are traditionally assumed to be random, resulting in the model

$$y = X\beta + Zu + \varepsilon,$$

where $\beta \in \mathbb{R}^l$ is a fixed effect, $u \sim \mathcal{N}(0, \sigma_u^2 \mathbb{1}_{k\times k})$ is a random effect and $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 \mathbb{1}_{n\times n})$, independent of u, is an error term. Often, the notation $g = Zu$ is used for the so-called breeding values. Then, the vector g is normally distributed with mean 0 and covariance matrix $\sigma_g^2 G$ for a constant $\sigma_g^2 > 0$. Furthermore, the best linear unbiased estimator for β is given by

$$\hat{\beta} = \left(X^T \left(\frac{\sigma_\varepsilon^2}{\sigma_g^2} \mathbb{1}_{n\times n} + G \right)^{-1} X \right)^{-1} X^T \left(\frac{\sigma_\varepsilon^2}{\sigma_g^2} \mathbb{1}_{n\times n} + G \right)^{-1} y$$

and the best linear unbiased predictor for g is given by

$$\hat{g} = G \left(G + \frac{\sigma_\varepsilon^2}{\sigma_g^2} \mathbb{1}_{n\times n} \right)^{-1} (y - X\hat{\beta}).$$

In practice, the variance components σ_u^2, σ_g^2 are either derived from previous estimates on the heritability of the trait y (e.g., by comparing offspring phenotypes with parental phenotypes) or estimated through Restricted Maximum Likelihood (REML), for instance, using the software package ASReml (Butler et al., 2017). Considering the above identities, the quantities $\hat{g}$ and $\hat{\beta}$ can be derived from the GRM G and estimates $\hat{\sigma}_u^2, \hat{\sigma}_g^2$ through a Cholesky decomposition. To this end, we utilize the `cuSOLVER` library to offload this computation to the GPU. In a recent empirical study, the full gBLUP calculation with `miraculix` showed an acceleration of up to 100 times compared to traditional software in the case where the heritability is known (Pook et al., 2021). Additionally, two recent studies investigating the effects of epistasis utilized the efficiency of optimized CPU functions in `miraculix` (Vojgani et al., 2021, 2023). However, it should be noted that the memory requirements for setting up the GRM increase quadratically with the number of individuals which puts a limit to potential problem sizes.

3.3.3 Linkage Disequilibrium

LD is a way of describing the dependence structure between pairs of alleles in a set of markers and there exist different statistics to capture this information in a population (Pritchard and Przeworski, 2001). The software `PLINK` implements the LD statistics r^2,

Table 3.3: Computing times for various steps in a gBLUP calculation. Computations were performed on a single NVIDIA® A100-40GB using Julia interface of miraculix. Total times include additional system start-up time.

Calculation	Wall clock time (s)	Main memory usage (GB)	Device memory usage (GB)
Data set-up	20.20	26.24	-
GRM	30.48	18.67	5.57
PCA	17.63	5.01	17.23
gBLUP	12.73	0.13	18.67
Total	95	36.08	18.67

D and D', which can be thought of as correlation measures between alleles. Though the true linkage value is based on haplotypes, it is sometimes approximated by the allele count correlations (e.g., in PLINK). That is, the squared correlation between the columns i and j of M is used as the value for r^2. Since the correlation matrix R can be written as

$$R = D^{-1/2}\Sigma_M D^{-1/2} \quad \text{for} \quad \Sigma_M = Z^T Z - 4npp^T, D = \text{diag}(\Sigma_M).$$

the matrix R^2, consisting of pairwise r^2 values, can be computed from Σ_M at low cost. While the computation of R^2 for a small block of SNPs with a limited number of individuals is straight-forward, a simple algorithm for the detection of LD between distant SNPs (so-called long-range LD) or the calculation of the average LD decay in a large part of the chromosome becomes cumbersome.

In our experiments, we calculated the matrix R^2 of 50,241 markers across three simulated populations: A small population of 102,000 individuals, a medium-sized one comprising 751,000 individuals and a large population of 3,101,000 individuals. As inflating the large population to single-precision floating-point values would require approximately 580GB of memory, this approach is impractical for LD calculation. As `miraculix` processes SNP data in compressed format and subdivides the computation of the SNP matrix multiplication into blocks, only about 6 GB of device memory was required.

Using the same hardware set-up as above, we compare our solution with the implementation in PLINK on 64 cores and a simple GPU solution in `cuBLAS`. However, due to its higher device memory requirements, the latter could only be evaluated on the small dataset. Results are displayed in Figure 3.2 and are the median of 5 evaluations for the GPU functions. PLINK calculations were only performed once as wall clock times on these test sets made further evaluations unreasonable. For LD calculation, the `PLINK` options `-r square` were used. We observe that compute times in PLINK were more than 400 times higher on the large dataset

3.3.4 PCA

For a column-wise standardized matrix $U \in \mathbb{R}^{n \times l}$, the first m principal components are defined by $Uv_1, ..., Uv_m$, where $v_1, ..., v_m$ solve the maximization problems

$$\max_{v_1 \in \mathbb{R}^l, \|v_1\|=1} \|Uv_1\|,$$

and

$$\max_{v_k \in \mathbb{R}^l, \|v_k\|=1} \|Uv_k\| \quad \text{such that} \quad v_k^T v_j = 0,\ j = 1, ..., q-1,$$

for $q = 2, ..., m$. Even though the transfer of the PCA to non-continuous data is not straightforward and a topic of ongoing research (see, e.g., Schlather and Reinbott (2021) for an approach to non-Euclidean data), PCA is still regularly used as an auxiliary tool in statistical genomics. Since principal component analysis (PCA) is a dimension-reducing method aimed at capturing large parts of variation in the dataset, PCs of the GRM are used in empirical studies in genetics to investigate population structure (e.g., by Steyn et al. (2022)) or as an auxiliary tool in the REML-based estimation of variance components (Thompson and Shaw, 1990; Lee and van der Werf, 2016). Principal components in the SNP direction are regularly used to control for population stratification in GWAS studies or in breeding value estimation to increase the accuracy of predictive models (Price et al., 2006). Popular software solutions include `Eigensoft` and `PLINK` (Price et al., 2006; Chang et al., 2015). Since PCA requires the multiplication of an orthonormal matrix of eigenvectors of $U^T U$ by U, `miraculix` can help to accelerate the PCs of a population by a fast computation of the GRM. Similarly, the PCs of SNPs can be derived from the $M^T M$ matrix. However, if the dataset contains a lot of markers, constructing this matrix is challenging. The functionality of `miraculix` to multiply a SNP matrix by a floating-point matrix helps to alleviate this burden since their exists randomized algorithms for the singular value decomposition that do not require an explicit construction and calculate the first m eigenvalues and their corresponding eigenvectors with high accuracy (Halko et al., 2011). We provide an exemplary implementation in Julia.

3.3.5 Computing times

To evaluate the performance of `miraculix` in a practical setting, we simulate a population of 50.000 animals with 727,605 variants based on the Illumina BovineHD BeadChip (Cunningham et al., 2021). Our supplementary Julia functions are linked to the interface of the library and perform low-cost post-processing operations on its return values. Emulating typical computational tasks in practice, we first load and process our data, which is stored in PLINK binary format on the disk, then calculate the GRM of the population and the SNP-wide PCs for later usage in inferring the parameters of the gBLUP model. PCs were modeled as fixed effects.

Data processing operations were performed on an AMD® EPYC 7513 (2.6 GHz), while SNP matrix operations were offloaded to an NVIDIA® A100. Since the CPU was mainly used for data preprocessing, we only used 8 dedicated cores. Due to the memory efficiency of our implementation, we were able to use the version of the A100 GPU with only 40GB of device memory. Computation of the GRM involved calculating the SNP matrix cross-product of dimensions 50,000 times 50,000 while retrieving the first 10 principal components required the multiplication of the SNP matrix by a floating-point matrix to obtain the approximate eigenvectors of SNP-wide covariance matrix. To estimate the vectors $\hat{\beta}$ and $\hat{g}$ of the gBLUP model, the Cholesky decomposition of the stretched GRM needed to be computed to solve the involved equation systems. Since heritability was assumed to be known, the ratio of variance components did not need to be estimated on the data.
Results are displayed in Table 3.3. We note that data processing now constitutes a significant portion of the total compute resource requirements both in terms of memory and computing times, as it requires a 2-bit format conversion and reordering of the bit-level values. The construction of the GRM was performed in just approx. 30 seconds, whereas the PCA and gBLUP calculations needed 18 seconds and 13 seconds each. In total, approx. 36 gigabytes of main memory and 19 gigabytes of device memory were used.

3.4 Discussion

We have presented the capability of `miraculix` to offload essential operations on genomic data to the GPU. We illustrated its benefits in four applications. The package outperforms existing CPU-based software solutions significantly and thereby enables much faster processing of genomic datasets of substantial size. Furthermore, it works on compressed data and therefore allows the processing of huge datasets.
Overall, our experiments showed that a full gBLUP on a population of 50,000 individuals could be performed in little more than 1.5 minutes. Considering that a similar task was assumed to be computationally infeasible by VanRaden (2008) at the time, we find the performance improvements to be promising and encourage the use of GPUs to accelerate the processing of large datasets in genomics. While there is a suite of established methods to deal with extraordinary dimensions, e.g., Algorithm for Proven and Young (APY) (Misztal et al., 2014a) or the use of iterative solvers (Strandén and Lidauer, 1999), these approaches can similarly benefit from the techniques introduced in this article.
To allow `miraculix` to handle further increases in dataset sizes, future software versions might include distributed calculations for high-performance clusters via a Message Passing Interface (MPI), which would extend its applicability to datasets that still cannot be fully stored in device memory. Furthermore, since the variance component estimation through REML is another computational bottleneck in genomic analyses, it would be interesting to offload this procedure to the GPU as well. Extensions of `miraculix` to

AMD® or Intel® GPUs would be useful to allow researchers to take full advantage of existing compute hardware.

Chapter 4

Acceleration of SNP matrix-vector multiplications for single-step models

In the last decade, a number of methods have been suggested to deal with large amounts of genetic data in genomic predictions. Yet, steadily growing population sizes and the suboptimal use of computational resources are pushing the practical application of these approaches to their limits. As an extension to the C/CUDA library `miraculix`, we have developed tailored solutions for the computation of genotype matrix multiplications which is a critical bottleneck in the empirical evaluation of many statistical models. We demonstrate the benefits of our solutions at the example of single-step models which make heavy use of this kind of multiplications. Targeting modern NVIDIA® GPUs as well as a broad range of CPU architectures, our implementation significantly reduces the time required for the estimation of breeding values in large population sizes. `miraculix` is released under the Apache 2.0 license and is freely available at https://github.com/alexfreudenberg/miraculix.
This chapter is based on a joint manuscript with Jeremie Vandenplas, Martin Schlather, Torsten Pook, Ross Evans and Jen ten Napel (Freudenberg et al., 2023c).

4.1 Introduction

Over the past 15 years, 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 passes on to its offspring. Though breeding values are estimated most accurately through the use 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 combine pedigree information

of ungenotyped animals with Single Nucleotide Polymorphism (SNP) data of genotyped animals to analyze phenotypic records.
Due to its desirable statistical properties, the single-step genomic BLUP (ssGBLUP) model (Legarra et al., 2009; Christensen and Lund, 2010) has gained popularity in animal breeding, joining both the genomic relationship matrix (GRM) G and the pedigree-based relationship matrix A into a generalized relationship matrix H (see Equation (2.13)). 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, G^{-1}, through genomic recursion on a subset of core animals (Misztal et al., 2014a). 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 G using the Woodbury decomposition. The resulting models were coined single-step GT(A)BLUP models (Mäntysaari et al., 2017, 2020). Additionally, single-step SNP BLUP (ssSNPBLUP) models were proposed to estimate SNP effects directly, similarly avoiding G and its inverse (Liu et al., 2014; Fernando et al., 2014; Taskinen et al., 2017).
Considering the computational aspects of these approaches, the use of highly-optimized sparse matrix operations has been established, thanks to the role of pedigree-based relationship matrix. 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 explicit construction of the full coefficient matrix.
Nevertheless, the computational load of the remaining mathematical operations and 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 the Intel® Math Kernel Library (MKL) and PARDISO (Alappat et al., 2020), and unpacking the compressed SNP matrix M 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 use in single-step BLUP evaluations. The software PLINK also implements BLAS-based routines for the calculation of G on an 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 compressed SNP matrix by a matrix of small width stored in floating-point format for CPUs and NVIDIA® GPUs. Our CPU code is optimized for all major instruction set architectures. To take advantage of the instruction-level parallelism capabilities of modern CPUs, our implementation uses Single Instruction-stream Multiple Data-stream (SIMD) operations explicitly (Tanenbaum, 2013). Extending the NVIDIA® CUTLASS library (Thakkar et al., 2023), our GPU approach benefits from fast tile iterators in the data movement during the matrix-matrix multiplication.
We demonstrate how these advances can drastically reduce the computing times of genotype matrix multiplications on CPUs and GPUs compared to double-precision matrix multiplication routines provided by the Intel® MKL, while simultaneously reducing memory requirements. We provide scripts for reproducing our results in the GitHub repository. 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 approaches bring down total run times in solving single-step evaluations by up to 62%, thereby paving the way to include even larger population sizes in genomic evaluations.
We provide our implementation as part of the C/CUDA software library `miraculix` (https://github.com/alexfreudenberg/miraculix). Interfaces to call the library from higher-level languages such as Fortran, R and Julia are provided. Through the modular structure of the `miraculix` library, which also supplies functions for the calculation of G, the code should be easily modifiable by researchers and practitioners interested in accelerating computations in other BLUP models (Meuwissen et al., 2001; VanRaden, 2008) or other genomic analyses.

4.2 Methods

The efficient multiplication of the genotype matrix by a double-precision matrix plays an important role in many genomic analyses. In this section, we explain how this multiplication can be decomposed to reduce the computational costs involved. Then, we propose novel techniques for the multiplication of a compressed SNP matrix with a double-precision matrix on CPUs and GPUs. We illustrate the role of an efficient genotype matrix multiplication at the example of single-step models. Lastly, we describe the methodology which we used for evaluating our approaches.

4.2.1 Computational bottlenecks

We consider the commonly encountered operation of multiplying the centered SNP matrix Z, or its transpose Z^T, by a matrix of low width. Here, the matrix Z can be

computed as

$$Z = M - 2 \cdot 1_{n_g} p^T,$$

where $M \in \{0, 1, 2\}^{n_g \times k}$ denotes the uncentered genotype matrix containing the k 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. Furthermore, p denotes the vector of allele frequencies and the subtraction of $2 \cdot 1_{n_g} p^T$ centers the columns of M.
Considering an arbitrary matrix $\Lambda \in \mathbb{R}^{k \times l}$, we first note that the multiplication of Z with Λ can be reformulated into

$$Z\Lambda = M\Lambda - 2 \cdot 1_{n_g} p^T \Lambda.$$

Since subtracting the matrix $21_{n_g} p^T \Lambda$ consists of a vector-matrix multiplication and subsequent additions of matrices of rank one, it is computationally cheap and can be achieved with BLAS Level-1 operations. The multiplication of the transposed matrix $Z^T \tilde{\Lambda}$ with an arbitrary matrix $\tilde{\Lambda} \in \mathbb{R}^{n_g \times l}$ is decomposed similarly into

$$Z^T \tilde{\Lambda} = M^T \tilde{\Lambda} - 2 \cdot p 1_{n_g}^T \tilde{\Lambda}.$$

Here, we use the distinction between Λ and $\tilde{\Lambda}$ to emphasize that Z and Z^T cannot be multiplied with the same matrix because they have different dimensions. Due to the low cost of SNP genotyping, the matrix M can have extremely large dimensions, capturing the genomic information of millions of animals. Furthermore, whereas Λ is a regular matrix of double precision, the SNP matrix is usually stored in a compressed format to save memory. For instance, the PLINK 1 binary file format stores the genotypes of four individuals in eight bits (corresponding to one byte), utilizing that one entry of M only requires 2 bits of storage. This compressed data format prevents naive calls to BLAS routines, and decompressing it explicitly is inefficient and increases memory requirements. Only recently, the problem of matrix multiplication of mixed input data types has gained attention (Kim et al., 2022) and no off-the-shelf solution exists for compressed 2-bit integer data types.
In general, algorithms for genotype matrix multiplications that operate on compressed data can be expected to be more efficient due to the better utilization of memory movements. In cases where the multiplication $Z\Lambda$ needs to be evaluated repeatedly for varying Λ, an optional conversion of the uncentered SNP matrix M to a different storage format is comparatively cheap. Therefore, switching the storage format has the potential to yield efficiency gains.

4.2.2 Acceleration of the genotype matrix-matrix multiplication

Previous approach

Vandenplas et al. (2020) proposed a decompress-on-the-fly approach, consisting of unpacking tiles of submatrices of M small enough to store the result in the cache and

perform matrix multiplication on these tiles. Briefly, modern computer architecture implements different levels of cache memory (commonly, L1, L2 and L3) to reduce access times to repeatedly processed data. While infrequently used data can be stored in the random access memory (RAM) or even the disk, accessing it over the memory bus combined with the lower clock cycles of the RAM compared to the CPU dramatically slows down the execution of the program. While low levels of cache close to the core allow faster memory reads, they come with lower capacity (Tanenbaum, 2013). Hence, efficient code which processes large amounts of data strives to reduce data movement along the memory hierarchy and utilizes the fast access times of low-level caches. In the aforementioned local decompression approach by Vandenplas et al. (2020), small submatrices of SNP data in PLINK 1 binary format are converted to double-precision floating-point values at a time. Since these small submatrices are used repeatedly in a loop, they should not be evicted from the L1 cache. Therefore, the unpacked SNP data is readily at hand when new tiles of Λ are loaded and can be multiplied without additional conversion operations.

The *5codes* algorithm for CPUs

Building on the idea of keeping frequently used data close to the core, our novel approach for CPU computations aims to reduce data streams of Λ through the cache hierarchy by fully avoiding decompression. To explain this approach, we assume that there are no missing values at this point. Instead, they are coded as zero and their effect is taken into account when subtracting $2 \cdot 1_{n_g} p^T \Lambda$ or $2 \cdot {p^T}_{n_g} \tilde{\Lambda}$ later. Since the number of missing values is usually not substantial, this correction comes at a low cost.
We view the problem of storing compressed SNP data through the lens of combinatorics: Since there are only three possible states (0, 1 and 2) for a SNP, any vector $m \in \{0, 1, 2\}^5$ of five contiguous SNPs can assume only one of $3^5 = 243$ values. Assuming there are no missing values, this format is 20% more memory efficient than storing SNP values in 2 bits. Hence, each realized vector m 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 compressed format, which we coined *5codes*. At multiplication time, we treat the columns of Λ separately and load a vector $\lambda \in \mathbb{R}^5$ of five entries in a column of Λ, which is stored in double precision. Subsequently, we compute all possible results of the scalar product

$$x = m^T \lambda, \quad \text{for } m \in \{0, 1, 2\}^5$$

and store them in a lookup table. Then, we iterate over the five columns of M under consideration and look up the values of x depending on the realization of m. We provide pseudo-code for one iteration of this approach in Algorithm 1. The multiplication $M^T \tilde{\Lambda}$ is achieved analogously.
It is worth noting that at least 16 of these tables fit into the L1 cache because each lookup table holds $3^5 = 243$ double-precision values of 64 bits and the L1 cache in modern CPU architectures typically holds between 32KB and 96KB. Since the lookup table of the different values of x only needs to be computed once for every five-row tile of

Λ, we keep register instructions to a minimum. Furthermore, compressed matrix values are loaded into integer registers whereas real values are stored in SIMD registers. The implementation aims to optimize load operations and store operations. For instance, if SIMD registers of 256-bit width are available, one entry of the lookup table contains a vector of four double-precision floating-point values.

The computation is parallelized among the available processor cores by splitting M and Λ into chunks along the column axis and row axis respectively. Finally, the results of each thread are united by a single reduction operation at the end that computes the sum of all individual results.

As discussed above, genotype centering is not a bottleneck due to its low complexity. Thanks to the structure of the *5codes* encoding, the centering can actually be included in the lookup table for the operation $Z\Lambda$, meaning that instead of holding the possible values of $m^T\lambda$, the lookup table can store the centered values

$$x = z^T\lambda = m^T\lambda - 2 \cdot p^T\lambda.$$

Subtracting $2 \cdot p^T\lambda$ at this point further reduces the number of memory accesses and decreases numerical accumulation errors.

Algorithm 1: Pseudo-code for the *5codes* algorithm for the multiplication of five columns of SNPs M with a vector λ of length 5. The multiplication of five columns of M^T is analogous. Note that the variable `idx` in the algorithm is smaller than 243. For a general number of SNPs k, five columns of M are multiplied at a time.

Data: Compressed SNP matrix $M \in \{0,1,2\}^{n_g \times 5}$, floating-point vector $\lambda \in \mathbb{R}^5$
Result: Genotype matrix multiplication $x = M\lambda$

```
/* Compute and store the dot products of λ with all possible SNP
   vectors                                                      */
for m ∈ {0,1,2}^5 do
    idx ← m_1 × 3^0 + m_2 × 3^1 + m_3 × 3^2 + m_4 × 3^3 + m_5 × 3^4
    /* Compute and store m^T λ                                  */
    table[idx] ← m_1 × λ_1 + m_2 × λ_2 + m_3 × λ_3 + m_4 × λ_4 + m_5 × λ_5
end
/* Compute Mλ by looking up the realization of m in the matrix
   M.                                                           */
for j ← 1 to n_g do
    key ← M_{j,1} × 3^0 + M_{j,2} × 3^1 + M_{j,3} × 3^2 + M_{j,4} × 3^3 + M_{j,5} × 3^4
    x[j] ← table[key]
end
```

SNP matrix multiplication with mixed data types on GPUs

To make use of 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) in its version 2.10. CUTLASS is a C++ template library for high-performance matrix operations on NVIDIA® GPUs. In contrast to CPU operations, GPU functions need to align parallel operations both within thread blocks as well as within warps of threads (Sanders and Kandrot, 2010). CUTLASS assists this task by providing a framework that allows software solutions to target only a subset of levels in this hierarchy. In our case, we implement a template specialization for the Single Instruction Multiple Threads (SIMT) subsection within warp-level API in CUTLASS. Since computing times for our GPU approach are mostly driven by data movements instead of algebraic operations, we do not make use of the *5codes* algorithm in this implementation but distribute scalar products of four-dimensional vectors to the cores of the GPU. Therefore, we keep the established PLINK 8-bit-sized format for storing 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. The microkernel uses bitmasks to extract the SNP values from the compressed storage format and converts them into double-precision floating-point values for immediate multiplication afterward. 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 Λ and transfer data objects which are required in every PCG iteration (that is, M and p) only once 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) who extended the CUTLASS library to include, among others, matrix multiplication of 4-bit integers with half-precision floating-point values.

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 $M^T\tilde{\Lambda}$. A memory-efficient implementation would transpose chunks of M of low dimension which are iterated over. For instance, transposing sub-matrices of dimensions 16 by 16 in PLINK 1 binary format would allow distributing the transpose operation in a warp of threads on a GPU. However, thanks to the compressed data storage, we are not memory-bound with the current number of genotyped animals and SNPs, and we, therefore, choose to transpose M as a whole during start-up and store it separately to reduce computation time associated with transposition. For instance, the dataset of

2.61m animals with 47k SNP markers we use in this article for testing purposes only requires about 57 gigabytes of RAM for both M and M^T combined.

4.2.3 SNP matrix multiplications in single-step models

Single-step models in animal breeding programs commonly comprise hundreds of thousands or even millions of animals. Therefore, the MME for these models 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.
To illustrate the necessity of a fast genotype matrix multiplication, we give an overview of the matrix operations involved in the ssSNPBLUP approach, proposed by Liu et al. (2014), and the ssGTABLUP approach, introduced by Mäntysaari et al. (2017). Both models can be easily extended to multivariate models. The numerical treatment of these approaches is described in detail by Vandenplas et al. (2023) who found that they have similar computational costs per iteration when applied to large datasets since they require the same matrix computations.
A standard univariate mixed model for ssGBLUP can be written as:

$$y = X\beta + \begin{pmatrix} W_n & 0 \\ 0 & W_g \end{pmatrix} \begin{pmatrix} g_n \\ g_g \end{pmatrix} + \varepsilon$$

where y is the vector of records, β is the vector of p fixed effects, g_n is the vector of additive genomic values for the non-genotyped animals, g_g is the vector of additive genomic values for the genotyped animals, and ε is the vector of residuals. The matrices X, W_n, and W_g are incidence matrices relating records in y to the corresponding effects. The random effects vector g_g can be decomposed into $g_g = a_g + Zu$, where u is the vector of SNP effects and a_g contains the residual polygenic effects.
Due to the assumed covariance structure, the ssSNPBLUP system of equations proposed by Liu et al. (2014) involves the matrix $\mathbf{\Sigma}^{-1}$ defined as in Equation (2.16). Furthermore, the matrix

$$A = \begin{pmatrix} A_{nn} & A_{ng} \\ A_{gn} & A_{gg} \end{pmatrix},$$

denotes the pedigree-based relationship matrix and

$$A^{-1} = \begin{pmatrix} A^{nn} & A^{ng} \\ A^{gn} & A^{gg} \end{pmatrix}.$$

is its inverse, which is sparse (Henderson, 1976). The inverse of A_{gg} can be computed as

$$A_{gg}^{-1} = A^{gg} - A^{gn}(A^{nn})^{-1}A^{ng}.$$

Because the coefficient matrices of the ssSNPBLUP and ssGTABLUP models are too large to be constructed explicitly, it is decomposed into submatrices which are multiplied separately (Vandenplas et al., 2020). Importantly, most of these submatrices (e.g., $A^{nn}, A^{ng}, A^{gn}, A^{gg}$) in $\mathbf{\Sigma}^{-1}$ are sparse and can be multiplied by a vector or a matrix at a relatively low cost by using iteration-on-data techniques (Schaeffer and Kennedy, 1986) and sparse matrix operations (Legarra et al., 2009; Vandenplas et al., 2020). Additionally, although the matrix $(A^{nn})^{-1}$ is not sparse in general, equation systems involving $(A^{nn})^{-1}$ can be solved by using forward and backward substitution techniques with the Cholesky factor of A^{nn}, which only needs to be computed once. Therefore, this calculation is also not computationally demanding in practice. In contrast, the multiplication of Z by an arbitrary matrix of low width Λ has been a computational bottleneck so far.

4.2.4 Evaluation

Since both the *5codes* algorithm and our GPU approach take advantage of modern hardware architectures, it can be expected that they outperform the existing implementations. To examine this assumption, we used 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 evaluated the wall clock times for inferring the breeding values in this population required by the PCG implementation in the software MiXBLUP (ten Napel et al., 2021). We compared our novel approaches with the wall clock time required by the current Fortran-based matrix multiplication implementation in MiXBLUP.

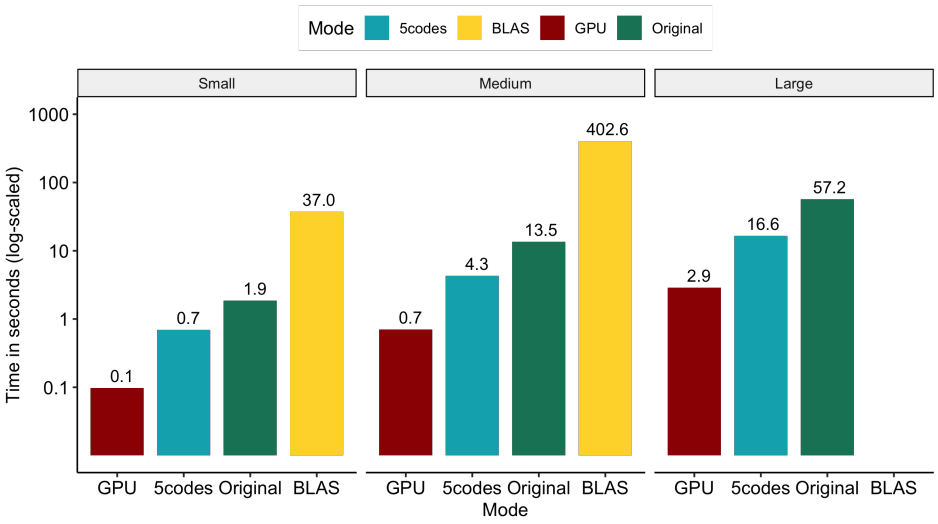
Simulated data

For benchmarking our two novel approaches, we simulated genotype data of various dimensions using the software suite PLINK (Chang et al., 2015). 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: a small population with 102k animals, a medium population with 751k animals and a large population with 3.1m 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 matrix Λ of 10 normally distributed traits.

Cattle data

We tested our two novel approaches when integrated into the PCG solver 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. We solved the

Figure 4.1: Total wall clock time for consecutively calculating $Z\Lambda$ and $Z^T\tilde{\Lambda}$ on a system with a dual-socket AMD® Milan EPYC 7513 (2.6 GHz) processor, 2 TB of RAM and a single NVIDIA® H100 GPU. Annotations show formatted computing times in seconds. CPU calculations used 20 dedicated cores. No test was performed for the BLAS routine with the large population size, as this would have required ca. 2.27 TB of RAM. Results are the average of 10 repeated calculations.



equation systems associated with the ssSNPBLUP and ssGTABLUP models. 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 16.59m data records (across 6 traits), and the pedigree included 26.46m animals. The genotypes of 2.61m animals included 47,006 SNP markers from 29 bovine autosomes, with a minor allele frequency greater or equal to 0.01. The genotype data were from a range of 30 different arrays ranging from 3k to 850k SNPs that had been imputed using FImpute (Sargolzaei et al., 2014) to a 50k SNP set based on version 3 of the International Beef and Dairy (IDB) chip. For both single-step approaches, the genotype matrix was centered using observed allele frequencies and the proportion of residual polygenic effects was set to $w = 0.20$.

4.3 Results

4.3.1 Benchmarks

In Figure 4.1, we assess the performance of consecutively multiplying $Z\Lambda$ and $Z^T\tilde{\Lambda}$, as required in each iteration of a PCG solver. We compare our implementation of the *5codes* algorithm and the GPU implementation with two alternative solutions: 1) The

Table 4.1: Computation wall clock times in seconds of the GPU implementation and the CPU algorithm *5codes* for the multiplication of the genotype matrix Z and its transposed Z^T with simulated matrices $\Lambda, \tilde{\Lambda}$. CPU operations were performed with 2 TB of RAM on the Intel® Xeon Gold 6230 (2.1 GHz) and the AMD® EPYC 7513 (2.6 GHz) on 1 core, 10 cores and 20 cores. The V100 GPUs were equipped with 32 GB of device memory, while the A100 and H100 models had a capacity of 80 GB. Dashes (-) indicate out-of-memory events. Results are the average of 10 repeated calculations.

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

Table 4.2: Computation wall clock times of single-step genomic models on ICBF cattle data in seconds. The SNP matrix Z and its transposed Z^T are multiplied by matrices Λ and $\tilde{\Lambda}$ respectively to compute the candidate matrix in the PCG. 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}. The total time contains preprocessing (I/O operations and set-up of the preconditioner matrix), solving of the MME, and postprocessing (mainly I/O operations).

Wall clock time (s)	**ssSNPBLUP**			**ssGTABLUP**		
	Current	**5codes**	**GPU**	**Current**	**5codes**	**GPU**
Average time $Z\Lambda$	5.41	4.81	1.50	8.76	10.38	1.53
Average time $Z^T\tilde{\Lambda}$	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

decompress-on-the-fly approach implemented in Fortran and proposed by Vandenplas et al. (2020), which we will call "original solution" below, and 2) a single call to the double-precision matrix multiplication function (`dgemm`) which is part of all major BLAS libraries. As for the latter, we first needed to inflate the full SNP matrix to double-precision floating-point values and store both the SNP matrix as well as its transpose in memory. We use the BLAS library included in the Intel® Math Kernel Library (MKL). 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. We ran our CPU benchmarks on a single AMD® Milan EPYC 7513 (2.6 GHz) CPU and our GPU implementation on an NVIDIA® H100. Since our implementation of the *5codes* algorithm did not display considerable scalability advances beyond 20 cores (see Table 4.1) and the software MiXBLUP recommends a similar number of cores for parallelizing computations, we refrain from testing it on more cores.

Evaluating our two novel approaches against a crude call to the BLAS-based double-precision matrix multiplication function shows that computing 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 of RAM, which could not be met in our hardware setup, hindering us from benchmarking the BLAS library on this dataset. Similarly, the CPU-based original solution for multiplying compressed SNP matrices with Λ took about 19 times longer on all population sizes compared to the GPU implementation and three times longer compared to the *5codes* algorithm.

To mitigate the impact of hardware-specific optimizations which might limit the 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 4.1 are the average computing times of 10 replicates. As for the GPU implementation, our findings suggest that reductions in computing times of approx. 25% (50%) can be achieved when using the recently introduced H100 compared to the A100 (V100). More importantly, the V100 ships with 16 GB or 32 GB 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, computing times for both multiplications do not differ substantially.
Despite using the Intel® C/C++ Classic Compiler with Intel-specific optimizations, the *5codes* algorithm performs slightly better on the more powerful AMD® chip than on the Xeon Gold 6230. We observe that our implementation scales reasonably well to 10 cores, decreasing computing 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 (see Table 4.1).
Overall, our evaluations suggest that the GPU implementation significantly outperforms the *5codes* implementation, though practical applications should evaluate the costs and benefits of adding a GPU to their hardware setup based on their respective compute time restraints.

4.3.2 Impact on single-step genomic evaluations

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 two novel implementations through an option. 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 square relative residual below 10^{-13} was achieved. To put our observed computing times into perspective, we also solved both models with the current approach for multiplying genotype matrices implemented in MiXBLUP 3.1 (which we will call "current" below).
Results are displayed in Table 4.2. 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 $Z\Lambda$ and by approx. 95% (76%) for $Z^T\tilde{\Lambda}$ on GPUs (CPUs) respectively. When solving the ssGTABLUP equation system, the average time for multiplying $Z\Lambda$ was slightly higher in the *5codes* algorithm. Yet, both *5codes* and the GPU implementation significantly decreased the time for computing $Z^T\tilde{\Lambda}$.
As the genotype matrix multiplication constitutes a significant portion of the time per

iteration in the PCG solver, the *5codes* implementation reduced the time for solving the ssSNPBLUP model from approx. 6.47 hours to 4.22 hours on the CPU. Similarly, the wall clock time for solving the ssGTABLUP model was reduced from 4.03 hours to 3.30 hours. Outperforming these results, the GPU approach only required 2.49 hours for the ssSNPBLUP model and 1.98 hours for the ssGTABLUP model. A similar number of iterations was observed for solving all models, though the *5codes* algorithm required 27 iterations less for solving the MME associated with the ssSNPBLUP model, which might be explained by its inherently higher precision.

4.4 Discussion

We have presented novel approaches for multiplying centered genotype matrices M by a continuously-scaled matrix Λ which are applicable both on CPUs as well as on modern GPUs. Useful applications include single-step genomic 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 other 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 the results of a principal component analysis (PCA) of G for population stratification (Price et al., 2006; Meuwissen et al., 2017; Ødegård et al., 2018) 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 up to 3 compared to the methodology by Vandenplas et al. (2020) using CPUs, and a factor of up to 20 using GPUs. Thanks to this acceleration, we have shown how our software library can be used by researchers and practitioners to estimate breeding values in a population of 26.46m animals, 2.61m 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 4.2, system memory requirements might be reduced by a factor of approximately two by transposing the compressed genotype matrix on-the-fly during matrix multiplication instead of storing the transpose explicitly. 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 improvement would extend the dimensions of possible problem sizes addressable with our proposed matrix-multiplication microkernel. Second, though our GPU implementation uses highly efficient data access iterators provided by the CUTLASS library, a further reduction in computing 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)). Additionally, we have seen that our *5codes* implementation suffers from scalability issues when extending the number of cores.
Finally, it should be noted that in the evaluation of single-step models, the preprocessing time required by the PCG solver (e.g. to set up the preconditioner) now constitutes a significant portion of the total computation time. Reducing this contribution holds the potential for additional performance improvements.
Notwithstanding these potential improvements, our software can be used in a variety of computational tasks in genomics to reduce computing times.

Chapter 5

Estimation of covariance parameters

One of the core assumptions in the use of SNP markers for quantitative trait modeling is that they are in LD with causal QTL, that is, their genotypic value is linked during genomic assemblage. Yet, correlations between SNPs themselves are often treated as an artifact in predictive models (Calus and Vandenplas, 2018; Chang et al., 2015), while negligent treatment of LD is simultaneously considered a contributor to an incomplete understanding of heritability (de los Campos et al., 2015; Schreck et al., 2019). However, investigating the dependence structure of effect sizes might be equally as interesting: Since their treatment as random is not based on genetic theory, their covariance could be used as an auxiliary tool for capturing the underlying genomic variation, for which empirical data is incomplete.
One such approach has been suggested by Ober et al. (2011): They use the numerator relationship matrix as covariance matrix for the genomic value and include the SNP effects through an additional random term:

$$y_i = X_i\beta + W_i u + g(Z_i) + \varepsilon_i, \quad \text{for} \quad i = 1, ..., n,$$

where X_i, W_i and Z_i denote i-th row of X, W and Z respectively and y_i and ε_i are the i-th entry of y and ε. They assume that $(g(z), z \in \mathbb{R}^k)$ is an isotropic Gaussian random field with Whittle-Matérn covariance function, that is,

$$\mathrm{Cov}(Z_i, Z_j) = C_{p,\nu,\sigma^2}(\|Z_i - Z_j\|),$$

where $\|\cdot\|$ denotes the Euclidean norm and

$$C_{\rho,\nu,\sigma^2}(h) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}h/\rho\right)^{\nu} K_\nu \left(\sqrt{2\nu}h/\rho\right), \quad h \geq 0, \tag{5.1}$$

for $\rho, \nu, \sigma > 0$. Here, K_ν is the modified Bessel function of the second kind (DLMF, 2023, Equation 10.25.1).
The Whittle-Matérn covariance function has the useful property that it coincides with the exponential covariance function for $\nu = \frac{1}{2}$, i.e.,

$$C_{\rho,\frac{1}{2},\sigma^2}(h) = \sigma^2 \exp\left(-\frac{h}{\rho}\right), \quad \rho, \sigma > 0,$$

and with the Gaussian covariance function for $\nu \to \infty$,

$$\lim_{\nu\to\infty} C_{\rho,\nu,\sigma^2}(h) = \sigma^2 \exp\left(-\frac{1}{2}\frac{h^2}{\rho}\right), \quad \rho, \sigma > 0,$$

indicating different levels of smoothness. For a more detailed discussion of this covariance function, the reader is referred to the work by Stein (1999).
As an extension to the classic BLUP model, Ober et al. (2011) use simple kriging and universal kriging for predicting the vector $(g(Z_1), ..., g(Z_n))^T$. To this end, they estimate the parameters ν, ρ and σ^2 through maximum likelihood (ML) by using the R package RandomFields (Schlather et al., 2015). The package infers ML estimates (MLEs) by combining heuristics for good starting values for optimization with the nonlinear optimization procedure implemented in the R function `optim`.
However, this approach motivates a new line of interest. The parameters ρ, ν and σ^2 have a huge impact both on the covariance function as well as on the structure of the random field $(g(z), z \in \mathbb{R}^k)$. Consequently, different parameters would result in vastly different dependence structures of genetic effects. The rising popularity of kernel-based approaches in genomic prediction (Morota and Gianola, 2014) motivates the question of how to infer the parameters ρ, ν and σ^2 accurately from real data.
To start this chapter, we briefly recall how the method of maximum likelihood is used in quantitative genetics. Specifically, we present the ideas behind restricted maximum likelihood (REML) which is widely used for inferring model parameters.
Then, current approaches for obtaining ML estimates of the parameters of the Whittle-Matérn and their limitations are discussed. A novel technique for numerically computing the Whittle-Matérn covariance function and its partial derivatives is presented. This section is based on joint work with Martin Schlather.

5.1 Restricted maximum likelihood

Consider the Mixed Effects Model

$$y = X\beta + Zu + \varepsilon$$

with normally distributed random effect vector $u \sim \mathcal{N}(0, \sigma_u^2 \Sigma_u)$ and error vector $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 \Sigma_\varepsilon)$, independent of u. While the previous applications of this model in this thesis have taken the covariance matrices Σ_u and Σ_ε as given, they are in fact commonly modeled as a function of hyperparameters in quantitative genetics. For instance, the variance of random effects $u_1, ..., u_k$ is usually not assumed to be constant along the genome. Since the variability of random effects is used as an auxiliary tool for capturing genomic variation, the R package `sommer` provides the functionality for inferring SNP variances which differ for each chromosome, for instance.
Therefore, in the following the covariance matrices will be denoted as a function of additional parameters $\gamma \in \Theta_u$ and $\delta \in \Theta_\varepsilon$ for parameter spaces $\Theta_u \subseteq \mathbb{R}^{n_\gamma}, \Theta_\varepsilon \subseteq \mathbb{R}^{n_\delta}$,

resulting in the notation $\Sigma_u(\gamma)$ and $\Sigma_\varepsilon(\delta)$. E.g., when setting $\gamma = \delta = (\rho, \nu)$, the parameterization of the covariance matrices could be modeled as

$$\sigma_u^2 \Sigma_u((\rho_u, \nu_u)) = \left(C_{\rho_u, \nu_u, \sigma_u^2}(|u_i - u_j|)\right)_{i,j=1,\dots,k},$$

and

$$\sigma_\varepsilon^2 \Sigma_\varepsilon((\rho_\varepsilon, \nu_\varepsilon)) = \left(C_{\rho_\varepsilon, \nu_\varepsilon, \sigma_\varepsilon^2}(|\varepsilon_i - \varepsilon_j|)\right)_{i,j=1,\dots,n},$$

where C_{ρ,ν,σ^2} is the Whittle-Matérn covariance function from before. In general, the matrix-valued functions $\gamma \mapsto \Sigma_u(\gamma)$ and $\delta \mapsto \Sigma_\varepsilon(\delta)$ are assumed to be smooth.
Denoting the variance matrix of y by

$$V(\sigma_u^2, \sigma_\varepsilon^2, \gamma, \delta) = \mathrm{Cov}(y) = \sigma_u^2 Z \Sigma_u(\gamma) Z^T + \sigma_\varepsilon^2 \Sigma_\varepsilon(\delta),$$

the log-likelihood ℓ function of y with respect to $\beta, \sigma_u^2, \sigma_\varepsilon^2, \gamma$ and δ is equal to

$$\begin{aligned}\ell^{\mathrm{ML}}(\beta, \sigma_u^2, \sigma_\varepsilon^2, \gamma, \delta) = &-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(\det(V(\sigma_u^2, \sigma_\varepsilon^2, \gamma, \delta))) \\ &- \frac{1}{2}(y - X\beta)^T (V(\sigma_u^2, \sigma_\varepsilon^2, \gamma, \delta))^{-1}(y - X\beta).\end{aligned}$$

Considering the covariance parameters $\theta = (\sigma_u^2, \sigma_\varepsilon^2, \gamma, \delta)$ as fixed and only optimizing with respect to β, the BLUE for β can be retrieved (Henderson, 1963). However, the optimization of the log-likelihood with respect to θ is biased in general. This is commonly illustrated at the example of σ_ε^2 whose MLE does not correct for the number of fixed effects β (Thompson, 2008).
The REML method was first introduced by Patterson and Thompson (1971) to mitigate this flaw and a number of different motivations for this approach have been presented in the literature (Harville, 1974; Verbyla, 1990; Thompson, 2008). REML transforms the data vector y linearly by the transposed of a matrix $K \in \mathbb{R}^{s \times n}$, where $s = n - \mathrm{rank}(X)$ and K is chosen such that its columns are linearly independent and

$$K^T X = 0.$$

The REML log-likelihood function of $K^T y$ is then given by

$$\ell^{\mathrm{REML}}(\theta) = -\frac{s}{2}\log(2\pi) - \frac{1}{2}\log(\det(KV(\theta)K^T)) - \frac{1}{2} y^T K (KV(\theta)K^T)^{-1} K^T y.$$

Theory for the calculus of matrix-valued functions (Magnus, 2019) yields that the partial derivatives of ℓ^{REML} are given by

$$\begin{aligned}\left(\frac{\partial}{\partial\theta_i}\ell^{\mathrm{REML}}\right)(\theta) = &-\frac{1}{2}\mathrm{tr}\left(\left(K^T V(\theta) K\right)^{-1} K^T \left(\frac{\partial}{\partial\theta_i} V\right)(\theta) K\right) \\ &+ \frac{1}{2} y^T K \left(\left(K^T V(\theta) K\right)^{-1} K^T \left(\frac{\partial}{\partial\theta_i} V\right)(\theta) K \left(K^T V(\theta) K\right)^{-1}\right) K^T y.\end{aligned}$$

When assuming that X has full rank $p < n$, the identity

$$K(K^T V(\theta) K)^{-1} K^T = (V(\theta))^{-1} - (V(\theta))^{-1} X (X^T V(\theta) X)^{-1} X^T (V(\theta))^{-1} =: P(\theta),$$

can be shown by analyzing the singular value decomposition of the respective matrices. This identity can then be utilized to derive a representation of ℓ^{REML} independent of K:

$$\begin{aligned}\ell^{\text{REML}}(\theta) = &-\frac{s}{2}\log(2\pi) - \frac{1}{2}\log(\det(V(\theta))) - \frac{1}{2}\log(\det(X^T (V(\theta))^{-1} X)) \\ &+ \frac{1}{2}\log(\det(X^T X)) - \frac{1}{2} z^T P(\theta) z.\end{aligned}$$

This holds the advantage that for obtaining the REML estimates the matrix K does not need to be constructed explicitly which would require searching the kernel space of X.
Similarly, the matrix P can be used to derive representations of the partial derivatives independent of K:

$$\left(\frac{\partial}{\partial\theta_i}\ell^{\text{REML}}\right)(\theta) = -\frac{1}{2}\text{tr}\left(P(\theta)\left(\frac{\partial}{\partial\theta_i}V\right)(\theta)\right) + \frac{1}{2} y^T P(\theta)\left(\frac{\partial}{\partial\theta_i}V\right)(\theta) P(\theta) y,$$

where the cyclic property of the trace was used. Furthermore, the partial derivatives of the second order can be expressed in terms of $P(\theta)$:

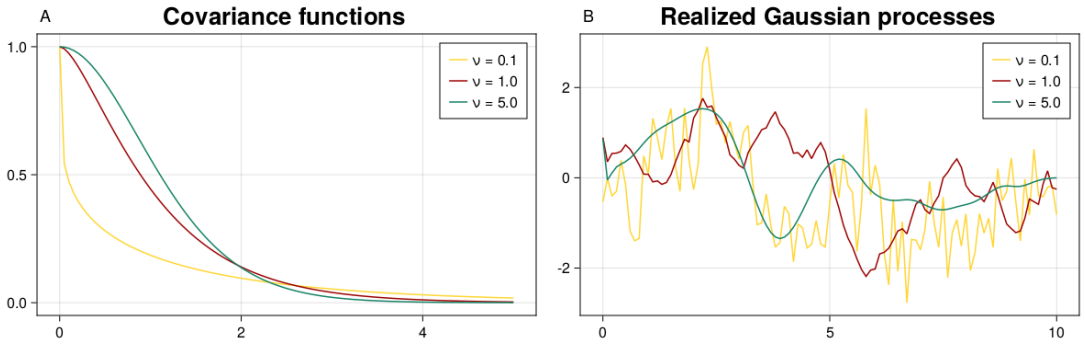
$$\begin{aligned}&\left(\frac{\partial^2}{\partial\theta_j\partial\theta_i}\ell^{\text{REML}}\right)(\theta) \\ =&\frac{1}{2}\text{tr}\left(P(\theta)\left(\frac{\partial}{\partial\theta_j}V\right)(\theta) P(\theta)\left(\frac{\partial}{\partial\theta_i}V\right)(\theta)\right) - \frac{1}{2}\text{tr}\left(P\left(\frac{\partial^2}{\partial\theta_j\partial\theta_i}V\right)(\theta)\right) \\ &- y^T P(\theta)\left(\frac{\partial}{\partial\theta_j}V\right)(\theta) P(\theta)\left(\frac{\partial}{\partial\theta_i}V\right)(\theta) P y + \frac{1}{2} y^T P\left(\frac{\partial^2}{\partial\theta_j\partial\theta_i}V\right)(\theta) P y\end{aligned}$$

More elegant representations for the first-order and second-order partial derivatives with respect to σ_u^2 and σ_ε^2 can be derived. Furthermore, it is often the goal to avoid the second partial derivatives of V when performing numerical optimization of the log-likelihood function by means of Quasi-Newton methods. To this end, the Fisher matrix is employed which can be computed by assuming vector y in the second-order partial derivative to be random and then computing the expectation of the expression:

$$\mathbb{E}_{y\sim\mathcal{N}(0,V(\theta))}\left[\left(\frac{\partial^2}{\partial\theta_j\partial\theta_i}\ell^{\text{REML}}\right)(\theta)\right] = -\frac{1}{2}\text{tr}\left(P(\theta)\left(\frac{\partial}{\partial\theta_j}V\right)(\theta) P(\theta)\left(\frac{\partial}{\partial\theta_i}V\right)(\theta)\right).$$

The Fisher matrix then replaces the role of the Hessian in the Newton method.
Since the trace of a matrix can be quite expensive to compute for large matrices, the Average Information (AI) algorithm proposed by Searle et al. (1992) uses a mixture of Fisher matrix entries and Hessian matrix entries to avoid computationally intensive calculation.

Figure 5.1: Three levels of smoothness for the Whittle-Matérn covariance. Plot A shows the graph of the covariance function $C_{1.0,\nu,1.0}$ for $\nu \in \{0.1, 1.0, 5.0\}$ and parameters $\rho, \sigma^2 = 1.0$. Plot B shows one realization of a Gaussian process on the grid $\{0.0, 0.1, ..., 10.0\}$ for each value of ν.



5.2 Inference of parameters of the Whittle-Matérn covariance function

The Whittle-Matérn covariance function as defined in Equation (5.1) offers flexibility to model a great bandwidth of dependence structures of Gaussian processes. In the left-hand side of Figure 5.1, the graph of the covariance function for three different values of ν (0.1, 1.0 and 5.0) are compared. While the covariance function drops rather quickly when moving away from 0 for $\nu = 0.1$, higher levels of ν lead to slower convergence to zero and smoother covariance functions. On the right-hand side, one realization of a Gaussian time series with Whittle-Matérn covariance function on the equidistant time scale $\{0.0, 0.1, 0.2, ..., 10.0\}$ for each value of ν. It can be observed that the higher values of ν also coincide with smoother trajectories of the Gaussian process.
This illustrates that the parameters of the Whittle-Matérn covariance are important information when this covariance function is used in the modeling of data. Yet, the estimation of these parameters is not straightforward. ML estimates depend on partial derivatives of ℓ^{ML} which are structurally similar to the ones of ℓ^{REML}. Hence, they depend on the partial derivatives which can be computed using calculus for matrix-valued function (Magnus, 2019) to be equal to

$$\left(\frac{\partial}{\partial\theta_i}V\right)(\theta) = \left(\frac{\partial}{\partial\theta_i}V_{j,l}(\theta)\right)_{j,l=1,\dots,n} = Z\frac{\partial}{\partial\theta_i}(\sigma_u^2\Sigma_u(\gamma))Z^T + \frac{\partial}{\partial\theta_i}(\sigma_\varepsilon^2\Sigma_\varepsilon(\delta)).$$

If, for example, $\Sigma_u(\gamma)$ is isotropic, then this partial derivative with respect to γ_i can be expressed as

$$\frac{\partial}{\partial\gamma_i}(\sigma_u^2\Sigma_u(\gamma)) = \left(\frac{\partial}{\partial\gamma_i}C_\gamma(|u_j - u_l|)\right)_{j,l=1,\dots,k}.$$

To return to the Whittle-Matérn covariance function, we will compute its partial derivatives analytically. First, the partial derivative with respect to σ^2 is given by

$$\frac{\partial}{\partial\sigma^2}C_{\rho,\nu,\sigma^2}(h) = \frac{1}{\sigma^2}C_{\rho,\nu,\sigma^2}(h), \quad h > 0. \tag{5.2}$$

As the second-order partial derivatives with respect to σ^2 are similarly simple, $\sigma^2 = 1$ will be assumed in the following to simplify notation.
The partial derivative of $C_{\rho,\nu,1}$ with respect to ρ is given by

$$\begin{aligned}
&\frac{\partial}{\partial\rho}C_{\rho,\nu,1}(h)\\
=&\frac{2^{1-\nu}}{\Gamma(\nu)}\frac{-\sqrt{2\nu}h}{\rho^2} \times \nu\left(\sqrt{2\nu}\frac{h}{\rho}\right)^{\nu-1} K_\nu\left(\sqrt{2\nu}\frac{h}{\rho}\right)\\
&+\frac{2^{1-\nu}}{\Gamma(\nu)}\frac{-\sqrt{2\nu}h}{\rho^2} \times \left(\sqrt{2\nu}\frac{h}{\rho}\right)^{\nu}\left(\frac{\nu}{\sqrt{2\nu}\frac{h}{\rho}}K_\nu\left(\sqrt{2\nu}\frac{h}{\rho}\right) - K_{\nu+1}\left(\sqrt{2\nu}\frac{h}{\rho}\right)\right)\\
=&-\frac{2\nu}{\rho}C_{\rho,\nu,1}(h) + \frac{2\nu}{\rho}C_{\rho,\nu+1,1}\left(h\frac{\sqrt{\nu}}{\sqrt{\nu+1}}\right), \quad h > 0,
\end{aligned}$$

where the formula for the derivative of K_ν with respect to its argument,

$$K_\nu'(x) = \frac{\partial}{\partial x}K_\nu(x) = \frac{\nu}{x}K_\nu(x) - K_{\nu+1}(x), \quad x > 0, \tag{5.3}$$

for $\nu > 0$ (DLMF, 2023, Equation 10.29.2) was used. Additionally,

$$\begin{aligned}
\frac{\partial}{\partial\nu}C_{\rho,\nu,1}(h) =& -2\frac{\Gamma'(\nu)}{\Gamma(\nu)^2}\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right)^{\nu} K_\nu\left(\sqrt{2\nu}\frac{h}{\rho}\right)\\
&+\frac{2}{\Gamma(\nu)}\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right)^{\nu} K_\nu\left(\sqrt{2\nu}\frac{h}{\rho}\right)\left(\log\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right) + \nu\frac{\sqrt{2}\rho}{\sqrt{\nu}h}\frac{h}{2\rho\sqrt{2\nu}}\right)\\
&+\frac{2}{\Gamma(\nu)}\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right)^{\nu}\left(\left(\frac{\partial}{\partial\nu}K_\nu\right)\left(\sqrt{2\nu}\frac{h}{\rho}\right) + K_\nu'\left(\sqrt{2\nu}\frac{h}{\rho}\right)\frac{h}{\sqrt{2\nu}\rho}\right)\\
=&C_{\rho,\nu,1}(h)\cdot\left(\log\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right) + \frac{1}{2} - \psi(\nu)\right)\\
&+\frac{2}{\Gamma(\nu)}\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right)^{\nu}\left(\frac{\partial}{\partial\nu}K_\nu\right)\left(\sqrt{2\nu}\frac{h}{\rho}\right)\\
&+\frac{2}{\Gamma(\nu+1)}\left(\frac{\sqrt{\nu}h}{\sqrt{2}\rho}\right)^{\nu+1} K_\nu'\left(\sqrt{2\nu}\frac{h}{\rho}\right), \quad h > 0,
\end{aligned}$$

where ψ is used to denote the digamma function,

$$\psi(\nu) = \frac{\Gamma'(\nu)}{\Gamma(\nu)}, \quad \nu > 0.$$

Again, the derivative of K_ν with respect to its argument can be expanded as in Equation (5.3).
However, the partial derivative of K_ν with respect to its order poses a computational challenge. In Equation 10.38.2, DLMF (2023) give an expression for this partial derivative in terms of the modified Bessel function of the first kind, I_ν,

$$\frac{\partial}{\partial\nu}K_\nu(x) = \frac{1}{2\sin(\nu\pi)}\left(\frac{\partial}{\partial\nu}I_{-\nu}(x) - \frac{\partial}{\partial\nu}I_\nu(x)\right) - \frac{\pi}{\tan(\nu\pi)}K_\nu(x), \quad x \geq 0, \nu \notin \mathbb{Z} \quad (5.4)$$

In the accompanying Equation 10.38.1, the partial derivative of I_ν is given as

$$\frac{\partial}{\partial\nu}I_\nu(x) = I_\nu(x)\log\left(\frac{x}{2}\right) - \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{\psi(k+1+\nu)}{\Gamma(k+1+\nu)}\frac{(z^2/4)^k}{k!}, \quad x \geq 0, \nu \notin \mathbb{Z}.$$

For $\nu \in \mathbb{Z}$, the derivative is given by Equation 10.38.4:

$$\frac{\partial}{\partial\nu}K_\nu(x) = \frac{\nu!}{2}\left(\frac{x}{2}\right)^{-\nu} \sum_{k=0}^{\nu-1} \left(\frac{x}{2}\right)^k \frac{K_k(x)}{k!(\nu-k)}, \quad x \geq 0, \nu \in \mathbb{Z}.$$

However, it is well-known that the representation (5.4) is numerically unstable and cannot be used for high-precision evaluations.
In general, the question of how to obtain ML estimates for ν has not been sufficiently answered so far. For instance, by using the R function `optim`, the package RandomFields (Schlather et al., 2015) implicitly employs finite differences for approximating the gradient of ℓ^{ML}. Finite differences show a decent performance in regions where the Whittle-Matérn covariance is numerically well-conditioned. An alternative method using the Whittle likelihood, which is based on the periodogram of the realized data, has been proposed recently (Sykulski et al., 2019; Guillaumin et al., 2022). The authors show that this method can recover the true values of the Whittle-Matérn covariance function reasonably well. Yet, this technique is somewhat contrary to the established methods in quantitative genetics and their numerical precision and computational performance would require further investigation.
Another novel approach has been proposed recently by Geoga et al. (2023) in their Julia package BesselK.jl. By dividing the domain of the function $(\nu, x) \mapsto K_\nu(x)$ into subregions and performing series expansions within these subregions, they implement a new implementation of the Bessel-K function which is suitable for automatic differentiation. The authors compare their function against a high-precision implementation in the arbitrary precision library Arb (Johansson, 2017) and find that their function shows competitive absolute accuracy when x is large. However, for small values of x the accuracy deteriorates and especially when $x \leq 1$ and $\nu > 5$, the absolute error is in the order of 10^{-1} which is not sufficient from a numerical point of view. Additionally, the absolute accuracy of the derivatives of first and second order with respect to ν is compared to a reference solution which is provided by tenth-order adaptive finite differences. Again, they find that large values of x provide a good approximation of the reference solution

but for large values of ν and small values of $x \approx 0.25$, the absolute error is up to 10^5 for the first order derivative and even 10^{10} for the second order derivative.

As an alternative, it might have merit to study recently introduced representations of the partial derivatives of K_ν. For instance, Brychkov (2016) derives the expression

$$\begin{aligned}\frac{\partial}{\partial\nu}K_\nu(x) =& \frac{\pi}{2\sin(\pi\nu)}\Bigg(\frac{\pi}{\tan(\pi\nu)}I_\nu(x) - (I_\nu(z) + I_{-\nu}(x))\times \\ & \left(\frac{x^2}{4(1-\nu^2)}{}_3F_4\left[\begin{matrix}1,1,3/2\\2,2,2-\nu,2+\nu\end{matrix}\;\middle|\; x^2\right] + \log(x/2) - \psi(\nu) - \frac{1}{2\nu}\right)\Bigg) + \\ & \frac{1}{4}\Bigg(I_{-\nu}(x)\Gamma^2(-\nu)\left(\frac{x}{2}\right)^{2\nu}{}_2F_3\left[\begin{matrix}\nu,1/2+\nu\\1+\nu,1+\nu,1+2\nu\end{matrix}\;\middle|\; x^2\right] - \\ & I_\nu(x)\Gamma^2(-\nu)\left(\frac{x}{2}\right)^{2\nu}{}_2F_3\left[\begin{matrix}\nu,1/2+\nu\\1+\nu,1+\nu,1+2\nu\end{matrix}\;\middle|\; x^2\right] - \\ & I_\nu(z)\Gamma^2(\nu)\left(\frac{x}{2}\right)^{-2\nu}{}_2F_3\left[\begin{matrix}-\nu,1/2-\nu\\1-\nu,1-\nu,1-2\nu\end{matrix}\;\middle|\; x^2\right]\Bigg), \quad \nu \notin \mathbb{Z},\end{aligned}$$

where ${}_pF_q$ denotes the generalized hypergeometric function (DLMF, 2023, Equation 16.2.1), and a shorter expression involving the Meijer G function (DLMF, 2023, Equation 16.7.1) has been presented by González-Santander (2017)

$$\left(\frac{\partial}{\partial\nu}K\right)(x) = \frac{\nu}{2}\left(\frac{K_\nu(x)}{\sqrt{\pi}}G^{3,1}_{2,4}\left[\begin{matrix}1/2,1\\0,0,\nu,-\nu\end{matrix}\;\middle|\; x^2\right] - \sqrt{\pi}I_\nu(z)G^{4,0}_{2,4}\left[\begin{matrix}1/2,1\\0,0,\nu,-\nu\end{matrix}\;\middle|\; x^2\right]\right),$$

for $\nu > 0, x > 0$.

However, both expressions are hard to evaluate numerically. For the generalized hypergeometric functions of higher-order, a series needs to be computed and our experiments showed that numerical errors in the above representation of $\frac{\partial}{\partial\nu}K_\nu$ tend to accumulate. The Meijer G function is defined as a contour integral on the complex plane whose path depends on the parameters and no implementation of the required form exists. Hence, the numerical computation of this function is not straightforward either.

In the following, we explore a different approach. First, note that the integral representation of the Bessel-K function (Gradshteyn and Ryzhik, 2007, 8.432 Equation 6.) enables the equality

$$
\begin{aligned}
C_{\rho,\nu,1}(h) =& \frac{4^{-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}h}{\rho} \right)^{2\nu} \int_0^\infty \exp\left(-t - \frac{(\sqrt{2\nu}h/\rho)^2}{4t} \right) t^{-\nu-1} \, \mathrm{d}t \\
=& \frac{4^{-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}h}{\rho} \right)^{2\nu} \times \\
& \int_0^\infty \exp\left(-\frac{(\sqrt{2\nu}h/\rho)^2}{u} - \frac{u}{4} \right) \cdot \frac{u^{\nu+1}}{(\sqrt{2\nu}h/\rho)^{2\nu+2}} \frac{(\sqrt{2\nu}h/\rho)^2}{u^2} \, \mathrm{d}u \\
=& \frac{1}{\Gamma(\nu)} \int_0^\infty \exp\left(-\frac{\nu h^2/\rho^2}{2u} - u \right) u^{\nu-1} \, \mathrm{d}u
\end{aligned} \tag{5.5}
$$

where we used the transformation $u = \frac{(\sqrt{2\nu}h/\rho)^2}{t}$. Since the integrand in (5.5) is structurally similar to the integrand involved in the definition of the Gamma function, it is reasonable to assume that this expression can be evaluated numerically to some extent, though the fraction $\frac{\nu h^2/\rho^2}{2u}$ might pose problems. For small values of h and ν, the transformation

$$
\int_0^\infty f_{h,\nu,\rho}(u) \, \mathrm{d}u = \int_{-\infty}^\infty \exp\left(-\frac{\nu h^2}{2\rho^2} e^{-u} - e^u + \nu u \right) \, \mathrm{d}u \tag{5.6}
$$

might be useful to mitigate these numerical problems, where we used the notation

$$
f_{h,\nu,\rho}(u) = \exp\left(-\frac{\nu h^2/\rho^2}{2u} - u \right) u^{\nu-1}.
$$

Furthermore, the connection to the Gamma function allows us to assume that the partial derivatives with respect to ν for the expression in (5.5) can be computed by interchanging differentiation and integration later on. An illustration of the integrands in (5.5) and (5.6) for a number of combinations of ν and h is provided in Figure 5.2.

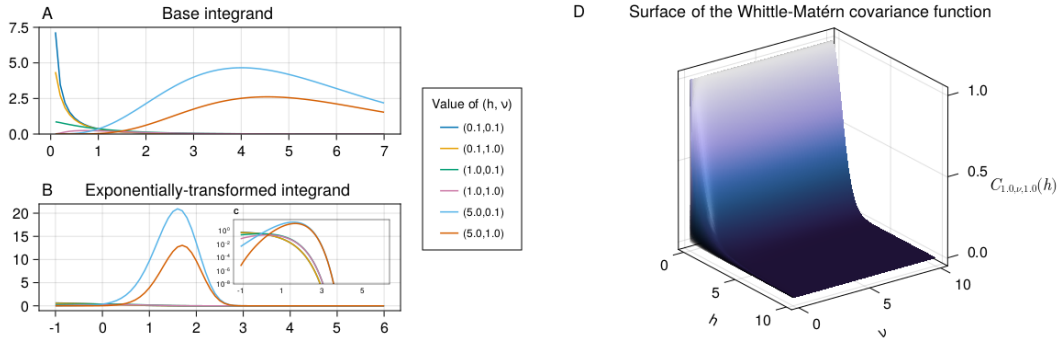
Before we study partial derivatives of the integrals with respect to ν, we first attempt to achieve a decent computation of the covariance function itself. This has the advantage that a high-precision reference implementation as a golden standard can be computed using the Julia package ArbNumeric.jl (Sarnoff, 2023). The package includes functions to evaluate C_{ρ,ν,σ^2} in Equation (5.1) to a precision of 512 bits. Once we have found a method for computing one of the integral representations of the function, we can transfer this method to the partial derivatives since they are structurally similar as we will see later.

We compute our high-precision reference solution on the logarithmic-scaled grid

$$
(h, \nu) \in \{10^{-4.0}, 10^{-3.9}, ..., 10^{0.9}, 10^{1.0}\}^2,
$$

with $\rho = \sigma^2 = 1.0$. The surface of the function on this grid is depicted in Figure 5.2.

Figure 5.2: A: Plot of the integrand $f_{h,\nu,1.0}$ for different values of h and ν. B: Plot of the exponentially transformed integrands in Equation (5.6). C: Plot of the exponentially transformed integrands on a log-scale. D: Surface of the Whittle-Matérn covariance function for $\rho, \sigma^2 = 1.0$ and evaluated on the grid $(h, \nu) \in \{0, 0.1, ..., 10\}^2$.



There are several possibilities for evaluating the integrals (5.5) and (5.6) numerically. First, the connection to the Gamma function allows a stochastic approximation via a Monte Carlo procedure. Specifically, Equation (5.5) yields the relationship

$$C_{\rho,\nu,1}(h) = \mathbb{E}\left[\exp\left(-\frac{\nu h^2}{2\rho^2 X}\right)\right] \approx \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\frac{\nu h^2}{2\rho^2 X^{(i)}}\right),$$

for $X, X^{(1)}, ..., X^{(n)}$ independently $\Gamma(\nu)$-distributed. Unfortunately, the Monte Carlo approximation has a slow convergence rate of $n^{-1/2}$.
Next, another natural attempt for evaluating the integral (5.5) is the numerical integration method by Gauss-Laguerre quadrature. This method uses Laguerre polynomials L_i (DLMF, 2023, Table 18.3.1) to approximate integrands of the form $e^{-x}f(x)$ by

$$\int_0^\infty e^{-x} f(x)\, \mathrm{d}x \approx \sum_{i=1}^{n} f(x_i) w_i,$$

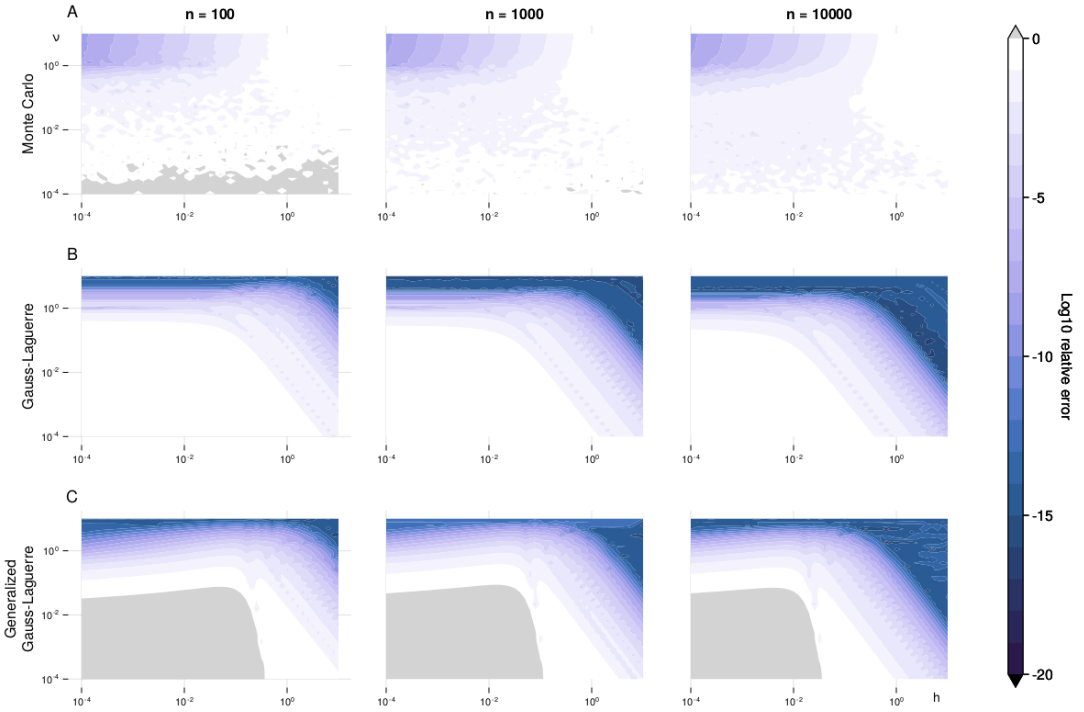
where $x_1, ..., x_n$ are the roots of L_n and

$$w_i = \frac{x_i}{(n+1)^2 L_{n+1}^2(x_i)}, \quad i = 1, ..., n.$$

The motivation for this quadrature method is that the above approximation is exact if f is a polynomial of order small than $2n$.
Employing the generalized Laguerre polynomials L_i^α for $\alpha > -1$, even integrands of the form $x^\alpha e^{-x} f(x)$ can be approximated by

$$\int_0^\infty x^\alpha e^{-x} f(x)\, \mathrm{d}x \approx \sum_{i=1}^{n} f(x_i) w_i,$$

Figure 5.3: Relative error of the Monte Carlo method (row A), Gauss-Laguerre quadrature (row B) and generalized Gauss-Laguerre quadrature (row C) when compared against a high-precision reference solution on a logarithmic grid. Each method was evaluated for values of $n \in \{10^2, 10^3, 10^4\}$, where n is the number of random samples for each point of the grid and the number of nodes and weights for the quadrature methods.



and the nodes $x_1, ..., x_n$ and weights $w_1, ..., w_n$ are adapted to L_n^α.

In Figure 5.3, we compare the Monte Carlo approximation, the Gauss-Laguerre quadrature and the generalized Gauss-Laguerre method with $\alpha = \nu - 1$ for the integral (5.5) against the reference dataset. The plots show the order of the relative error when evaluating these approaches on double-precision floating point values. For the Monte Carlo method, we simulate a number n of $\Gamma(\nu)$-distributed random variables for each point of the grid and compute the arithmetic mean of this sample. To compute the (generalized) Gauss-Laguerre quadrature, we use the Julia package FastGaussQuadrature.jl to compute n nodes and weights based on the (generalized) Laguerre polynomials.

We observe that the approximation by the Monte Carlo method is rather weak across the full domain and increasing the number of samples n does not yield substantial improvements. The Gauss-Laguerre and the generalized Gauss-Laguerre quadrature show decent performance in regions of the domain where h and ν are large. However, small values of h in particular lead to deteriorating approximations. Interestingly, the generalized Gauss-Laguerre method performs worse than the regular Gauss-Laguerre method

although its approach is seemingly better suited to the structure of the integrand. This can be explained by the fact that for $\nu \to 0$, the value of α goes to -1, approaching the boundary of the domain for the generalized Laguerre polynomials. This leads to numerically worse evaluations of the nodes and weights for this method. Additionally, it can be expected that the approximation of the integrand $e^{-\nu h^2/(2u)}$ by polynomials is poor, thereby undercutting one of the core assumptions for a successful application of the Gauss-Laguerre quadrature method.
It is worth noting that our implementation of these three methods does not take advantage of numerical tricks and precision could likely be improved. For instance, Kahan correction could be used to avoid numerical accumulation errors and the integral could be evaluated separately on different regions of $(0, \infty)$. Additionally, the approximation of the integral representation (5.6) might show better numerical behavior. However, in our experiments the achieved improvements through these techniques were marginal and we deemed that further investigation would not increase precision sufficiently.
A lesser-known quadrature method is the tanh-sinh quadrature which was introduced by Takahasi and Mori (1974). In essence, this method applies the transformation

$$\varphi : (-\infty, \infty) \to (-1, 1),\ u \mapsto \tanh\left(\frac{\pi}{2}\sinh(u)\right)$$

to an integrable function f on the domain $(-1, 1)$. The integral then can be approximated by

$$\int_{-1}^{1} f(t)\ \mathrm{d}t = \int_{-\infty}^{\infty} f(\varphi(t))\varphi'(t)\ \mathrm{d}t \approx \sum_{k=-n}^{n} f(x_k)w_k,$$

where

$$x_l = \varphi(lh) \quad \text{and} \quad w_l = h\varphi'(lh) = \frac{h\pi}{2}\frac{\cosh(t)}{\cosh^2\left(\frac{\pi}{2}\sinh(t)\right)}, \quad l = -n, ..., n,$$

for $h > 0$. For integrals on the positive half-axis, the transformation

$$\psi : (-\infty, \infty) \to (0, \infty),\ u \mapsto \exp\left(\frac{\pi}{2}\sinh(u)\right)$$

is suggested instead. Another transformation which uses hyperbolic sine twice is suitable for integration on $(-\infty, \infty)$. Since the hyperbolic functions can be expressed in terms of the exponential function, the tanh-sinh quadrature and the exp-sinh quadrature are also called double-exponential formulae. These quadrature methods are implemented, for example, in the Boost C++ library and the Julia package DoubleExponentialFormulas.jl
In Tanaka et al. (2009), the authors were able to prove bounds on the approximation error for the double-exponential formulae for functions f which are holomorphic on a certain subset of the complex plane. To summarize these results without going into details, if $f : [0, \infty) \to (-\infty, \infty)$ suffices this differentiability condition and

$$|f(x)| \leq C\left|\frac{x^{\beta-1}}{(1+x^2)^{\beta}}\right|, \tag{5.7}$$

Table 5.1: Number of nodes and weights n for exp-sinh quadrature for different combinations of the value of δ^{-1} and precisions of the significand in the arbitrary-precision floating-point format.

Precision	**Value of** δ^{-1}				
	4	8	64	256	1024
32 bits	47	188	749	2993	11972
64 bits	56	224	896	3581	14322
128 bits	65	260	1040	4157	16626
256 bits	74	296	1182	4725	18900

for a $\beta > 0$ and all x in the differentiability subset defined by the parameter $d \in (0, \pi/2)$, then

$$\left| \int_0^\infty f(t)\ \mathrm{d}t - h \sum_{l=-n}^{n} f(\psi(l\delta))\psi'(l\delta) \right| \leq C \exp\left(-\frac{2\pi d}{\delta}\right),$$

for $\delta = \frac{\log(8dn/\beta)}{n}$ and a constant $C > 0$ independent of n. Unfortunately, the underlying theorem from the numerical integration theory for contour integrals does not produce an explicit representation of the constant C.

As the integrand in (5.5) decays exponentially fast for $u \to 0$ and for $u \to \infty$, it satisfies inequality (5.7) for, e.g., $\beta = \nu$. Hence, the exp-sinh quadrature could be considered for computing this integral. To compute the nodes and weights for this method, we consider δ as a tuning parameter and compute the minimum $n_1 \in \mathbb{N}$ for a given $\delta > 0$ such that either $\psi(-(n_1+1)\cdot\delta)$ or $\psi'(-(n_1+1)\cdot\delta)$ is smaller than machine epsilon for the giving floating-point format. Similarly, we compute n_2 to be the minimum integer such that $\psi((n_2+1)\cdot\delta)$ or $\psi'((n_2+1)\cdot\delta)$ is greater than the largest floating-point number representable by the given format. Then, the integers n_1 and n_2 are used as limits for the sum, i.e.

$$\int_0^\infty f(x)\ \mathrm{d}x \approx \sum_{l=-n_1}^{n_2} f(x_l) w_l$$

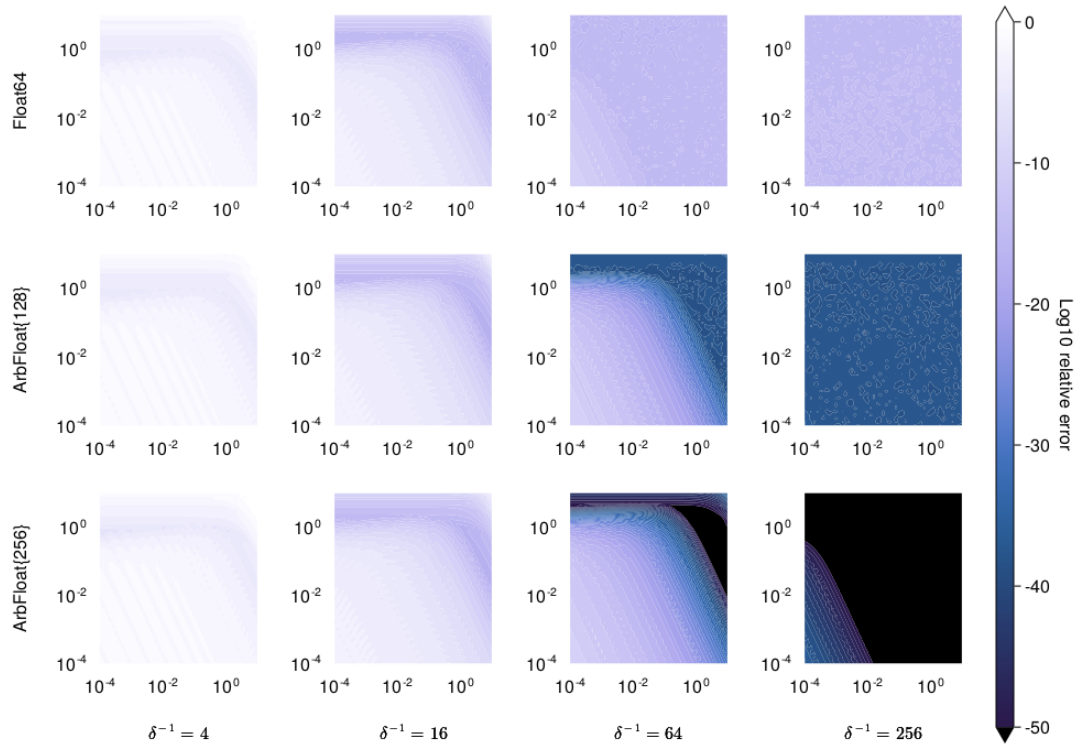
for

$$x_l = \psi(l\delta) \quad \text{and} \quad w_l = \delta\psi'(l\delta), \quad l = -n_1, ..., n_2.$$

The value of δ is chosen as a negative power of 2. Table 5.1 shows the total number of weights and nodes, i.e., $n = n_1 + n_2 + 1$, for different precisions and different values of δ.

In Figure 5.4, we compare our implementation against the reference dataset on the logarithmic grid. We test on three different floating-point formats: double precision of 64 bits ("Float64") and two formats exported by the ArbNumerics.jl library with significands of 128 bits ("ArbFloat{128}") and 256 bits ("ArbFloat{256}") precision

Figure 5.4: Relative error of the exp-sinh quadrature for the integral (5.5) for different combinations of the value of δ and floating-point formats.



respectively. It should be noted that the arbitrary numerics library uses custom floating-point arithmetics of higher precision intrinsically and, therefore, numerical results cannot be compared directly to standard formats.

Yet, we observe that, though small values of δ^{-1} do not suffice to evaluate the integral well, $\delta^{-1} = 64$ already approximates the reference solution close to machine epsilon for the standard double-precision format. Increasing δ^{-1} by another factor of 2 puts the 128-bit precision format within machine epsilon. The 256-bit computation has a relative error smaller than 10^{-50} on the entire grid except when h and ν are both small.

This method holds an additional computational advantage. The standard implementation of the Bessel-K function computes the function values as a series whose coefficients are determined by a recursive sequence (Campbell, 1980) and no vectorized version of this procedure has been published. Considering the small value of δ^{-1} required for a high-precision approximation of the Whittle-Matérn in Figure 5.4, the use of the exp-sinh quadrature method could be beneficial when a large number of evaluations of the Whittle-Matérn covariance function is required. For a satisfactory accuracy on the full logarithmic grid in double precision, $\delta^{-1} = 256$ might be chosen. Since this corresponds to 2720 nodes and weights in double-precision floating-point format, the total memory

requirement for the quadrature table can be computed as

$$2720 * 2 * 8\text{B} \approx 44.52\text{KB}.$$

Considering that many modern CPUs have a capacity of 64KB of L1 cache memory, an efficient implementation of the exp-sinh quadrature could hold the table of nodes and weights in the cache. Alternatively, the domain of (h, ν) could be divided into subregions for evaluating the integral to account for the fact that for $h > 10^{-2}$ smaller values of δ^{-1} yield a sufficient approximation. Furthermore, the integrand $f_{h,\nu,\rho}$ does not rely on special functions anymore and can be fully computed using assembler instructions. In fact, the AVX instruction set extension even provides intrinsics for computing a vectorized version of $f_{h,\nu,\rho}$ using SIMD instructions. Similarly, NVIDIA® GPUs of compute capability 7.0 or newer provide a 64KB block of constant memory which has decreased latency for memory accesses from a thread.
As there is no publicly available implementation of the Bessel-K function for general orders ν on NVIDIA® GPUs, a CUDA function for the exp-sinh quadrature of integrand (5.5) would introduce a convenient way for evaluating the Whittle-Matérn covariance function. Additionally, a GPU implementation could be useful for computing the realized values of a covariance matrix, for example, if a Cholesky decomposition of this matrix is required afterward. In Table 5.2, we compare the performance of a CUDA implementation utilizing a quadrature table stored in constant memory. For this purpose, we simulated a number of $N \in \{5 \cdot 10^3, 10^4, 2 \cdot 10^4\}$ points uniformly distributed on $[0, 1]^2$ and computed their pairwise Euclidean distances. This data of $\frac{N(N-1)}{2}$ values was used as input for h, while $\rho = \sigma^2 = 1.0$ and $\nu = 0.1$ was fixed. Since extremely small distances are unlikely in this setting, we chose to use $\delta^{-1} = 64$, sacrificing a higher precision for better performance. Furthermore, as discussed above, there was no reference implementation available on NVIDIA® GPUs. Hence, we compared our CUDA implementation against a CPU-based multithreaded Whittle-Matérn function based on the original representation (5.1) involving the Bessel-K function. We evaluated this function on a dual-socket AMD® EPYC 7513 (each with a TDP of 200W, 32 cores and 2,840USD price recommendation), while the GPU solution was tested on an NVIDIA® A100 GPU with 40GB of device memory (TDP of 300W, available at approx. 5,000USD). We see that our GPU version performs slightly worse than the CPU implementation, with a markup of about 20%. However, we emphasize that the new solution should only be regarded as a proof-of-concept and more efficient implementations are likely feasible. For instance, when considering fixed ν, the nodes and weights could be rearranged into

$$\tilde{x}_l = \exp\left(-\frac{\nu}{2l\delta}\right) \quad \text{and} \quad \tilde{w}_l = \exp(-l\delta)(l\delta)^{\nu-1} w_l \quad l = -n_1, ..., n_2,$$

so that

$$\int_0^\infty f_{h,\nu,\rho}(u)\ \mathrm{d}u = \sum_{l=-n_1}^{n_2} (\tilde{x}_l)^{h^2} \tilde{w}_l.$$

This would result in an additional reduction in the number of arithmetic operations if precision can be maintained. Furthermore, bringing the required number of weights and

Table 5.2: Wall-clock times times for evaluating the Whittle-Matérn covariance function $C_{1,0.1,1}$ on pairwise Euclidean distances between N random samples with probability distribution $\mathcal{U}([0,1]^2)$. CPU computations were based on the representation of $C_{1,0.1,1}$ in terms of the Bessel-K function while the GPU implementation used exp-sinh quadrature. Calculations were performed on a dual-socket AMD® EPYC 7513 with 64 dedicated cores and an NVIDIA® A100-40GB. Mean relative error was computed as the average absolute difference between the values of CPU and GPU computations over the CPU function values.

Number of samples N	Number of evaluations $\frac{N(N-1)}{2}$	Computing times CPU	Computing times GPU	Mean rel. deviation
5,000	$\sim 1.2 \cdot 10^7$	0.08s	0.11s	$5.13 \cdot 10^{-16}$
10,000	$\sim 5.0 \cdot 10^7$	0.37s	0.45s	$5.14 \cdot 10^{-16}$
20,000	$\sim 2.0 \cdot 10^8$	1.71s	1.81s	$5.14 \cdot 10^{-16}$

nodes for a precise evaluation down to 500 might yield disproportionate speed gains as the constant memory cache on NVIDIA® GPUs only has a capacity of 8KB.
For evaluating the partial derivatives of C_{ρ,ν,σ^2}, it is not obvious what a good reference solution might be as there is no high-precision implementation of the derivative of K_ν with respect to ν. Therefore, let us consider the partial derivative of the integral (5.5) for now. Since

$$\left(\frac{\partial}{\partial\nu} f_{h,\nu,\rho}\right)(u) = \exp\left(-\frac{\nu h^2/\rho^2}{2u} - u\right) u^{\nu-1} \left(\log(u) - \frac{h^2/\rho^2}{2u}\right)$$

decays exponentially fast for $u \to 0$ and $u \to \infty$, it can be bounded by an integrable function locally in ν. Therefore, the partial derivative of $C_{\rho,\nu,1}(h)$ is equal to

$$\left(\frac{\partial}{\partial\nu} C_{\rho,\nu,1}\right)(h) = \frac{1}{\Gamma(\nu)} \int_0^\infty \left(\frac{\partial}{\partial\nu} f_{h,\nu,\rho}\right)(u)\, \mathrm{d}u - \psi(\nu) C_{\rho,\nu,1}(h) \quad h > 0.$$

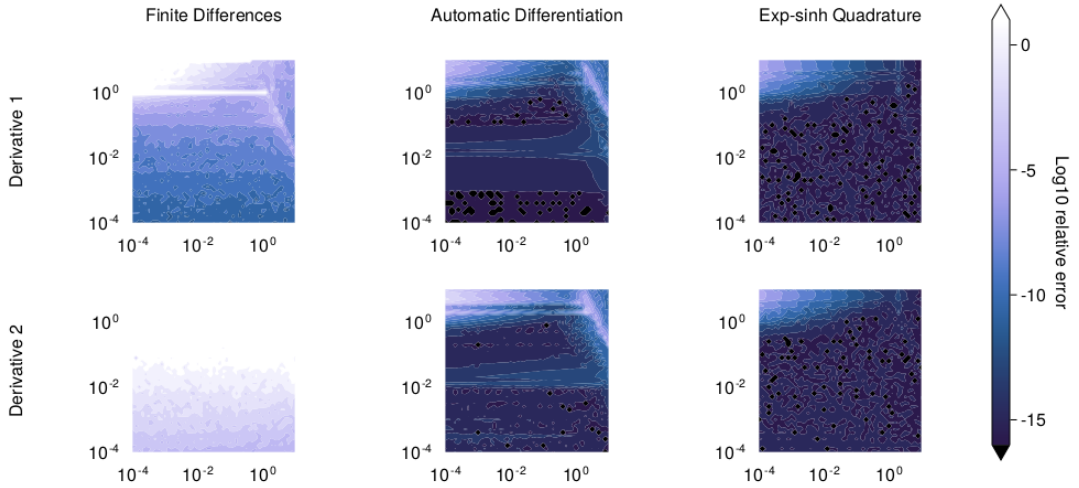
Again, the partial derivative of the integrand $\frac{\partial}{\partial\nu} f_{h,\nu,\rho}$ satisfies the conditions for interchanging integration and differentiation with respect to ν and, hence, the second-order partial derivative of $C_{\rho,\nu,1}$ can be computed as

$$\begin{aligned}\left(\frac{\partial^2}{\partial\nu^2} C_{\rho,\nu,1}\right)(h) =& \frac{1}{\Gamma(\nu)} \int_0^\infty \left(\frac{\partial^2}{\partial\nu^2} f_{h,\nu,\rho}\right)(u)\, \mathrm{d}u \\ &- 2\psi(\nu) \left(\frac{\partial}{\partial\nu} C_{\rho,\nu,1}\right)(h) - \psi'(\nu) C_{\rho,\nu,1}(h), \quad h > 0,\end{aligned}$$

where ψ' denotes the derivatives of the digamma function, also known as the trigamma function, and

$$\left(\frac{\partial^2}{\partial\nu^2} f_{h,\nu,\rho}\right)(u) = \exp\left(-\frac{\nu h^2/\rho^2}{2u} - u\right) u^{\nu-1} \left(\log(u) - \frac{h^2/\rho^2}{2u}\right)^2.$$

Figure 5.5: Relative error in the approximation of first-order and second-order partial derivatives of the Whittle-Matérn covariance function with respect to its smoothness parameter ν. Evaluated are tenth-order finite differences, automatic differentiation provided by BesselK.jl (Geoga et al., 2023) and exp-sinh quadrature with $\delta^{-1} = 256$. All methods were computed in double-precision floating-point format.



Since the integrands $\frac{\partial}{\partial \nu} f_{h,\nu,\rho}$ and $\frac{\partial^2}{\partial \nu^2} f_{h,\nu,\rho}$ also decay exponentially fast for $u \to 0$ and $u \to \infty$, it is reasonable to assume that exp-sinh quadrature yields a decent approximation. Furthermore, high-precision implementations of the digamma function and the trigamma function are available in many special functions libraries and hence a quadrature-based computation of the partial derivatives of C_{ρ,ν,σ^2} with respect to ν is worth investigating.

While the authors of Geoga et al. (2023) opted for an adaptive finite difference method as a reference implementation for the partial derivatives in their analysis, this approach showed insufficient numerical stability for small values of h and ν in our analysis. Therefore, we decided to compute a high-precision reference dataset for the integrals of $\frac{\partial}{\partial \nu} f_{h,\nu,\rho}$ and $\frac{\partial^2}{\partial \nu^2} f_{h,\nu,\rho}$ for $(h, \nu) \in \{10^{-4}, 10^{-3.9}, ..., 10^{1}\}$ based on exp-sinh quadrature. Considering the sound approximation quality of this method for integrating $f_{h,\nu,\rho}$ and the similar behavior of the integrands for $u \to 0$ and $u \to \infty$, using exp-sinh quadrature with $\delta^{-1} = 1024$ and the arbitrary-precision floating-point format with 1024 bits of precision could be reasonably presumed to deliver a more precise reference dataset as a golden standard than finite differences.

In Figure 5.5, we compare the accuracy of exp-sinh quadrature at computing the first-order and second-order partial derivatives of $C_{1,\nu,1}(h)$ on the logarithmic grid $(h, \nu) \in \{10^{-4}, 10^{-3.9}, ..., 10^{1}\}^2$. It is apparent that the approximation by finite differences is rather poor, especially for the partial derivatives of the second order. Both the automatic differentiation method, as implemented in the Julia package BesselK.jl, as well as the

exp-sinh quadrature show a decreasing accuracy for large ν and small h. However, in the region where $\nu \sim 10^{-2}$ the exp-sinh quadrature shows a higher precision. Potentially, this could be attributed to the branching behavior of the implementation of the Bessel-K function in BesselK.jl.
Despite these promising first results, a number of questions remain to be answered for an application in ML estimation. For instance, it needs to be investigated how the higher numerical accuracy in the computation of the partial derivatives of the Whittle-Matérn covariance function transfers to the partial derivatives of the log-likelihood and how this impacts the convergence behavior of optimization algorithms. Furthermore, it will be interesting to study if a vectorized implementation of the exp-sinh quadrature could improve computing times for the evaluation of the covariance function itself. Nevertheless, the increased precision we demonstrated for our novel technique for evaluating the Whittle-Matérn covariance function and its partial derivatives has the potential to mitigate some of the numerical issues that have been known to be problematic in the inference of its parameters.

Bibliography

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 Transactions on Parallel Computing* 7. doi: 10.1145/3399732

Alkhnbashi, O. S., Meier, T., Mitrofanov, A., Backofen, R., and Voß, B. (2020). CRISPR-Cas bioinformatics. *Methods* 172, 3–11. doi: 10.1016/j.ymeth.2019.07.013

Barrett, J. C., Fry, B., Maller, J., and Daly, M. J. (2004). Haploview: Analysis and visualization of LD and haplotype maps. *Bioinformatics* 21, 263–265. doi: 10.1093/bioinformatics/bth457

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review* 59, 65–98. doi: 10.1137/141000671

Brychkov, Y. (2016). Higher derivatives of the Bessel functions with respect to the order. *Integral Transforms and Special Functions* , 1–12doi: 10.1080/10652469.2016.1164156

Butler, DG., Cullis, BR., Gilmour, AR., Gogel, BJ., and Thompson, R. (2017). ASReml-R reference manual version 4. *VSN International Ltd, Hemel Hempstead, HP1 1ES, UK*

Bycroft, C., Freeman, C., Petkova, D., Band, G., Elliott, L., Sharp, K., et al. (2018). The UK Biobank resource with deep phenotyping and genomic data. *Nature* 562. doi: 10.1038/s41586-018-0579-z

Calus, M. P. L. and Vandenplas, J. (2018). SNPrune: an efficient algorithm to prune large SNP array and sequence datasets based on high linkage disequilibrium. *Genetics Selection Evolution* 50, 34. doi: 10.1186/s12711-018-0404-z

Campbell, J. B. (1980). On Temme's algorithm for the modified bessel function of the third kind. *ACM Trans. Math. Softw.* 6, 581–586. doi: 10.1145/355921.355928

Canela-Xandri, O., Rawlik, K., Woolliams, J. A., and Tenesa, A. (2016). Improved genetic profiling of anthropometric traits using a big data approach. *PloS one* 11, e0166755

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

Christensen, O. and Lund, M. (2010). Genomic prediction when some animals are not genotyped. *Genetics Selection Evolution* 42, 2. doi: 10.1186/1297-9686-42-2

Cunningham, E. P. and Henderson, C. R. (1968). An iterative procedure for estimating fixed effects and variance components in Mixed Model situations. *Biometrics* 24, 13–25. doi: 10.2307/2528457

Cunningham, F., Allen, J. E., Allen, J., Alvarez-Jarreta, J., Amode, M. R., Armean, I. M., et al. (2021). Ensembl 2022. *Nucleic Acids Research* 50, D988–D995. doi: 10.1093/nar/gkab1049

de los Campos, G., Sorensen, D., and Gianola, D. (2015). Genomic heritability: What is it? *PLOS Genetics* 11, 1–21. doi: 10.1371/journal.pgen.1005048

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, eds. A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. vol. 35, 30318–30332

DLMF (2023). *NIST Digital Library of Mathematical Functions*. `https://dlmf.nist.gov/`, Release 1.1.10 of 2023-06-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

Endelman, J. B. (2011). Ridge regression and other kernels for genomic selection with R package rrBLUP. *The plant genome* 4, 250–255

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

Falconer, D. S. (1996). *Introduction to Quantitative Genetics* (Pearson Education India)

Fernando, R. L., Dekkers, J. C., and Garrick, D. J. (2014). A class of Bayesian methods to combine large numbers of genotyped and non-genotyped animals for whole-genome analyses. *Genetics Selection Evolution* 46, 1–13. doi: 10.1186/1297-9686-46-50

Fisher, R. A. (1919). XV.—The correlation between relatives on the supposition of Mendelian inheritance. *Earth and Environmental Science Transactions of the Royal Society of Edinburgh* 52, 399–433

Freudenberg, A., Schlather, M., Moerkotte, G., and Pook, T. (2023a). miraculix: Accelerated Computations for Genomic Analysis. *Manuscript submitted.*

Freudenberg, A., Schlather, M., Vandenplas, J., and Evans, R. (2023b). Accelerating single-step evaluations through GPU offloading. *Manuscript in preparation*

Freudenberg, A., Vandenplas, J., Schlather, M., Pook, T., Evans, R., and ten Napel, J. (2023c). Accelerated matrix-vector multiplications for matrices involving genotype covariates with applications in genomic prediction. *Accepted for publication in Frontiers in Genetics.*

Garrick, D. J., Garrick, D. P., and Golden, B. (2018). An introduction to BOLT software for genetic and genomic evaluations. Proceedings of the 11th World Congress on Genetics Applied to Livestock Production

Gazal, S., Finucane, H. K., Furlotte, N. A., Loh, P.-R., Palamara, P. F., Liu, X., et al. (2017). Linkage disequilibrium–dependent architecture of human complex traits shows action of negative selection. *Nature genetics* 49, 1421–1427

Geoga, C. J., Marin, O., Schanen, M., and Stein, M. L. (2023). Fitting Matérn smoothness parameters using automatic differentiation. *Statistics and Computing* 33. doi: 10.1007/s11222-022-10127-w

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*

Gilmour, A. R., Gogel, B. J., Cullis, B. R., Welham, SJ., and Thompson, R. (2002). *ASReml User Guide Release 1.0.* Hemel Hempstead

Giovanny, C.-P. (2016). Genome assisted prediction of quantitative traits using the R package sommer. *PLoS ONE* 11, 1–15

González-Santander, J. (2017). Closed form expressions for derivatives of Bessel functions with respect to the order. *Journal of Mathematical Analysis and Applications* 466. doi: 10.1016/j.jmaa.2018.06.043

Gradshteyn, I. S. and Ryzhik, I. M. (2007). *Table of Integrals, Series, and Products* (Elsevier/Academic Press, Amsterdam), seventh edn.

Granato, I. S. C., Galli, G., Couto, E. G. D. O., Souza, M. B. E., Mendonça, L. F., and Fritsche-Neto, R. (2018). snpReady: A tool to assist breeders in genomic analysis. *Molecular Breeding* 38. doi: 10.1007/s11032-018-0844-8

Guillaumin, A. P., Sykulski, A. M., Olhede, S. C., and Simons, F. J. (2022). The debiased spatial whittle likelihood. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 84, 1526–1557

Halko, N., Martinsson, P.-G., and Tropp, J. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review* 53, 217–288. doi: 10.1137/090771806

Harville, D. A. (1974). Bayesian inference for variance components using only error contrasts. *Biometrika* 61, 383–385. doi: 10.1093/biomet/61.2.383

Hemani, G., Knott, S., and Haley, C. (2013). An Evolutionary Perspective on Epistasis and the Missing Heritability. *PLOS Genetics* 9, e1003295. doi: 10.1371/journal.pgen.1003295

Henderson, C. R. (1949). Estimation of changes in herd environment. *J. Dairy Sci* 32, 706–706

Henderson, C. R. (1963). Selection Index and Expected Genetic Advance. In *Statistical Genetics and Plant Breeding* (Washington, DC: The National Academies Press), vol. 982. 141–163. doi: 10.17226/20264

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 6869–6898

Jiang, L., Zheng, Z., Qi, T., Kemper, K. E., Wray, N. R., Visscher, P. M., et al. (2019). A resource-efficient tool for mixed model association analysis of large-scale data. *Nature genetics* 51, 1749–1755

Jiang, Y. and Reif, J. C. (2015). Modeling epistasis in genomic selection. *Genetics* 201, 759–768

Johansson, F. (2017). Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers* 66, 1281–1292. doi: 10.1109/TC.2017.2690633.

Kim, H., Grueneberg, A., Vazquez, A. I., Hsu, S., and de los Campos, G. (2017). Will big data close the missing heritability gap? *Genetics* 207, 1135–1145. doi: 10.1534/genetics.117.300271

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)*, eds. A. Fan, I. Gurevych, Y. Hou, Z. Kozareva, S. Luccioni, N. S. Moosavi, S. Ravi, G. Kim, R. Schwartz, and A. Rücklé (Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics), 36–43

Krishna, R. and Elisseev, V. (2021). User-centric genomics infrastructure: trends and technologies. *Genome* 64, 467–475. doi: 10.1139/gen-2020-0096

Lee, S. H. and van der Werf, J. H. J. (2016). MTG2: An efficient algorithm for multivariate linear mixed model analysis based on genomic information. *Bioinformatics* 32, 1420–1422. doi: 10.1093/bioinformatics/btw012

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

Legarra, A., Lourenco, D., and Vitezica, Z. (2022). Bases for Genomic Prediction. Available at https://genoweb.toulouse.inra.fr/~alegarra/GSIP.pdf. Last accessed 24/07/2023. Version 0.9.1

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

Loiselle, B. A., Sork, V. L., Nason, J., and Graham, C. (1995). Spatial Genetic Structure of a Tropical Understory Shrub, Psychotria officinalis (Rubiaceae). *American Journal of Botany* 82, 1420–1425. doi: 10.2307/2445869

Magnus, J. R. (2019). *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley Series in Probability and Statistics (Hoboken, NJ: Wiley), third edn.

Manolio, T. A., Collins, F. S., Cox, N. J., Goldstein, D. B., Hindorff, L. A., Hunter, D. J., et al. (2009). Finding the missing heritability of complex diseases. *Nature* 461, 747–753. doi: 10.1038/nature08494

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

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. doi: 10.2527/jas2017.1912

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

Meuwissen, T. H. E., Hayes, B., and Goddard, M. E. (2001). Prediction of total genetic value using genome-wide dense marker maps. *Genetics* 157, 1819–1829. doi: 10.1093/genetics/157.4.1819

Misztal, I. and Legarra, A. (2017). Invited review: Efficient computation strategies in genomic selection. *Animal* 11, 731–736. doi: 10.1017/S1751731116002366

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. (2014a). 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)*, eds. R. F. Veerkamp and Y. de Haas (Wageningen Academic Publishers), 2757–2760. doi: 10.3920/978-90-8686-940-4

Misztal, I., Tsuruta, S., Lourenco, D., Masuda, Y., Aguilar, I., Legarra, A., et al. (2014b). *Manual for BLUPF90 Family of Programs*. University of Georgia, Athens. Available at http://nce.ads.uga.edu/wiki/lib/exe/fetch.php?media=blupf90_all8.pdf. Last accessed 02/08/2023.

Morota, G. and Gianola, D. (2014). Kernel-based whole-genome prediction of complex traits: a review. *Frontiers in genetics* 5, 363

Mrode, R. A. (2014). *Linear Models for the Prediction of Animal Breeding Values* (Cabi)

Mäntysaari, E. and Strandén, I. (2016). Single-step genomic evaluation with many more genotyped animals. In *Proceedings of the 67th Annual Meeting of the European Association for Animal Production, , Belfast, UK, 29 August-2 September 2016* (Wageningen Academic Publishers), 336

NVIDIA (2023). *Parallel Thread Execution ISA - Application Guide*. NVIDIA Corporation & Affiliates. Available at https://docs.nvidia.com/cuda/parallel-thread-execution. Last accessed 02/08/2023. Version 8.2.

Ober, U., Erbe, M., Long, N., Porcu, E., Schlather, M., and Simianer, H. (2011). Predicting genetic values: A kernel-based best linear unbiased prediction with genomic data. *Genetics* 188, 695–708

Ocaña, K. and de Oliveira, D. (2015). Parallel computing in genomic research: advances and applications. *Advances and Applications in Bioinformatics and Chemistry* 8, 23–35. doi: 10.2147/AABC.S64482

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, 6. doi: 10.1186/s12711-018-0373-2

O'Connell, K. A., Yosufzai, Z. B., Campbell, R. A., Lobb, C. J., Engelken, H. T., Gorrell, L. M., et al. (2023). Accelerating genomic workflows using NVIDIA Parabricks. *BMC bioinformatics* 24, 1–15. doi: 10.1186/s12859-023-05292-2

Pallares, L. (2019). Searching for solutions to the missing heritability problem. *eLife* 8, e53018. doi: 10.7554/eLife.53018

Patterson, H. D. and Thompson, R. (1971). Recovery of inter-block information when block sizes are unequal. *Biometrika* 58, 545–554. doi: 10.1093/biomet/58.3.545

Pook, T., Reimer, C., Freudenberg, A., Büttgen, L., Geibel, J., Ganesan, A., et al. (2021). The Modular Breeding Program Simulator (MoBPS) allows efficient simulation of complex breeding programs. *Animal Production Science* 61, 1982–1989

Pook, T., Schlather, M., and Simianer, H. (2020). MoBPS - modular breeding program simulator. *G3: Genes, Genomes, Genetics* 10, 1915–1918. doi: 10.1534/g3.120.401193

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. doi: 10.1038/ng1847

Pritchard, J. K. and Przeworski, M. (2001). Linkage disequilibrium in humans: models and data. *The American Journal of Human Genetics* 69, 1–14. doi: 10.1086/321275

Rampazo Amadeu, R., Cellon, C., Olmestead, J. W., Franco Garcia, A. A., and Resende Jr, M. F. (2016). AGHmatrix: R package to construct relationship matrices for autotetraploid and diploid species: a blueberry example. *The Plant Genome* 9, 1–10. doi: 10.3835/plantgenome2016.01.0009

Rohde, P. D., Fourie Sørensen, I., and Sørensen, P. (2019). qgg: An R package for large-scale quantitative genetic analyses. *Bioinformatics* 36, 2614–2615. doi: 10.1093/bioinformatics/btz955

Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming* (Addison-Wesley Professional)

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

Sarnoff, J. (2023). ArbNumerics.jl. https://github.com/JeffreySarnoff/ArbNumerics.jl. Version 1.3.4

Schaeffer, L. R. and Kennedy, B. W. (1986). Computing strategies for solving mixed model equations. *Journal of Dairy Science* 69, 575–579. doi: 10.3168/jds.S0022-0302(86)80441-6

Schaeffer, LR. (2006). Strategy for applying genome-wide selection in dairy cattle. *Journal of Animal Breeding and Genetics* 123, 218–223. doi: 10.1111/j.1439-0388.2006.00595.x

Schlather, M. (2020). Efficient calculation of the genomic relationship matrix. *Preprint on bioRxiv* doi: 10.1101/2020.01.12.903146

Schlather, M., Malinowski, A., Menck, P. J., Oesting, M., and Strokorb, K. (2015). Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software* 63, 1–25. doi: 10.18637/jss.v063.i08

Schlather, M. and Reinbott, F. (2021). A semi-group approach to principal component analysis. doi: 10.48550/arXiv.2112.04026. Preprint on arxiv.

Schreck, N. (2018). *From Estimation to Prediction of Genomic Variances: Allowing for Linkage Disequilibrium and Unbiasedness*. Ph.D. thesis, University of Mannheim, Mannheim

Schreck, N., Piepho, H.-P., and Schlather, M. (2019). Best prediction of the additive genomic variance in random-effects models. *Genetics* 213, 379–394. doi: 10.1534/genetics.119.302324

Schwarze, K., Buchanan, J., Taylor, J. C., and Wordsworth, S. (2018). Are whole-exome and whole-genome sequencing approaches cost-effective? A systematic review of the literature. *Genetics in Medicine* 20, 1122–1130. doi: 10.1038/gim.2017.247

Searle, S., Casella, G., and McCulloch, C. (1992). *Variance Components*. Wiley Series in Probability and Statistics (Wiley)

Shi, L. and Wang, Z. (2019). Computational strategies for scalable genomics analysis. *Genes* 10, 12. doi: 10.3390/genes10121017

Singh, R. K. and Prasad, M. (2021). Big genomic data analysis leads to more accurate trait prediction in hybrid breeding for yield enhancement in crop plants. *Plant Cell Reports* 40, 2009–2011. doi: 10.1007/s00299-021-02761-x

Solberg, L. C., Valdar, W., Gauguier, D., Nunez, G., Taylor, A., Burnett, S., et al. (2006). A protocol for high-throughput phenotyping, suitable for quantitative trait analysis in mice. *Mammalian Genome* 17, 129–146. doi: 10.1007/s00335-005-0112-1

Stein, M. L. (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics (Springer Science & Business Media)

Steyn, Y., Masuda, Y., Tsuruta, S., Lourenco, D., Misztal, I., and Lawlor, T. (2022). Identifying influential sires and distinct clusters of selection candidates based on genomic relationships to reduce inbreeding in the US Holstein. *Journal of Dairy Science* 105, 9810–9821. doi: 10.3168/jds.2022-22143

Strandén, I. and Lidauer, M. (1999). Solving large mixed linear models using preconditioned conjugate gradient iteration. *Journal of Dairy Science* 82, 2779–2787. doi: 10.3168/jds.S0022-0302(99)75535-9

Sykulski, A. M., Olhede, S. C., Guillaumin, A. P., Lilly, J. M., and Early, J. J. (2019). The debiased whittle likelihood. *Biometrika* 106, 251–266. doi: 10.1093/biomet/asy071

Takahasi, H. and Mori, M. (1974). Double exponential formulas for numerical integration. *Publications of the Research Institute for Mathematical Sciences* 9, 721–741. doi: 10.2977/PRIMS/1195192451

Tanaka, K., Sugihara, M., Murota, K., and Mori, M. (2009). Function classes for double exponential integration formulas. *Numerische Mathematik* 111, 631–655. doi: 10.1007/s00211-008-0195-1

Tanenbaum, A. S. (2013). *Structured Computer Organization*. Always Learning (Pearson Education India), sixth edn.

Taskinen, M., Mäntysaari, E., and Strandén, I. (2017). Single-step SNP-BLUP with on-the-fly imputed genotypes and residual polygenic effects. *Genetics Selection Evolution* 49, 36. doi: 10.1186/s12711-017-0310-9

Taylor-Weiner, A., Aguet, F., Haradhvala, N. J., Gosai, S., Anand, S., Kim, J., et al. (2019). Scaling computational genomics to millions of individuals with GPUs. *Genome Biology* 20, 228. doi: 10.1186/s13059-019-1836-7

ten Napel, J., Vandenplas, J., Lidauer, M. H., Strandén, I., Taskinen, M., Mäntysaari, E. A., et al. (2021). *MiXBLUP 3.0.1 Manual*. Wageningen, the Netherlands. Version 3.0. Last accessed 16/07/2023

Thakkar, V., Ramani, P., Cecka, C., Shivam, A., Lu, H., Yan, E., et al. (2023). *CUTLASS*. NVIDIA Corporation & Affiliates. https://github.com/NVIDIA/cutlass. Version 3.0.0

Thompson, E. A. and Shaw, R. G. (1990). Pedigree Analysis for Quantitative Traits: Variance Components without Matrix Inversion. *Biometrics* 46, 399–413. doi: 10.2307/2531445

Thompson, R. (2008). Estimation of quantitative genetic parameters. *Proceedings of the Royal Society B: Biological Sciences* 275, 679–686. doi: 10.1098/rspb.2007.1417

Vandenplas, J., Calus, M. P. L., Eding, H., and Vuik, C. (2019). A second-level diagonal preconditioner for single-step SNPBLUP. *Genetics Selection Evolution* 51, 30. doi: 10.1186/s12711-019-0472-8

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. doi: 10.1186/s12711-020-00543-9

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. doi: 10.1186/s12711-018-0429-3

Vandenplas, J., ten Napel, J., Darbaghshahi, S. N., Evans, R., Calus, M. P. L., Veerkamp, R., et al. (2023). Efficient large-scale single-step evaluations and indirect genomic prediction of genotyped selection candidates. *Genetics Selection Evolution* 55, 37. doi: 10.1186/s12711-023-00808-z

VanRaden, P. (2008). Efficient methods to compute genomic predictions. *Journal of Dairy Science* 91, 4414–4423. doi: 10.3168/jds.2007-0980

Venkataramani, V., Yang, Y., Schubert, M. C., Reyhan, E., Tetzlaff, S. K., Wißmann, N., et al. (2022). Glioblastoma hijacks neuronal mechanisms for brain invasion. *Cell* 185, 2899–2917.e31. doi: 10.1016/j.cell.2022.06.054

Verbyla, A. (1990). A conditional derivation of residual maximum likelihood. *Australian Journal of Statistics* 32, 227–230. doi: 10.1111/j.1467-842X.1990.tb01015.x

Vojgani, E., Hölker, A., Mayer, M., Schön, C.-C., Simianer, H., and Pook, T. (2023). Genomic prediction using information across years with epistatic models and dimension reduction via haplotype blocks. *PloS one* 18, e0282288. doi: 10.1371/journal.pone.0282288

Vojgani, E., Pook, T., Martini, J., Hölker, A., Mayer, M., Schön, C.-C., et al. (2021). Accounting for epistasis improves genomic prediction of phenotypes with univariate and bivariate models across environments. *Theoretical and Applied Genetics* 134, 2913–2930. doi: 10.1007/s00122-021-03868-1

Vuori, K., Strandén, I., Lidauer, M., and Mäntysaari, E. (2006). MiX99-Effective solver for large and complex linear models. *Proceedings of the 8th World Congress on Genetics Applied to Livestock Production (WCGALP)* , 27–33

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* (Humana, New York, NY). 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. doi: 10.1016/j.ajhg.2010.11.011

Yao, Z. (2020). LDkit: A parallel computing toolkit for linkage disequilibrium analysis. *BMC Bioinformatics* 21, 461. doi: 10.1186/s12859-020-03754-5

Zhao, Y., Thorwarth, P., Jiang, Y., Philipp, N., Schulthess, A. W., Gils, M., et al. (2021). Unlocking big data doubled the accuracy in predicting the grain yield in hybrid wheat. *Science Advances* 7, eabf9106. doi: 10.1126/sciadv.abf9106

Zheng, X., Levine, D., Shen, J., Gogarten, S., Laurie, C., and Weir, B. (2012). A high-performance computing toolset for relatedness and principal component analysis of SNP data. *Bioinformatics* 28, 3326–3328. doi: 10.1093/bioinformatics/bts606

Zhou, H., Sinsheimer, J. S., Bates, D. M., Chu, B. B., German, C. A., Ji, S. S., et al. (2020). OpenMendel: A cooperative programming project for statistical genetics. *Human genetics* 139, 61–71. doi: 10.1007/s00439-019-02001-z