Groundwork for the Development of GPU Enabled Group Testing Regression Models

Paul Cubre
pcubre@clemson.edu

Groundwork for the Development of GPU Enabled Group Testing Regression Models

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Mathematical Sciences

by
Paul Cubre
May 2022

Abstract

In this dissertation, we develop novel techniques that allow for the regression analysis of data emerging from group testing processes and set the groundwork for graphic processing units (GPU) enabled implementations. Group testing primarily occurs in clinical laboratories, where it is used to quickly and cheaply diagnose patients. Typically, group testing tests a pooled specimen–several specimens combined into one sample–instead of testing individual specimens one-by-one. This method reduces costs by using fewer tests when the disease prevalence is low. Due to recent advances in diagnostic technology, group testing protocols were extended to incorporate multiplex assays, which are diagnostic tests that, unlike their predecessors, test for multiple infectious agents simultaneously. The diseases that a multiplex assay screen typically share co-infection risks. The positive correlation stemming from co-infection risks creates a more challenging modeling framework. In this work, we develop a Bayesian regression methodology that can analyze multiplex testing outcomes collected as part of any group testing protocol. The model can maintain marginal interpretability for regression parameters and, when the assay accuracies are unknown, we can simultaneously estimate regression parameters with the test's sensitivity (true positive rate) and specificity (true negative rate). Based on a carefully constructed data augmentation strategy, we derive a Markov chain Monte Carlo (MCMC) posterior sampling algorithm that can be used to complete model fitting. We demonstrate our methodology via numerical simulations and by using it to analyze chlamydia and gonorrhea, which are sexually transmitted infections, data collected as part of Iowa's public health laboratory's testing efforts. This regression framework's drawback is the computational intensity of the proposed steps in the MCMC algorithm. Due to computational costs, this algorithm does not scale well to the high-volume clinical laboratory settings where group testing is commonly employed. We need to accelerate the proposed algorithms to provide for faster modeling fitting and prediction. The devised MCMC algorithm has several independent steps (e.g., sampling the latent disease status of each patient), or matrix operations. These sections of the algorithm are well-suited to be accelerated with parallel processing. Parallel processing is a software task that was historically used on large-scale computer clusters, but GPUs perform similar computations. To solve the scaling issue with our MCMC algorithms, we explore the GPU as a remedy. We discuss the necessary GPU techniques to take the first steps toward fitting group testing regression models with GPUs. Specifically, we explore stochastic gradient and coordinate descent algorithm implementation with independent steps and matrix operations. Finally, we provide an example MCMC implementation on a GPU to demonstrate potential acceleration.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Group testing, also known as pooled testing, is a common technique used in clinical laboratories to screen for infectious diseases. In this setting, group testing proceeds to test pooled specimens, which are formed from physically combining specimens from each individual instead of testing each specimen one-by-one. Robert Dorfman [3] was the first to demonstrate the benefits of group testing which was motivated by the syphilitic screening efforts of the Second World War. In the 1940s, the U.S. Public Health Service and Selective Service drafted millions of men for the war. As part of a health screening, a blood draw was performed on each person to help identify those with syphilis, a bacterial infection. Each blood draw was tested for a syphilitic antigen, in other words, every draftee was tested to find every person with an immune response to the infection. In 1942, economists Robert Dorfman and David Rosenblatt attended an impromptu discussion on the wastefulness of performing the test millions of times to find a few syphilitic men [4]. As a result of those discussions, Robert Dorfman [3] authored the seminal paper that introduced group testing. In this work, Dorfman [3] proposed the following two-stage group testing procedure called Dorfman testing (DT). Suppose there are n specimens to be tested. Following Dorfman's procedure, a portion of each specimen is combined to form a pooled sample. In the first stage, the pooled sample is tested. If the test result is negative, then the individuals are classified as negative and we have classified n individuals with one test. If the test result is positive, we proceed to the second stage in which we retest each specimen one-by-one with the individuals diagnosed based on their individual level test result. It is easy to see that in low prevalence settings, this two-stage procedure has the potential to reduce testing costs.

Owing to the reduction in testing costs, group testing has been widely adopted across many different types of applications. The largest application is testing blood donations [5]. The American Red Cross [6], Canadian Blood Services [7], Japanese Red Cross [8], and German Red Cross [9] use a variation of DT with mini-pools of size 16, 24, 50, and 96, respectively, to screen blood donations for hepatitis B virus (HBV), hepatitis C virus (HCV) and human immunodeficiency virus type-1 (HIV-1). Group testing has been used to screen for a host of other infectious diseases such as chlamydia and gonorrhea [10], human immunodeficiency virus [11], influenza viruses [12], SARS-CoV-2 [13], West Nile virus [14], and Zika virus [15]. Group testing has also been used in a variety of other application areas, to include detecting rare mutations [16], chromatography in analytic chemistry [17], prevalence estimation for insect disease vectors [18], genomics studies [19], water quality studies [20], detecting disease-causing bacteria in sheep [21] and salmonella on chicken eggs [22], among others.

Group testing research typically focuses on addressing either the identification or estimation problem. The identification problem involves the task of identifying the disease status of the screened individuals, while the estimation problem is concerned with estimating either the disease's population characteristics; e.g., disease prevalence or regression functions [5] [23]. As part of the identification problem testing algorithms are developed to minimize the number of tests performed when classifying individuals. While DT can reduce costs, there are a wide variety of group testing algorithms that can be employed. Generally, there are two types of algorithms hierarchical and non-hierarchical. Hierarchical algorithms have the property that tests conducted in the previous stages are known before the next stage of testing is performed and each subject is tested, either in a pool or individually, at most, once per stage. DT is an example of a two-stage hierarchical procedure. In contrast, array testing, originally used in genomics research [19] (see Phatarfod and Sudbury [24]), is an example of a non-hierarchical algorithm. Two-stage array testing is non-hierarchical because individuals are assigned to a rectangular grid in the first stage and, based on this assignment, row and column pools are formed and tested. Therefore, each individual is tested in two overlapping pools, regardless of the other pool's outcome. In the next stage, individual testing resolves potential positives; for further details see Kim, Hudgens, Dreyfuss, Westreich, and Pilcher [25].

Many hierarchical and non-hierarchical algorithms have been proposed, and a primary focus in the development of these schema is to characterize their performance. Early algorithms assume perfect testing and a homogeneous population, i.e., every subject has the same infection risk. Many authors developed new algorithms that further reduced testing costs under these conditions. Sterrett [26] proposed that if a master pool tests positive, then the positive pool would be resolved by randomly selecting and retesting subjects from the pool individually until the first positive was found, at which point the remaining subjects would be repooled and tested. This process would be repeated until all individuals were classified. Sobel and Groll [27] developed a recursive splitting algorithm that successively splits positive pools into smaller subgroups until all the individual's statuses are resolved. However, most tests for infectious diseases are not perfect and these inaccuracies are accounted for by introducing testing errors. False-positive and false-negative errors occur when a test declares a diagnosed status that is different than an individual's true status. As part of evaluating a diagnostic tests' performance, these errors are described by the assay accuracies probabilities, sensitivity and specificity, known as the true positive rate and true negative rate, respectively. In the presence of imperfect testing, Kim, Hudgens, Dreyfuss, Westreich, and Pilcher [25] describe the operating characteristics of S-stage hierarchical group testing procedures as well as two and three stage array testing.

All of the aforementioned works still assume a homogeneous population, but realistically, we can to assume that individuals have different disease risks. Hwang [28] was the first to acknowledge and exploit this heterogeneity. The paper develops a dynamic programming algorithm to classify individuals with different probabilities of disease and was the first step in informative group testing. Informative group testing leverages individual patient information to optimize a group testing procedure. For example, Lewis, Lockary, and Kobic [29] describe the Idaho Bureau of Laboratories' stratified specimen pooling scheme where the reason for visiting a health care provider, such as routine pregnancy screening, exposure to Chlamydia trachomatis (CT) or Neisseria gonorrhoeae (NG), symptomatic, and STD screening, determines the use of group or individual testing. Bilder, Tebbs, and Chen [30] devise an informative group testing algorithm that modifies Sterrett's [26] strategy so that subject specific probabilities under imperfect testing conditions guides retesting. Similarly, McMahan, Tebbs, and Bilder [31] modify DT by guiding pool construction via individual level probabilities. Both of these procedures need a priori knowledge of the individual's infection probabilities, which are generally unknown. Thus, to implement informative techniques, we need an estimation of these probabilities.

Much of the estimation work in group testing focuses on developing techniques that can aggregate pool testing outcomes to estimate a disease's prevalence within a population. Thompson [18] first explored the estimation problem and derived a maximum likelihood estimator of the prevalence based on MPT outcomes and provided the asymptotic distribution. Hughes-Oliver and Swallow [32] proposed a two-stage adaptive estimation strategy to estimate the prevalence where the pool size in the second stage is set to minimize the mean square error of the prevalence estimator. Thompson [18] and Hughes-Oliver [32] estimate the prevalence based on data from master pool testing only and fail to account for imperfect testing. Sobel and Elashof [33] propose a maximum likelihood estimator that accounts for retesting. Chen and Swallow [34] evaluate the robustness of a prevalence estimator using testing and retesting errors. More recently, Bilder and Tebbs [35] develop an empirical Bayes estimator for the prevalence with credible intervals. The aforementioned works are all limited to estimating only population level prevalence.

Apart from estimating population prevalence, recent work has allowed for the incorporation of covariate information in the estimation of regression functions based on group testing data. Estimating regression models relate patients' covariates to their infection status. The first to use regression modeling in group testing was Farrington [22]. The paper proposes estimating a generalized linear model using a complementary log-log link function based on testing outcomes takes from master pools that were formed homogenously with respect to covariate information. Vansteelandt, Goetghebeur, and Verstraeten [36] extended Farrington's [22] work by allowing for imperfect testing, general link function, and any composition of covariates within the pool. Using regression analysis on group testing data is not limited to parametric models, Delaigle and Meister [37] developed a nonparametric regression model with imperfect testing for equal pool sizes while Delaingle, Hall, and Wishart [38] extend this approach to allow unequal pool sizes. However, all of the aforementioned regression works are capable of analyzing data arising from master pool testing. Xie [39] was the first to incorporate retesting information to estimate a regression model. This author developed an expectation-maximization algorithm to estimate regression parameters with imperfect testing and covariates. Zhang, Bilder, and Tebbs [40] expand on Xie's [39] work by looking at other common algorithms and demonstrate that better estimation efficiency can be obtained if resting information is included. These works assume that the assay's sensitivity and specificity are known and designed for specific group testing algorithms. McMahan, Tebbs, Hanson, and Bilder [41] create a Bayesian regression framework that allows for general link functions and can analyze data from any group testing scheme with imperfect testing. Liu, McMahan, Tebbs, Gallagher, and Bilder [42] build upon the Bayesian framework to allow for a generalized additive regression framework to alleviate model misspecification from strict linear covariate relations while Joyner, McMahan, Tebbs, and Bilder [43] incorporate random effects and variable selection.

The previously mentioned group testing literature uses an assay that screens for a single trait. Recent medical diagnostic developments introduced multiplex assays that test for multiple diseases simultaneously, which can facilitate completing a test panel comprised of a set of related tests. For example, a STD multiplex assay simultaneously screens patients for the bacteria CT and NG using nucleic acid amplification [1]. The SHL at the University of Iowa, Idaho Bureau of Laboratories [29] and Nebraska Public Health Laboratory [44] use this technique. A multiplex assay screens blood donations for three STDs: HBV, HCV, and HIV-1 [45] [46]. More considerations are needed for multiplex assays. First, since related diseases form the test panel, the test outcomes are potentially correlated as shown by Zhang, Bilder, and Tebbs [44] for CT and NG. Second, the testing accuracies may differ for each disease. Finally, group testing procedures must be reassessed when there are multiple traits.

In the context of multiplex assays, group testing algorithms are modified to account for multiple diseases. For example, the SHL uses an adaptation of Dorfman's group testing algorithm to test for CT and NG. The modification is to always perform multiplex assays at each stage and retest if any test is positive because it is simpler and more cost-effective [47]. Tebbs, McMahan, and Bilder [47] were the first to characterize the performance of group testing on multiple infections. Later, Hou, Tebbs, Bilder, and McMahan [48] generalize the work for higher-stage hierarchical algorithms. Hou, Tebbs, Wang, McMahan, and Bilder [49] propose non-hierarchical algorithms utilizing two-dimensional arrays and derive its operating characteristics. These works include imperfect testing and retesting information but early work in the estimation problem did not. Hughes-Oliver and Rosenberger [50] generalized Hughes-Oliver and Swallow's [32] earlier work for a two-stage adaptive group testing procedure designed for multiple diseases. The adaptation gives an optimal design for the estimation of the prevalence of three diseases in Ethiopian women: HIV, chlamydia, and syphilis. However, they assume perfect testing and master pool testing only. Tebbs, McMahan, and Bilder [47] expand on the work of Hughes-Oliver and Rosenberger [50] to propose an expectation-maximization algorithm to estimate disease prevalence with imperfect testing and retesting information.

Since the test outcomes from multiplex assays are not independent, estimating the regression function is difficult. Zhang, Bilder and Tebbs [44] generalize the work of Xie [39] and Vansteelandt et al. [51] by using a expectation-solution algorithm to estimate a regression model for master pool testing data for multiple diseases. Later, Lin, Wang, and Zheng [52] propose a generalized expectation-maximization algorithm for the regression analysis of data arising from a two-stage group testing protocol using a multiplex assay. Estimating regression functions have two primary benefits. First, as part of surveillance studies, estimating the regression function allows us to identify risk factors that relate to infection status. Thus, we can identify parts of the population that are at higher risk for disease. Second, by leveraging estimated regression functions to differentiate individuals based on risk, informative group testing techniques can be employed to reduce the cost of testing. For multiplex assays, Bilder, Tebbs, and McMahan [53] show substantial cost savings by designing an optimal hierarchical S-stage procedure with individual probabilities of infection. Bilder, Tebbs, and McMahan [54] propose the first non-hierarchical algorithm for informative multiplex array group testing. Previous multiplex assay regression works by Zhang, Bilder and Tebbs [44] and Lin, Wang, and Zheng [52] are limited to group testing algorithms for master pool testing and two-stage group testing, respectively.

In the following chapter, we develop a Bayesian regression framework to simultaneously estimate covariate effects, correlation structure, and assay accuracies using data from any group testing scheme arising from a discriminating multiplex assay. Our work is an extension of the Bayesian regression framework of McMahan, Tebbs, Hanson, and Bilder [41] to that allows for the joint analysis of multiple diseases. We develop implementations of our approach under both the multivariate probit and logistic models following the data augmentation strategies by Albert and Chib [55] and O'Brien and Dunson [56], respectively. These implementations maintain their marginal interpretability this allows inference of marginal effects rather than conditional effects. Moreover, we develop easy to implement Markov chain Monte Carlo (MCMC) algorithm that can estimate the proposed model. A benefit of using a Bayesian methodology is we can incorporate expert knowledge to inform model fitting such as the assay's clinical data for sensitivity and specificity estimations. We explore the finite sample properties of our proposed approach through simulation studies and by applying it to STD data collected by the SHL in Iowa. A drawback of our Bayesian implementation is the amount of time required to compute enough samples for large data sets like the ones that naturally arise from the high volume settings where group testing is utilized. For example, in our application we have 13,862 female test results which are easily analyzed by our algorithm. Although, when considering the test results for the United States our implementation would be unteable. National test results would be at least an order of magnitude larger than the national infection rates for CT and NG, which where the most reported notifiable disease in 2019 with more than 1.8 million and 616,392 infections, respectively, according to a 2021 report [57]. With a low prevalence rate the number of tests needed for analysis can be an order of magnitude larger. Our MCMC methodology does not scale efficiently to these infection rates.

To overcome the computational barriers limiting the scaling of our Bayesian implementation a common approach is to employ parallel processing. Parallel processing involves computing independent steps simultaneously rather than sequentially for time savings. Historically, this was accomplished on multiple CPUs in mainframes and supercomputers. More recently graphics processing unit (GPU) have been used for parallelized tasks. GPUs were not always for the task of general-purpose computing. In 2001, Larsen and McAllister demonstrate matrix-matrix multiplications with 8-bit fix point precision [58] opening the door for GPUs in general-purpose computing. There are several early applications such as finite element simulations [59], physics-based simulations [60], visually simulating ice crystal growth [61], and accelerating neural networks [62]. An important milestone occurred in 2003, GPUs switched from 8-bit integer color to 32-bit floating point that allows for a large range of matrix operations [63]. However, these operations were slower than their CPU counterparts. In 2005, Galoppo, Govindaraju, Henson, and Manocha demonstrated LU decomposition was faster on a GPU than a CPU [64]. By 2006, these advancements allowed people to contribute their GPUs for Folding@home, one of the largest distributed computing projects that simulates protein folding [65]. All the early applications for GPUs had their drawbacks. Many developers found designing and writing programs for GPUs difficult with numerous limitations and workarounds using functions designed for video game development. The leading GPU designer, NVIDIA, released their CUDA toolkit to address the need for an accessible framework [66]. The toolkit is used by Terenin, Dong, and Draper [67] to demonstrate the effectiveness and speedups of using a GPU for MCMC methods.

To lay the foundations for adapting a group testing MCMC algorithm for GPU acceleration, we introduce GPU computing and run several test cases to demonstrate its effectiveness. We provide a basic introduction to programming GPUs with NVIDIA CUDA, a common development library. Following our introduction to GPUs, we implement two optimization routines using the hardware: stochastic gradient descent and stochastic coordinate descent. These two methods are commonly used in machine learning and statistics to minimize an objective function. They are particularly useful in large data problems as they effectively solve the problem in small steps. The methods are stochastic because of the randomization of taking a sample from the available data or a parameter to update. The difference between stochastic gradient descent and coordinate descent is how the parameters are updated. Stochastic gradient descent updates all of the parameters simultaneously based on a gradient vector computed using a randomly selected subset of the data. In contrast, stochastic coordinate descent updates a randomly selected parameter based on the entire data set. In our exploration, we gain techniques to parallelize independent processes and matrix operations. We apply those lessons to accelerate a MCMC algorithm. We demonstrate the algorithm for the Horseshoe probit regression model also shown by Terenin, Dong, and Draper [67].

This dissertation is organized as follows: In Chapter 2, we discuss our Bayesian regression model, show its finite sample properties by several simulations and demonstrate its application with Iowa's CT and NG data. In Chapter 3, we give an introduction meant for a beginner with no prior knowledge to familiarize with the terminology and set up of a simple GPU program. In Chapter 4, we implement two descent algorithms on a CPU and GPU for comparison in complexity. In Chapter 5, we recreate Terenin, Dong, and Draper's work [67] to show how randomization, sampling, and MCMC methods are implemented on a GPU. Finally, in Chapter 6, we discuss future work.

Chapter 2

Bayesian Regression for Group Testing

Group testing is the process of combining specimens (e.g., urine, blood, etc.) from different individuals into a pool, and testing the pool for disease or infection. Group testing was introduced by Dorfman [3] to screen draftees for syphilis in the Second World War as a cost-savings measure to reduce the number of tests performed. This type of testing is guided by testing schemes or protocols that determine how testing proceeds algorithmically. Typically, the pooled specimen is tested first. If the result of the pooled specimen is negative, then all individuals in the pool are declared negative and only one test is used to classify all the members. If the pooled specimen test is positive, then the members of the pool are retested in subset pools to eventually determine the positive individuals. Since Dorfman introduced group testing, it has become a popular method to screen large populations for diseases. The main advantage of group testing is the cost savings, especially in high-volume settings. Group testing has proven effective in screening for sexually transmitted diseases including chlamydia and gonorrhea [47], and screening blood donations for infectious diseases [5] by blood banks (e.g., the German Red Cross [9], Japanese Red Cross [8], Canadian Blood Services [7] and the American Red Cross [6]) for hepatitis B virus (HBV), hepatitis C virus (HCV) and human immunodeficiency virus type-1 (HIV-1).

While group testing can result in cost savings for classifying every individual, imperfect testing and dependencies between pools make group testing more complicated for estimation. Imperfect testing or testing errors mean that the true disease status of an individual is never known; only the diagnosed status of the individual at the time of testing is known. Moreover, in most group testing protocols, many individuals are in multiple, possibly overlapping, pools. Pools sharing common individuals no longer are independent. With imperfect testing, we cannot sidestep the dependencies by assigning the true disease statuses to every individual and we cannot use regression models that need independent outcomes. Instead, we have the diagnosed status of individuals through a set of dependent tests. These issues complicate data analysis for estimation. For example, the most basic group testing protocol is master pool testing. In master pool testing, all subjects are divided into separate pools which are subsequently tested, with no testing being performed beyond this first stage. If the pool is positive, then at least one member may be infected as imperfect testing means a positive outcome could arise from a pool with no individual who has the disease. Moreover, the number of likely positive individuals in the pool is unknown. With all these drawbacks, master pool testing can be used to estimate population prevalence or a regression function. Farrington [22] was the first to propose estimating the prevalence with a generalized linear model using a complementary log-log link function. His work was restricted to perfect testing on master pools with the additional limitation that the individual's covariates were identical for each master pool. It is not reasonable in a low volume clinical setting to have groups of patients with the same covariates. Moreover, most all tests are imperfect. Tests have a less than perfect true positive rate (sensitivity) and true negative rate (specificity) known as the assay's accuracy probabilities. Motivated by HIV surveillance data, work by Vansteelandt, Goetghebeur, and Verstraeten [36] allow for estimation with a general link function and any set of covariates for each pool. They included imperfect testing and reflecting the realities of HIV's three layers of tests. However, they continued to only consider master pool test results. For parametric extensions, Chen, Tebbs, and Bilder [68] add random effects, and research by McMahan, Tebbs, and Bilder [69] account for biomarker distributions. For non-parametric regression, Delaigle and Meister [37] introduce testing errors and any individual covariates but use only equal-sized master pooled specimen test results. Later, Delaingle, Hall, and Wishart [38] extended this non-parametric approach to allow for unequal pool sizes. The primary goal of group testing algorithms is to diagnosis individuals; however, all of the previously mentioned regression methods are designed to analyze data arising from master pool testing. In contrast, most group testing protocols resolve postive master pools in an effort to diagnose each individual. For example, see Kim, Hudgens, Dreyfuss, Westreich, and Pilcher [25], Kim and Hudgens [70], Sterrett [26], Sobel and Groll [27], Sobel and Elashof [33], Hughes-Oliver and Swallow [32], Gastwirth and Johnson [71], Bilder, Tebbs, and Chen [30], and McMahan, Tebbs, and Bilder [31]. These algorithms guide retesting subsets of individuals after a positive pool result to find the positive individuals or for quality control. Xie [39] was the first to incorporate retesting information for estimation via an expectation-maximization algorithm that estimates an individual's probability of disease by viewing their true disease status as a latent variable. The algorithm accounted for different testing errors for the pool test and confirmation test. Zhang, Bilder, and Tebbs [40] provide further analysis of the methodology. However, like most all other regression methods in group testing, it assumes the assay errors were known. McMahan, Tebbs, Hanson, and Bilder [41] take a Bayesian approach that allows for unknown testing error and any group testing protocol. Recent statistical research by Liu, McMahan, Tebbs, Gallagher, and Bilder [42] extends the Bayesian approach to allow for generalized additive models. All of the previously mentioned works are designed to analyze data arising from a group testing protocol that uses a diagnostic test that screens for a single disease.

Recently, advances in medical diagnostic equipment have given rise to multiplex assays that test for multiple diseases simultaneously. Equipment manufacturers brought the Aptima Combo 2 Assay for chlamydia and gonorrhea in 2001 and the cobas TaqScreen MPX Test [45] for HIV, HBV, and HCV in 2009. The Aptima Combo 2 is in use at the State Hygienic Laboratory (SHL) at the University of Iowa [41], the Idaho Bureau of Laboratories [29] and the Nebraska Public Health Laboratory (NPHL) [44]. The cobas TaqScreen MPTX Test is used to screen blood donations [9]. Group testing algorithms are used for the multiplex assays at SHL according to Tebbs, McMahan, and Bilder [47]. Multiplex assays generate even more complex data structures due to the correlation of the outcomes. For example, risks for one STD can be risks for another STD. For descriptions and properties of multiplex assay group testing schemes see Hughes-Oliver and Rosenberger [50], Tebbs, McMahan, and Bilder [47], Hou, Tebbs, Bilder, and McMahan [48], Hou, Tebbs, Wang, McMahan, and Bilder [49] and Bilder, Tebbs, and McMahan [54]. Warasi, Tebbs, McMahan, and Bilder [72] develop a Bayesian framework to estimate the prevalence of multiple diseases with unknown assay probabilities from multiplex assays. However, every patient has the same probability of infection per disease. Building off of the methods of Xie [39], recent works incorporate data from multiplex assays in the regression models of Zhang, Bilder, and Tebbs [44] and Lin, Wang, and Zheng [52]. The work of Zhang, Bilder, and Tebbs [44] develops an expectation-solution algorithm for multiple diseases but they are restricted to master pool testing. The work of Lin, Wang, and Zheng [52] develop a generalized expectation-maximization for a copula-based multivariate binary regression that analyses data limited to a two-stage hierarchical testing scheme.

To provide a general regression framework, we develop a Bayesian regression model that can be used to analyze group testing data arising from any group testing protocol that utilizes a multiplex assay. Our work generalizes the single disease work of McMahan, Tebbs, Hanson, and Bilder [41] to analyze data from multiplex assays. Our Bayesian regression framework allows us to analyze data from any group testing protocol that uses a discriminating multiplex assay. We can estimate regression models that describe the risk of having all the diseases simultaneously while specifically accounting for the correlation structure between diseases. The regression functions that we estimate, maintain their marginal interpretability. When the assay accuracies are unknown, we are able to estimate them simultaneously with the regression coefficients. With deliberate choices for data augmentation stategies, we implement easy to compute Markov chain Monte Carlo (MCMC) sampling algorithms that allow for model fitting. Because we use Bayesian methodologies, we can use expert knowledge to guide model fitting. We provide two implementations of the model that utilizes the data augmentation strategies for multivariate probit regression as described by Albert and Chib [55] and multivariate logistic regression by O'Brien and Dunson [56]. We explore the finite sample properties of our proposed estimation framework through simulation studies and further illustrate its performance by applying it to chlamydia and gonorrhea test data collected by the SHL in Iowa.

The remaining sections of this chapter are as follows: Section 2.1 provides the preliminary information for the regression model and the modeling assumptions. In Section 2.2, we outline the prior model specification and the development of the full conditional distributions. Section 2.3 provides the results of a simulation study that demonstrates the effectiveness of our approach. Section 2.4 presents an analysis of chlamydia and gonorrhea testing data collected in Iowa and concludes with a summary discussion. Additional tables, figures and results are in Appendix B.

2.1 Model Development

Suppose that N individuals are screened for D infectious diseases by a group testing protocol. Let $\widetilde{\mathbf{Y}}_i = (\widetilde{Y}_{i1}, \ldots, \widetilde{Y}_{iD})'$ be the vector of true disease statuses of the ith patient such that $\widetilde{Y}_{id} = 1$, if the ith patient's true status is positive for the dth disease and $\widetilde{Y}_{id} = 0$ otherwise. We assume the availability of a collection of risk factors and demographic information that we denote as $\mathbf{x}_{id} = (1, x_{id1}, x_{id2}, \ldots, x_{idQ_d})'$ for the ith patient and dth disease. Let $\mathbf{X}_i = \oplus_{d=1}^{D} \mathbf{x}'_{id}$ be the block diagonal matrix for the ith patient with $\mathbf{x}'_{id}$ for the dth block. We relate the predictor variables $\mathbf{X}_i$ to the response variable $\widetilde{\mathbf{Y}}_i$ through the following general linear model, $P(\widetilde{\mathbf{Y}}_i | \mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) = H(\mathbf{X}'_i \boldsymbol{\beta}, \mathbf{R})$ where H is a known inverse link function, $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_D)'$ be the vector of regression coefficients and $\mathbf{R}$ is a correlation term. We restrict $\mathbf{R}$ to be a symmetric matrix where ρ_{rs} is the r, s entry and 1 is along the diagonal. We later examine the multivariate probit and multivariate logistic link functions and for convenience, we denote the linear predictors as $\boldsymbol{\mu}_i = (\mu_{i1}, \ldots, \mu_{iD})' = (\mathbf{x}'_{i1}\boldsymbol{\beta}_1, \ldots, \mathbf{x}'_{iD}\boldsymbol{\beta}_D)' = \mathbf{X}_i \boldsymbol{\beta}$. We also assume an individual's infection status is conditionally independent across individuals given the covariates, i.e. $\widetilde{\mathbf{Y}}_i | \mathbf{X}_i \perp \widetilde{\mathbf{Y}}_j | \mathbf{X}_j$ for $j \neq i$. If the true disease statuses $\widetilde{\mathbf{Y}}_i$ where known then we could easily estimate the model parameters. However, the true disease statuses are unobservable in most group testing settings.

We are proposing a general framework that can analyze test data arising from any group testing algorithm that uses a discriminating assay. There are many different types of group testing protocols such as Dorfman testing [3], S-stage hierarchical testing, [25], array testing [24], 3-stage array testing [25], three dimensional array testing [70], quality control retesting [71] [16], Sterrett's [26] strategy, Sobel and Groll's [27] recursive splitting and many others [30] [31]. Many of these protocols require individuals to be tested in multiple and possibly overlapping pools, individually or as part of confirmatory retests. In order to handle all these algorithms and more, we introduce the index set $\mathcal{P}_j$ to track pool membership. This index set identifies the individuals who contributed to the jth pool for $j = 1, \ldots, M$ where M is the total number of tests performed. The true disease status of the pool is determined by the individuals assigned to it. In particular, if any individual is positive for the dth disease, then the pool is positive for the dth disease. We let $\widetilde{\mathbf{Z}}_j = (\widetilde{Z}_{j1}, \ldots, \widetilde{Z}_{jD})'$ be the vector of true pool disease statuses for the jth pool where $\widetilde{Z}_{jd} = 1$ if the jth pool has any member with disease d and $\widetilde{Z}_{jd} = 0$ otherwise. Thus, we have $\widetilde{Z}_{jd} = \mathbb{1}(\sum_{i \in \mathcal{P}_j} \widetilde{Y}_{id} > 0)$ where $\mathbb{1}(\cdot)$ is the usual indicator function. In the trivial case with $|\mathcal{P}_j| = 1$ and $\mathcal{P}_j = \{i\}$, the pool has a single individual. Much like $\widetilde{\mathbf{Y}}_i$, $\widetilde{\mathbf{Z}}_j$ are unobservable as a result of imperfect testing.

With a true pool status and pool test results, we can relate the two by testing errors. That is, we allow for imperfect testing which results in false-positive errors and false-negative errors. We denote the testing outcomes taken on the jth pool as $\mathbf{Z}_j = (Z_{j1}, \ldots, Z_{jD})'$ where $Z_{jd} = 1$ if the jth pool tests positive for disease d and $Z_{jd} = 0$ otherwise. Let the true positive rate or sensitivity for the

jth pool be defined as $S_{e_j:d} = P(Z_{jd} = 1|\widetilde{Z}_{jd} = 1)$ and the true negative rate or specificity be defined as $S_{p_j:d} = P(Z_{jd} = 0|\widetilde{Z}_{jd} = 0)$. In order to develop our model, we assume independence of testing between diseases conditionally on the true disease status, that is $\mathbf{Z}_j|\widetilde{\mathbf{Y}}_1, \ldots, \widetilde{\mathbf{Y}}_N \perp \mathbf{Z}_k|\widetilde{\mathbf{Y}}_1, \ldots, \widetilde{\mathbf{Y}}_N$ for all $j \neq k$. This is a common assumption in group testing [73]. We also assume that the conditional distribution of $\mathbf{Z}_j|\widetilde{\mathbf{Z}}_j$ does not depend on the covariates. Now we have the links between the testing outcomes and the covariates. Furthermore, it is natural to sometimes believe sensitivities are specificities are known and other times they are not and need to be estimated.

The sensitivity and specificity for each disease are provided by the manufacturer of the test that we use as a fixed value or to inform priors. For example, the manufacturer of the Aptima Combo 2 assay for CT and NG used by the SHL provides the assay's sensitivity and specificity for each disease, specimen type (swab or urine), and gender. In short, the assay's performance can vary not just between diseases but also based on the testing configuration. Additionally, the pool size may affect the assay's performance in the group testing context. Most tests rely on detecting biomarkers such as an antigen, snippets of genetic material, or chemical compounds. Typically, assays give a diagnosis based on elevated levels of a biomarker. If the biomarker exceeds a prespecified diagnostic threshold, then we classify the specimen to be positive and otherwise we classify the specimen to be negative. For example, after being exposed to a virus, a patient may test negative for a few days until the virus replicates to a detectable concentration. When specimens are pooled together, if a pool contains a positive specimen, then its signal could be diluted by the negative specimens, thus limiting the assays ability to correctly classify the pool. To prevent this, the diagnostic threshold can in some instances be recalibrated [74]. This recalibration step leads to the assumption by Kim et. al. [25] that the sensitivity and specificity are the same for all pool sizes. However, in our regression methodology we want the flexibility to acknowledge differences in the assay sensitivity and specificity that might exist as a function of the pool size, disease, specimen type, gender, etc. For practical purposes, we add the restriction that each pool is exactly one specimen type e.g. urine or swab.

We develop the notation for sensitivity and specificity to accommodate multiple test configurations. Let $S_{e(l):d}$ and $S_{p(l):d}$ denote the sensitivity and specificity associated with the lth assay test configuration for disease d for $l = 1, \ldots, L$. Denote $\mathbf{S}_e = (S_{e(1):1}, \ldots, S_{e(L):D})'$ and $\mathbf{S}_p = (S_{p(1):1}, \ldots, S_{p(L):D})'$. Let $\mathcal{M}(l) = \{j$: the lth test configuration was used to test the jth pool$\}$ that is $\mathcal{M}(l)$ tracks which tests use the lth test configuration. That is $S_{e_j:d} = S_{e'_j:d} = S_{e(l):d}$ and $S_{p_j:d} = S_{p'_j:d} = S_{p(l):d}$ when $j, j' \in \mathcal{M}(l)$. For generality, we consider three specific ways of handling the inclusion of testing accuracies. First, we consider the setting where the sensitivities and specificities are known. In our example Iowa data set we would have eight sets of known sensitivities and specificities provided by the manufacturer corresponding to the different combinations of disease, specimen and gender but do not vary by pool size. Second, we consider sensitivities and specificities configuration exactly the same as in the previous setting except they are unknown and have to be estimated. Third, it would be reasonable to assume the performance of the test is going to be the same when testing pools of a common size. That is sensitivity and specificity would be the same within pool size but vary across pool sizes. The SHL uses Dorfman testing with masterpools of size four. Therefore in our example we estimate sixteen sets of sensitivities and sixteen specificities for the previous combinations of diseases, specimen, and gender plus the additional two pool sizes. We have the additional consideration that a partial ordering of sensitivities and specificities exist for the group size within test configurations for disease, specimen, and gender to account for the dilution effects. Suppose pools j and k have different sizes with $|\mathcal{P}_j| < |\mathcal{P}_k|$. If we assume there is a partial order, then we assume that the larger sample has smaller sensitivity and larger specificity. Thus we have $S_{e_j:d} > S_{e_k:d}$ and $S_{p_j:d} < S_{p_k:d}$ for all d. We consider one test being used and its performance differs based on its test configuration. Generalizing to multiple assays is trivial within the notation and framework we develop but for purposes of brevity we omit it.

Under the aforementioned assumptions, we may write the conditional distribution of the observed data as

$$P(\mathbf{Z}|\mathbf{X},\boldsymbol{\beta},\mathbf{R}) = \sum_{\widetilde{\mathbf{Y}}\in\boldsymbol{\mathcal{Y}}} P(\mathbf{Z}|\widetilde{\mathbf{Y}})P(\widetilde{\mathbf{Y}}|\mathbf{X},\boldsymbol{\beta},\mathbf{R}). \tag{2.1}$$

where $\mathbf{Z} = (\mathbf{Z}_1,\ldots,\mathbf{Z}_M)'$, $\mathbf{X} = (\mathbf{X}_1,\ldots,\mathbf{X}_n)'$, $\widetilde{\mathbf{Y}} = (\widetilde{\mathbf{Y}}_1,\ldots,\widetilde{\mathbf{Y}}_N)$ and $\boldsymbol{\mathcal{Y}}$ is all the possible true disease statuses of $\widetilde{\mathbf{Y}}$. In the expression above, we have that

$$\begin{aligned} P(\mathbf{Z}=\mathbf{z}|\widetilde{\mathbf{Y}}=\widetilde{\mathbf{y}}) &= \prod_{j=1}^{M}\prod_{d=1}^{D} P(Z_{jd}=z_{jd}|\widetilde{Z}_{jd}=\widetilde{z}_{jd}) \\ &= \prod_{j=1}^{M}\prod_{d=1}^{D} \left\{S_{e_j:d}^{z_{jd}}\left(1-S_{e_j:d}\right)^{1-z_{jd}}\right\}^{\widetilde{z}_{jd}} \left\{S_{p_j:d}^{1-z_{jd}}\left(1-S_{p_j:d}\right)^{z_{jd}}\right\}^{1-\widetilde{z}_{jd}} \\ &= \prod_{l=1}^{L}\prod_{j\in\mathcal{M}(l)}\prod_{d=1}^{D} \left\{S_{e(l):d}^{z_{jd}}\left(1-S_{e(l):d}\right)^{1-z_{jd}}\right\}^{\widetilde{z}_{jd}} \left\{S_{p(l):d}^{1-z_{jd}}\left(1-S_{p(l):d}\right)^{z_{jd}}\right\}^{1-\widetilde{z}_{jd}}, \end{aligned} \tag{2.2}$$

where $\widetilde{z}_{jd} = I(\sum_{i\in\mathcal{P}_j} \widetilde{y}_{id} > 0)$. Equation 2.2 gives the probability of group testing outcomes given the latent true disease status of the pools and the testing errors for the L assays. Additionally in the expression above, we have:

$$P(\widetilde{\mathbf{Y}}|\mathbf{X},\boldsymbol{\beta},\mathbf{R}) = \prod_{i=1}^{N} P(\widetilde{\mathbf{Y}}_i|\mathbf{X}_i,\boldsymbol{\beta},\mathbf{R}) = \prod_{i=1}^{N} H(\mathbf{X}_i'\boldsymbol{\beta},\mathbf{R}) \tag{2.3}$$

where H is a known inverse link function which is linked to the specification of our multivariate generalized linear model. In what follows, we consider specifications under the multivariate logit and probit link functions. It is important to note, in the development the model described in equation 2.1 we make three primary assumptions. First, we assume an individual's infection status is conditionally independent across individuals given the covariates, i.e. $\widetilde{\mathbf{Y}}_i|\mathbf{X}_i \perp \widetilde{\mathbf{Y}}_j|\mathbf{X}_j$ for $j \neq i$. Second, we assume independence of testing between diseases conditionally on the true disease status, that is $\mathbf{Z}_j|\widetilde{\mathbf{Y}}_1,\ldots,\widetilde{\mathbf{Y}}_N \perp \mathbf{Z}_k|\widetilde{\mathbf{Y}}_1,\ldots,\widetilde{\mathbf{Y}}_N$ for all $j \neq k$. Lastly, we assume that the conditional distribution of $\mathbf{Z}_j|\widetilde{\mathbf{Z}}_j$ does not depend on the covariates.

2.1.1 Model Fitting Strategies

It is important to note that to evaluate 2.1 it requires us to compute 2^{ND} possible probabilities numerous times. It would be infeasible to compute every term, multiple times for a high volume setting with N large. Instead, we consider a two stage data augmentation process to circumvent direct numerical evaluation of 2.1 to facilitate the development of an posterior sampling algorithm. To reduce the complexity of the multiple fitting strategies that we outline, the posterior sampling techniques for the regression coefficients, and the correlation matrix are developed under the assumption that the testing errors are known and fixed. This assumption is relaxed in Section 2.2.3. The first of these data augmentation steps introduces the individual's true disease statuses as latent random variables. This leads to the following joint distribution:

$$P(\mathbf{Z},\widetilde{\mathbf{Y}}|\mathbf{X},\boldsymbol{\beta},\mathbf{R}) = P(\mathbf{Z}|\widetilde{\mathbf{Y}})P(\widetilde{\mathbf{Y}}|\mathbf{X},\boldsymbol{\beta},\mathbf{R}). \tag{2.4}$$

The second step introduces a series of latent variables that allows for the decomposition of the link function. We augment the likelihood so that the vector of binary indicator, $\widetilde{\mathbf{Y}}_i$, is derived from a vector of continuous variables, $\mathbf{T}_i$. Suppose we have a vector of latent variables $\mathbf{T}_i = (T_{i1},\ldots,T_{iD})'$

with $\widetilde{Y}_{id} = \mathbb{1}(T_{id} > 0)$ and $\mathbf{T}_i$ has distribution $f_{\mathbf{T}_i}(\mathbf{t}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R})$. This gives the following conditional distribution:

$$\begin{aligned} P(\mathbf{Z}, \widetilde{\mathbf{Y}}|\mathbf{X}, \boldsymbol{\beta}, \mathbf{R}) = &\prod_{l=1}^{L} \prod_{j \in \mathcal{M}(l)} \prod_{d=1}^{D} \left\{ S_{e(l):d}^{z_{jd}} \left(1 - S_{e(l):d}\right)^{1-z_{jd}} \right\}^{\widetilde{z}_{jd}} \left\{ S_{p(l):d}^{1-z_{jd}} \left(1 - S_{p(l):d}\right)^{z_{jd}} \right\}^{1-\widetilde{z}_{jd}} \\ &\times \left\{ \prod_{i=1}^{N} \int f_{\mathbf{T}_i}(\mathbf{t}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) \prod_{d=1}^{D} \mathbb{1}(t_{id} > 0)^{\widetilde{y}_{id}} \mathbb{1}(t_{id} \leq 0)^{1-\widetilde{y}_{id}} d\mathbf{t}_i \right\}. \end{aligned} \tag{2.5}$$

In what follows, we consider two specifications of $f_{\mathbf{T}_i}(\mathbf{t}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R})$ that are two of the most common link functions: probit and logistic. To decompose the probit link function, we follow Albert and Chib [55]. We use the strategy of Dunson and O'Brien [56] to decompose the logistic link function.

Under the probit model, $f_{\mathbf{T}_i}(\mathbf{t}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) = N_D(\mathbf{t}_i|\boldsymbol{\mu}_i, \mathbf{R})$ where $\boldsymbol{\mu}_i = \mathbf{X}_i'\boldsymbol{\beta}$. An important feature of this model is it retains the marginal intepretability of the regression coefficients. In the following, we demonstrate the marginal distribution is the univariate probit model. We define the set A_{id} as $(0, \infty)$ when $\widetilde{y}_{id} = 1$ and $(-\infty, 0]$ when $\widetilde{y}_{id} = 0$. Without loss of generality, we show this for the first true disease status.

$$\begin{aligned} P(\widetilde{Y}_{i1} = \widetilde{y}_{i1}|\mu_{i1}) &= \sum_{\widetilde{y}_{id}:d \neq 1} P(\widetilde{Y}_{i1}, \ldots, \widetilde{Y}_{iD}|\boldsymbol{\mu}_i, \mathbf{R}) \\ &= \sum_{\widetilde{y}_{ij}:j \neq d} \int_{A_{i1}} \int_{A_{iD}} N_{\mathbf{T}_i}(\mathbf{t}_i|\boldsymbol{\mu}_i, \mathbf{R}) d\mathbf{t}_i \\ &= \int_{A_{i1}} \int_{\mathbb{R}} \int_{\mathbb{R}} N_{T_{i1}}(t_{i1}|\mu_{i1}, 1) N_{\mathbf{T}_{i(-1)}|T_{i1}=t_{i1}}(\mathbf{t}_{i(-1)}|t_{i1}, \boldsymbol{\mu}_i, \mathbf{R}) d\mathbf{t}_i \\ &= \int_{A_{i1}} N_{t_{i1}}(t_{i1}|\mu_{i1}, 1) dt_{i1} \left\{ \int_{\mathbb{R}} \int_{\mathbb{R}} N_{\mathbf{T}_{i(-1)}|t_{i1}}(\mathbf{t}_{i(-1)}|t_{i1}, \boldsymbol{\mu}_{i(-1)}, \mathbf{R}) d\mathbf{t}_{i(-1)} \right\} \\ &= \Phi(\mathbf{x}_i'\boldsymbol{\beta}_1)^{\widetilde{y}_{i1}} \left\{1 - \Phi(\mathbf{x}_i'\boldsymbol{\beta}_1)\right\}^{1-\widetilde{y}_{i1}}. \end{aligned} \tag{2.6}$$

This leads to the conclusion that the regression coefficients are interpreted the same as in univariate probit model.

For our second implementation, we use the augmentation strategy for the multivariate logistic density function outlined by O'Brien and Dunson [56]. O'Brien and Dunson's [56] gives the distribution as the following:

$$f_{\mathbf{T}_i}(\mathbf{t}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) = \mathcal{T}_{D,\nu} \left\{ g_\nu(t_{i1} - \mu_{i1}), \ldots, g_\nu(t_{iD} - \mu_{iD}); \mathbf{0}, \mathbf{R} \right\}$$

$$\times \prod_{d=1}^{D} \mathcal{L}(t_{id}; \mu_{id}) / \mathcal{T}_{1,\nu} \left\{ g_\nu (t_{id} - \mu_{id}); 0, 1 \right\} \tag{2.7}$$

where $g_\nu(\cdot) = F_\nu^{-1}\{F(\cdot)\}$ using the notation F_ν to represent the c.d.f. of the student-t distribution, F to represent the c.d.f. of the logistic distribution, $\mathcal{L}$ represents the p.d.f. of the logistic distribution and $\mathcal{T}_{D,\nu}(\boldsymbol{\mu}_i, \mathbf{R})$ be the multivariate t distribution with ν degrees of freedom, mean $\boldsymbol{\mu}_i = \mathbf{X}_i'\boldsymbol{\beta}$ and correlation matrix $\mathbf{R}$. The distribution's derivation is not provided here but is shown in O'Brien and Dunson [56]. Briefly, 2.7 is derived by transforming the logistic random variable T_d into a uniform density, then connects both characteristics by the copula method using the multivariate t distribution. It can be shown that this distribution also provides the desired properties: the marginal distribution is the univariate logistic density and the correlation matrix has ones along the diagonal by construction.

When we introduce the t_i as missing data our data model is the following:

$$\begin{aligned} P(\mathbf{Z}, \widetilde{\mathbf{Y}}, \mathbf{T} | \mathbf{X}, \boldsymbol{\beta}, \mathbf{R}) = &\prod_{l=1}^{L} \prod_{j \in \mathcal{M}(l)} \prod_{d=1}^{D} \left\{ S_{e(l):d}^{z_{jd}} \left(1 - S_{e(l):d}\right)^{1-z_{jd}} \right\}^{\widetilde{z}_{jd}} \left\{ S_{p(l):d}^{1-z_{jd}} \left(1 - S_{p(l):d}\right)^{z_{jd}} \right\}^{1-\widetilde{z}_{jd}} \\ &\times \left\{ \prod_{i=1}^{N} f_{\mathbf{T}_i}(\mathbf{t}_i | \mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) \prod_{d=1}^{D} \mathbb{1}(t_{id} > 0)^{\widetilde{y}_{id}} \mathbb{1}(t_{id} \le 0)^{1-\widetilde{y}_{id}} \right\}. \end{aligned} \tag{2.8}$$

The decomposition of the probit link function into the multivariate normal distribution leads to a normal full conditional for the regression coefficients under a normal prior and leads to straightforward posterior sampling. This would also be desirable under the multivariate logistic model. O'Brien and Dunson [56] demonstrate that this is achieved using importance sampling to approximate the logistic distribution with the multivariate t distribution which can be decomposed into a mixture of a gamma distribution and multivariate normal distribution which is shown in Section 2.2.

The choice of priors has the potential to significantly impact the model in the Bayesian framework. For the regression part, our choice of priors follows from similar choices by O'Brien and Dunson [56]. These are made with computational ease in mind. We choose a conditional conjugate prior for $\boldsymbol{\beta}$ to be normal with mean 0 and covariance structure of $\boldsymbol{\Sigma}_0$. It is important to note that $\boldsymbol{\Sigma}_0$ could be chosen to assert minimal influence on the posterior analysis by choosing it such that the prior is diffuse. This can be achieved by allowing the variances to be large. Additionally, we select a uniform prior for ρ_{rs} over the support $[-1, 1]$.

2.2 Posterior Sampling

We describe the posterior sampling for $\boldsymbol{\beta}$, and $\mathbf{R}$ when the assay accuracy probabilities $S_{e(l):d}$ and $S_{p(l):d}$ are known and fixed. We develop a MCMC posterior sampling algorithm that consists of Gibbs steps and Metropolis-Hasting steps for each correlation term. To outline the development of the posterior sampling algorithm, we compute full conditional distributions for $\widetilde{\mathbf{Y}}_i$, $\mathbf{T}_i$, $\boldsymbol{\beta}$, and $\mathbf{R}$. Since T_{id} and $\widetilde{Y}_{id}$ have a straightforward relationship, it becomes hard to sample one without the other. For both our implementations, we are able to sample $\widetilde{Y}_{id}$ without T_{id} by integrating T_{id} out. Note that we are sampling $\widetilde{Y}_{id}$ and T_{id} and not their vector forms. We found it easier to sample each true disease status per individual rather than simultaneously sampling all true diseases statuses simultaneously per individual using a slice-sampler. This decomposition allows us to draw from a Bernoulli distribution and a truncated normal distribution for $\widetilde{Y}_{id}$ and T_{id} respectively. For $\mathbf{R}$, we updated each entry by a random walk Metropolis-Hastings step. Simultaneously updating the matrix faces difficulties arising from the restriction on the diagonal of the correlation matrix. In what follows, we outline the probit model in Section 2.2.1 and the logistic model in Section 2.2.2. We incorporate the testing errors in Section 2.2.3.

2.2.1 Probit Model

The full hierarchy of the proposed probit model is

$$\begin{aligned}
\widetilde{Y}_{id} &= \mathbb{1}(T_{id} > 0) && i = 1, \ldots, N, d = 1, \ldots, D, \\
\mathbf{t}_i &\sim N_D(\mathbf{X}_i\boldsymbol{\beta}, \mathbf{R}) && i = 1, \ldots, N, \\
\boldsymbol{\beta} &\sim N_Q(0, \boldsymbol{\Sigma_0}), \\
\rho_{rs} &\sim Uniform(-1, 1) && (r, s) \in [D] \times [D] \ni r < s. \qquad (2.9)
\end{aligned}$$

Under the aforementioned prior specifications and the data augmentation steps, we have the full conditional of the individual's continuous latent true disease status is given by

$$T_{id}|T_{i(-d)}, \mathbf{T}_{(-i)}, \widetilde{y}_i, \boldsymbol{\beta}, \mathbf{R}, \mathbf{Z} \sim \begin{cases} TN(T_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\Sigma}_{id}, 0, \infty), & \text{if } \widetilde{y}_{id} = 1, \\ TN(T_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\Sigma}_{id}, -\infty, 0), & \text{if } \widetilde{y}_{id} = 0, \end{cases} \qquad (2.10)$$

where

$$\tilde{\boldsymbol{\mu}}_{id} = \mu_{id} - \mathbf{R}_{d,(-d)}(\mathbf{R}_{(-d),(-d)})^{-1}(t_{i(-d)} - \boldsymbol{\mu}_{i(-d)}) \tag{2.11}$$

$$\tilde{\Sigma}_{id} = \mathbf{R}_{d,d} - \mathbf{R}_{d,(-d)}(\mathbf{R}_{(-d),(-d)})^{-1}\mathbf{R}_{(-d),d}. \tag{2.12}$$

While the individual's latent true disease status is

$$\widetilde{y}_{id}|\widetilde{y}_{i(-d)}, \widetilde{y}_{(-i)}, \mathbf{S}_e, \mathbf{S}_p, \boldsymbol{\beta}, \mathbf{R}, \mathbf{T}, \mathbf{Z} \sim \text{Bernoulli}\left[g_{id}(1)/\left\{g_{id}(0) + g_{id}(1)\right\}\right] \tag{2.13}$$

with

$$g_{id}(x) = \left\{\prod_{j\in\mathcal{A}_i}\left(S_{e_j:d}^{Z_{jd}}\overline{S}_{e_j:d}^{1-Z_{jd}}\right)^{\mathbb{1}(x+\sum\widetilde{y}_{(-i)d}>0)}\left(S_{p_j:d}^{1-Z_{jd}}\overline{S}_{p_j:d}^{Z_{jd}}\right)^{1-\mathbb{1}(x+\sum\widetilde{y}_{(-i)d}>0)}\right\}p_{id0}^{1-x}(1-p_{id0})^{x} \tag{2.14}$$

where $x \in \{0, 1\}$, $p_{id,\widetilde{y}_{id}} = \int N(t_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\Sigma}_{id})\mathbb{1}(t_{id} > 0)^{\widetilde{y}_{id}}\mathbb{1}(t_{id} \leq 0)^{1-\widetilde{y}_{id}}$, $\overline{S}_{e_j:d} = 1 - S_{e_j:d}$, $\overline{S}_{p_j:d} = 1 - S_{p_j:d}$, and $\mathcal{A}_i = \{j : i \in \mathcal{P}_j\}$. In other words, $\mathcal{A}_i$ tracks which pools the ith patient is a member. Then the full conditional for the regression coefficients may be sampled simultaneously by

$$\boldsymbol{\beta}|\mathbf{R}, \mathbf{T}, \mathbf{Z} \sim N_Q(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}). \tag{2.15}$$

where $\tilde{\boldsymbol{\Sigma}} = \left\{\boldsymbol{\Sigma}_0^{-1} + \sum_{i=1}^{N}\mathbf{X}_i'(\mathbf{R})^{-1}\mathbf{X}_i\right\}^{-1}$ and $\tilde{\boldsymbol{\mu}} = \tilde{\boldsymbol{\Sigma}}\left\{\sum_{i=1}^{N}\mathbf{X}_i'(\mathbf{R})^{-1}\mathbf{T}_i\right\}$. Lastly, the correlation parameters are individually sampled using Metropolis-Hastings steps. For each Metropolis-Hastings step, we draw a candidate sample from a uniform random walk on the interval $[\rho_{rs} - \delta, \rho_{rs} + \delta]$ where we use reflection to contain the candidate in the interval $(-1, 1)$. With the candidate and original value, we easily compute the acceptance value for the Metropolis-Hastings step used to accept or reject the candidate value. The full symbolic representation of the entire posterior sampling algorithm is also given in Appendix A.

2.2.2 Logistic Model

The full hierarchy of the proposed logit model is

$$\begin{aligned}
\widetilde{Y}_{id} &= \mathbb{1}(T_{id} > 0) && i = 1, \dots, N, d = 1, \dots, D, \\
\mathbf{t}_i &\sim f_{\mathbf{T}_i}(\mathbf{t}_i | \mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) && i = 1, \dots, N, \\
\boldsymbol{\beta} &\sim N_Q(0, \boldsymbol{\Sigma}_{\mathbf{0}}), \\
\rho_{rs} &\sim Uniform(-1, 1) && (r, s) \in [D] \times [D] \ni r < s. \qquad (2.16)
\end{aligned}$$

where $f_{\mathbf{T}_i}(\mathbf{t}_i | \mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R})$ is as in 2.7. In order to compute full conditional posteriors, we face difficulties with the distribution of the multivariate logistic density. We cannot decompose the density in the same manner as the multivariate probit density. Thus we are unable to sample the distribution. Instead, we use importance sampling to sample a similar distribution. We adjust our estimations of the similar distribution using importance weights to recovery estimations of the original distribution.

For example, suppose the random variable X with probability density f is a difficult distribution to sample from and the random variable Y with probability density g is an easy distribution to sample from. Normally, we estimate $E_f[h(X)]$ by drawing $X_1, \dots, X_n$ and computing the approximation $E_f[h(X)] \approx 1/n \sum_i h(X_i)$. However, we can only draw $Y_1, \dots, Y_n$. Therefore, we use the following relationship:

$$E_f[h(X)] = \int h(x) f(x) dx = \int h(x) \frac{f(x)}{g(x)} g(x) dx = E_g \left[h(Y) \frac{f(Y)}{g(Y)} \right]. \qquad (2.17)$$

We can estimate the right hand side with the relation $E_g[h(Y) f(Y)/g(Y)] \approx 1/n \sum_i h(Y_i) f(Y_i)/g(Y_i)$. Where the terms $w_i = f(Y_i)/g(Y_i)$ are called the importance weights. The performance of importance sampling is improved by selecting a distribution that is similar to the original.

The multivariate logistic density is similar to the multivariate t distribution. Hence, we substitute the multivariate logistic density for the multivariate t distribution, $\mathcal{T}_{D,\nu}(\mathbf{t}_i | \boldsymbol{\mu}_i, \mathbf{R})$, and use importance sampling for estimation. The multivariate t distribution is ideal because it has heavier tails and we can optimize the multivariate t distribution to be close to the multivariate logistic distribution. Following O'Brien and Dunson [56], we choose the optimal value to be $\nu = \tilde{\nu} = 7.3$ and replace $f_{\mathbf{T}_i}(\mathbf{t}_i | \mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R})$ with $\mathcal{T}_{D,\tilde{\nu}}(\mathbf{X}\boldsymbol{\beta}, \tilde{\sigma}^2 \mathbf{R})$ where $\tilde{\sigma}^2 = \pi^2(\tilde{\nu} - 2)/3\tilde{\nu}$. With importance sampling, we denote the target or original posterior density P and the posterior density of from which we draw samples as P^*. The following is our sample posterior density:

$$P^*(\mathbf{S}_e, \mathbf{S}_p, \mathbf{X}, \boldsymbol{\beta}, \mathbf{R}|\mathbf{Z}, \widetilde{\mathbf{Y}}, \mathbf{T}) \propto \prod_{l=1}^{L} \prod_{j\in\mathcal{M}(l)} \prod_{d=1}^{D} \left\{ S_{e(l):d}^{z_{jd}} \overline{S}_{e(l):d}^{1-z_{jd}} \right\}^{\widetilde{z}_{jd}} \left\{ S_{p(l):d}^{1-z_{jd}} \overline{S}_{p(l):d}^{z_{jd}} \right\}^{1-\widetilde{z}_{jd}}$$
$$\times \prod_{i=1}^{N} \mathcal{T}_{D,\tilde{\nu}}(\mathbf{T}_i|\mathbf{X}_i, \boldsymbol{\beta}, \mathbf{R}) N_Q(\boldsymbol{\beta}; \mathbf{0}, \boldsymbol{\Sigma}_\mathbf{0}) \prod_{r<s\in[D]\times[D]} U(\rho_{rs}; -1, 1) \quad (2.18)$$

where $\overline{S}_{e(l):d} = 1 - S_{e(l):d}$, $\overline{S}_{p(l):d} = 1 - S_{p(l):d}$. The corresponding importance weights are given by

$$w \propto \frac{P(\mathbf{S}_e, \mathbf{S}_p, \mathbf{X}, \boldsymbol{\beta}, \mathbf{R}|\mathbf{Z}, \widetilde{\mathbf{Y}}, \mathbf{T})}{P^*(\mathbf{S}_e, \mathbf{S}_p, \mathbf{X}, \boldsymbol{\beta}, \mathbf{R}|\mathbf{Z}, \widetilde{\mathbf{Y}}, \mathbf{T})}$$
$$= \prod_{i=1}^{N} \frac{f_{\mathbf{T}_i}(\mathbf{T}_i; \mathbf{X}_i\boldsymbol{\beta}, \mathbf{R})}{\mathcal{T}_{D,\tilde{\nu}}(\mathbf{T}_i; \mathbf{X}_i\boldsymbol{\beta}, \tilde{\sigma}^2\mathbf{R})}. \quad (2.19)$$

To find the conditional distribution for P^*, we add another latent variable ϕ_i for the ith subject to sample from the multivariate t distribution. We have $\mathbf{T}_i|\boldsymbol{\beta}_1, \mathbf{R} \sim \mathcal{T}_{D,\tilde{\nu}}(\boldsymbol{\mu}_i, \tilde{\sigma}^2\mathbf{R})$ is equivalent to the hierarchical model $\mathbf{T}_i|\boldsymbol{\beta}, \mathbf{R}, \phi_i \sim N_D(\boldsymbol{\mu}_i, \tilde{\sigma}^2\phi_i^{-1}\mathbf{R})$ and $\phi_i \sim \Gamma(\tilde{\nu}/2, \tilde{\nu}/2)$. Most other notation is shared with the probit model with the exception of $\tilde{\tilde{\Sigma}}_{id} = \left\{R_{d,d} - R_{d,(-d)}(R_{(-d),(-d)})^{-1}R_{(-d),d}\right\}$ $\tilde{\sigma}^2/\phi_i$. Thus we get the similar full conditional distributions:

$$T_{id}|T_{i(-d)}, \mathbf{T}_{(-i)}, \widetilde{y}_i, \boldsymbol{\beta}, \mathbf{R}, \phi, \mathbf{Z} \sim \begin{cases} TN(T_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\tilde{\Sigma}}_{id}, 0, \infty), & \text{if } \widetilde{y}_{id} = 1, \\ TN(T_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\tilde{\Sigma}}_{id}, -\infty, 0), & \text{if } \widetilde{y}_{id} = 0. \end{cases} \quad (2.20)$$

While the individual's latent true disease status is

$$\widetilde{y}_{id}|\widetilde{y}_{i(-d)}, \widetilde{y}_{(-i)}, \mathbf{S}_e, \mathbf{S}_p, \boldsymbol{\beta}, \mathbf{R}, \phi, \mathbf{T}, \mathbf{Z} \sim \text{Bernoulli}\left[g_{id}(1)/\left\{g_{id}(0) + g_{id}(1)\right\}\right] \quad (2.21)$$

with

$$g_{id}(x) = \left\{ \prod_{j\in\mathcal{A}_i} \left(S_{e_j:d}^{Z_{jd}} \overline{S}_{e_j:d}^{1-Z_{jd}} \right)^{\mathbb{1}(x+\sum \widetilde{y}_{(-i)d}>0)} \left(S_{p_j:d}^{1-Z_{jd}} \overline{S}_{p_j:d}^{Z_{jd}} \right)^{1-\mathbb{1}(x+\sum \widetilde{y}_{(-i)d}>0)} \right\} {p'}_{id0}^{1-x}(1 - {p'}_{id0})^x \quad (2.22)$$

where ${p'}_{id,\widetilde{y}_{id}} = \int N(t_{id}; \tilde{\boldsymbol{\mu}}_{id}, \tilde{\tilde{\Sigma}}_{id})\mathbb{1}(t_{id} > 0)^{\widetilde{y}_{id}}\mathbb{1}(t_{id} \le 0)^{1-\widetilde{y}_{id}}$.

Then the full conditional for the regression coefficients may be sampled simultaneously by

$$\boldsymbol{\beta}|\mathbf{R}, \phi, \mathbf{T}, \mathbf{Z} \sim N_Q(\tilde{\tilde{\boldsymbol{\mu}}}, \tilde{\tilde{\boldsymbol{\Sigma}}}). \tag{2.23}$$

where

$$\tilde{\tilde{\boldsymbol{\Sigma}}} = \left\{ \boldsymbol{\Sigma}_0^{-1} + \tilde{\sigma}^{-2} \sum_{i=1}^{N} \phi_i \mathbf{X}_i'(\mathbf{R})^{-1}\mathbf{X}_i \right\}^{-1} \tag{2.24}$$

$$\tilde{\tilde{\boldsymbol{\mu}}} = \tilde{\tilde{\boldsymbol{\Sigma}}} \left\{ \tilde{\sigma}^{-2} \sum_{i=1}^{N} \phi_i \mathbf{X}_i'(\mathbf{R})^{-1}\mathbf{T}_i \right\}. \tag{2.25}$$

While the full conditional of the gamma mixture is sampled individually as

$$\phi_i|\boldsymbol{\beta}, \mathbf{R}, \mathbf{T}, \mathbf{Z} \sim \Gamma\left[(\tilde{\nu}+2)/2, \left\{\tilde{\nu} + \tilde{\sigma}^{-2}(\mathbf{T}_i - \boldsymbol{\mu}_i)'(\mathbf{R})^{-1}(\mathbf{T}_i - \boldsymbol{\mu}_i)\right\}/2\right]. \tag{2.26}$$

Again, the correlation parameters are individually sampled using Metropolis-Hastings steps. For each Metropolis-Hastings step, we draw a candidate sample from a uniform random walk on the interval $[\rho_{rs} - \delta, \rho_{rs} + \delta]$ where we use reflection to contain the candidate in the interval $(-1, 1)$. With the candidate and original value, we easily compute the acceptance value for the Metropolis-Hastings step used to accept or reject the candidate value. The full symbolic representation of the entire posterior sampling algorithm is also given in Appendix A.

2.2.3 Unknown Testing Errors

We now relax the assumption that testing errors are known and fixed. The sensitivity and specificity of the model can be varied. Furthermore, the approach outlined in this section is general and applied to both the logistic and probit models. Priors can be informed with the use of previous studies [1] that seek to estimate the sensitivity and specificity. As part of validation of the assay the manufacturer typically performs clinical trials for each combination of disease, specimen type, and gender. For each test configuration, patients of known disease status are individually tested using the assay. With the patients true disease status, each of the assay's testing outcomes to be categorized as true positive ($TP_{l,d}$), false positive ($FP_{l,d}$), true negative ($TN_{l,d}$) and false negative ($FN_{l,d}$). Thus allowing for the estimation of sensitivity and specificity for each test configuration. The $TP_{l,d}$, $FP_{l,d}$, $TN_{l,d}$, and $FN_{l,d}$ from validation studies can be used to inform our assay accuracy priors.

We chose a beta distribution as a prior for both sensitivity and specificity. Many works in group testing such as Warasi et al. [72], McMahan et al. [41], Joyner et al. [43], etc. use a beta distribution prior for assay accuracies. We chose a beta distribution as a prior for sensitivity and specificity as we can switch from an informative prior to an uninformative prior that is uniform on $(0, 1)$. In this dissertation, we develop two variations to estimating assay accuracies.

One approach would be to estimate an independent sensitivity and specificity for every test configuration. In the following, we use a beta prior for every $S_{e(l):d}$ and $S_{p(l):d}$ which leads to the following prior for the set of sensitivities and specificities:

$$P(\mathbf{S}_e)P(\mathbf{S}_p) = \prod_{l=1}^{L} f(S_{e(l):d}; TP_{l,d}, FN_{l,d}, 0, 1) f(S_{p(l):d}; TN_{l,d}, FP_{l,d}, 0, 1) \tag{2.27}$$

where $f(x; \alpha, \beta, a, b)$ is the truncated beta prior distribution and the limits $a = 0, b = 1$ recover the beta distribution. Note when $\alpha = \beta = 1$, it follows the uniform distribution, an uninformative prior. Another approach is to link test configurations to others where they only differ by pool size. For example, test configuration l and l' may have the same gender, and specimen type but represent individual testing and group testing. We consider dilution effects with an additional constraint on $S_{e(l):d}$ and $S_{p(l):d}$. Without the data for an informed prior, this constraint is used to partially inform the sensitivity and specificity. In larger pools, we expect a decrease in the sensitivity and an increase in specificity. It is natural to enforce the ordering $S_{e(l):d} > S_{e(l'):d}$ and $S_{p(l):d} < S_{p(l'):d}$ for $|\mathcal{P}_l| < |\mathcal{P}_{l'}|$ by type. The ordering is a way to incorporated the dilution effect and bleed information across the different testing configurations as they relate to pool size. In the Iowa example, we have the stratum for CT, swab specimen and female with pools of size one and four. In this case, we may write the prior distribution of sensitivities for the two pools as the following:

$$P(\mathbf{S}_e) = f(S_{e(1):1}; TP_{1,1}, FN_{1,1}, S_{e(2):1}, 1) f(S_{e(2):1}; TP_{2,1}, FN_{2,1}, 0, 1). \tag{2.28}$$

In order to generalize, we see the following is equivalent:

$$\begin{aligned} P(\mathbf{S}_e) &\propto f_{S_{e(1):1}|S_{e(2):1}}(S_{e(1):1}; TP_{1,1}, FN_{1,1}, 0, 1|S_{e(1):1} > S_{e(2):1}) f_{S_{e(2):1}}(S_{e(2):1}; TP_{2,1}, FN_{2,1}, 0, 1) \\ &\propto f(S_{e(1):1}; TP_{1,1}, FN_{1,1}, 0, 1) f(S_{e(2):1}; TP_{2,1}, FN_{2,1}, 0, 1) \mathbb{1}(S_{e(1):1} > S_{e(2):1}). \end{aligned} \tag{2.29}$$

We add in an indicator function to represent the partial orderings within each test configuration. Let P be the partition of the testing configurations $[L]$ such that $\bigcup_{V\in P} V = [L]$ and for $l, l' \in V$ then test configuration l and l' differ only by size. That is for every $V \in P$, V represents the subset of test configurations where the stratum is the same except they vary in pool size. Then we have the following distributions:

$$P(\mathbf{S}_e) \propto \prod_{V\in P}\prod_{l\in V} f(S_{e(l):d}; TP_{l,d}, FN_{l,d}, 0, 1) \prod_{l'\in V:|\mathcal{P}_{l'}|<|\mathcal{P}_l|} \mathbb{1}(S_{e(l'):d} > S_{e(l):d}), \tag{2.30}$$

$$P(\mathbf{S}_p) \propto \prod_{V\in P}\prod_{l\in V} f(S_{p(l):d}; TN_{l,d}, FP_{l,d}, 0, 1) \prod_{l'\in V:|\mathcal{P}_{l'}|<|\mathcal{P}_l|} \mathbb{1}(S_{p(l'):d} < S_{p(l):d}). \tag{2.31}$$

The difference between assay accuracies with out ordering in Eqn. 2.27 and the product of assay accuracies with ordering in Eqn 2.30 and Eqn. 2.31 is the addition of indicator functions. We refer to the two approaches as no ordering and ordering, respectively.

With the beta distributions as a prior for the sensitivity and specificity, we compute the full conditional distribution under both approaches. The full conditional distribution when the test configurations are independent with no ordering are as follows:

$$S_{e(l):d} \sim f\left\{\sum_{j\in\mathcal{M}(l)} Z_{jd}\widetilde{z}_{jd} + TP_{l,d}, \sum_{j\in\mathcal{M}(l)} (1-Z_{jd})\widetilde{z}_{jd} + FN_{l,d}, 0, 1\right\} \tag{2.32}$$

$$S_{p(l):d} \sim f\left\{\sum_{j\in\mathcal{M}(l)} (1-Z_{jd})(1-\widetilde{z}_{jd}) + TN_{l,d}, \sum_{j\in\mathcal{M}(l)} Z_{jd}(1-\widetilde{z}_{jd}) + FP_{l,d}, 0, 1\right\} \tag{2.33}$$

When test configurations share information through ordering, we have the full conditional distribution as follows:

$$S_{e(l):d} \sim f\left\{\sum_{j\in\mathcal{M}(l)} Z_{jd}\widetilde{z}_{jd} + TP_{l,d}, \sum_{j\in\mathcal{M}(l)} (1-Z_{jd})\widetilde{z}_{jd} + FN_{l,d}, A_{e(l):d}, B_{e(l):d}\right\}, \tag{2.34}$$

$$S_{p(l):d} \sim f\left\{\sum_{j\in\mathcal{M}(l)} (1-Z_{jd})(1-\widetilde{z}_{jd}) + TN_{l,d}, \sum_{j\in\mathcal{M}(l)} Z_{jd}(1-\widetilde{z}_{jd}) + FP_{l,d}, A_{p(l):d}, B_{p(l):d}\right\}, \tag{2.35}$$

where if $V \in P$ is the subset of the partition containing l and l' then

$$A_{e(l):d} = \begin{cases} 0, & \text{if } \nexists\, l' \text{ s.t. } |\mathcal{P}_{l'}| > |\mathcal{P}_l|, \\ \max\{S_{e(l'):d} \,|\, |\mathcal{P}_{l'}| > |\mathcal{P}_l|\}, & \text{otherwise.} \end{cases} \tag{2.36}$$

$$B_{e(l):d} = \begin{cases} 1, & \text{if } \nexists\, l' \text{ s.t. } |\mathcal{P}_{l'}| < |\mathcal{P}_l|, \\ \min\{S_{e(l'):d} \,|\, |\mathcal{P}_{l'}| < |\mathcal{P}_l|\}, & \text{otherwise.} \end{cases} \tag{2.37}$$

$$A_{p(l):d} = \begin{cases} 0, & \text{if } \nexists\, l' \text{ s.t. } |\mathcal{P}_{l'}| < |\mathcal{P}_l|, \\ \max\{S_{p(l'):d} \,|\, |\mathcal{P}_{l'}| < |\mathcal{P}_l|\}, & \text{otherwise.} \end{cases} \tag{2.38}$$

$$B_{p(l):d} = \begin{cases} 1, & \text{if } \nexists\, l' \text{ s.t. } |\mathcal{P}_{l'}| > |\mathcal{P}_l|, \\ \min\{S_{p(l'):d} \,|\, |\mathcal{P}_{l'}| > |\mathcal{P}_l|\}, & \text{otherwise.} \end{cases} \tag{2.39}$$

$$\tag{2.40}$$

The full symbolic representation of the posterior sampling algorithm for $S_{e(l):d}$ and $S_{p(l):d}$ is also given in Appendix A

2.3 Simulation

We use the following population models to evaluate our estimation methodology:

$$\begin{aligned} \text{probit: } & \mathbf{T}_i = (T_{i1}, T_{i2}) \sim N_2(\mathbf{X}_i\boldsymbol{\beta}, \mathbf{R}) \\ & \widetilde{\mathbf{Y}}_i = (\mathbb{1}(T_{i1} > 0), \mathbb{1}(T_{i2} > 0))' \end{aligned} \tag{2.41}$$

$$\begin{aligned} \text{logistic: } & \mathbf{E}_i = (E_{i1}, E_{i2}) \sim N_2(\mathbf{0}, \mathbf{R}) \\ & \gamma_i \sim \Gamma\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \\ & T_{id} = \mathbf{x}_i'\boldsymbol{\beta} + \operatorname{logit}\left(F_\nu(E_{id}/\sqrt{\gamma_i})\right) \\ & \widetilde{\mathbf{Y}}_i = (\mathbb{1}(T_{i1} > 0), \mathbb{1}(T_{i2} > 0))' \end{aligned} \tag{2.42}$$

where $\mathbf{X}_i\boldsymbol{\beta} = (\mathbf{x}_i'\boldsymbol{\beta}_1, \mathbf{x}_i'\boldsymbol{\beta}_2)'$ with $\mathbf{x}_i = (1, x_{i1}, x_{i2})'$, $\rho_{12} = 0.3$, and $F_\nu(\cdot)$ is the student-t cumulative distribution function with $\nu = 7.3$ degrees of freedom. In addition, $x_{i1} \sim N(0,1)$ and $x_{i2} \sim$ Bernoulli(0.5). With the two population models, we choose $(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2) = (-2.1, 0.5, 0.5, -1.7, 0.5, 0.5)$

for the probit model and $(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2) = (-3.3, 0.5, 0.5, -2.55, 0.5, 0.5)$ for the logistic model. These parameters give the population prevalence of about 5% and 10% for disease 1 and 2 respectively. These models generate the true disease status and covariates for every simulated patient. We generate $\{(\widetilde{\mathbf{Y}}_i, \mathbf{X}_i)\}$ for $i = 1, \ldots, N$ once for every comparison and repeat this 500 times.

In these simulations, we study four testing scenarios: individual testing (IT), master pool testing (MPT), Dorfman's testing (DF), and two-stage array testing (AT). With IT, we perform a multiplex assay for both diseases on every patient. The simplest form of group testing is MPT. We arrange patients into separate pools of fixed size, four, and only perform the tests on the pools. We choose pool sizes of four to reflect the typical pool size used in our application set. In this case, the patient groups are disjoint, i.e., no patient can be in two groups. Moreover, we test for two diseases in each pool. For DT, we arrange all patients in disjoint sets of four and test their pooled results for two diseases similar to MPT. We retest all members of pools with any positive results, individually, for both diseases. Pools with negative test results for both diseases do not need retests because individuals in the pool are considered negative. For AT, we partition the patients into subgroups of sixteen. The subgroup is arranged in a square four by four array and the columns and rows are pooled. We follow the procedure outlined by Kim et. al. [25] for testing. Imperfect testing is used for each scenario. We generate the true pool status $\widetilde{Z}_{jd} = \mathbb{1}(\sum_{i \in \mathcal{P}_j} \widetilde{Y}_{id} > 0)$ simulate the test response by $Z_j | \widetilde{Z}_{jd} = 1 \sim \text{Bernoulli}(S_{e(l):d})$ and $Z_j | \widetilde{Z}_{jd} = 0 \sim \text{Bernoulli}(1 - S_{p(l):d})$ with $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$.

For each population model, probit and logistic, we examine the performance of the model under three cases for IT, MPT, DT, and AT. In the first case, we assume that assay accuracies are known and the correlation between diseases, ρ_{12}, is unknown. In the second case, we change the assumption that the assay accuracies are unknown and are estimated separately. In the third case, we assume the unknown assay accuracies have an ordering. In each of the cases, we simultaneously estimate the unknown parameters. Our simulation results for the probit population model for the cases are found in Tables B.1, B.2, and B.3 in Appendix B. The logistic population model results for the six cases are found in Tables B.4, B.5, and B.6.

For the simulation results, we use $N = 5008$ individuals so that every testing schema was complete without remainder pools. We chose priors with minimal information and diffuse. For the regression parameters $\boldsymbol{\beta}$, we use the multivariate normal prior with mean 0 and correlation matrix

$\mathbf{\Sigma}_0 = 100I_6$. For the correlation parameter, we use a uniform prior on the interval $[-1, 1]$. For the sensitivity and specificity, we used independent beta priors, b$(1, 1)$. For each case, we generate a population and permute the individuals before each testing protocol. We perform $100,000$ posterior draws, retain every tenth draw, and discard the first half as burn-in. This leaves us with $5,000$ posterior samples for each parameter.

We run $K = 500$ simulations. For each simulation, we compute the posterior mean and standard deviation using the posterior samples. We find the average of K posterior mean estimates of the bias, the standard deviation of the posterior mean estimates (SSD), and the average of the posterior standard deviation estimates (ESE). In addition, we calculate the 95% equal-tailed credible interval (ETI) for every parameter in the simulation with $\alpha = 0.05$. We verify if the true parameter is within the interval. The mean number of times the parameter is within the credible interval is the empirical coverage probability (CP95).

Table B.1 shows the simulation results when assay accuracies are unknown but the regression coefficients and the correlation term are simultaneously estimated for the probit population and model. We observe that MPT is the worst performing with roughly double the bias, ESE, and SSD as the best performing AT at the cost of approximately a third of the number of tests used. Notably with the exception of the bias in the first disease intercept term and the correlation parameter. In general, the bias, ESE, and SSD decrease in the order MPT, IT, DT, and AT with a few exceptions. Another notable observation is that for IT, DT, and AT the bias for the regression coefficients are the same sign as the true values.

We ran a simulation where assay accuracies are simultaneously estimated for the probit model. Our results are recorded in Table B.2. The regression coefficient estimations decrease for DT and AT and increase for IT and MPT over the previous study where assay accuracies are known. This causes a significant reduction in the coverage probabilities for the regression coefficients for MPT. The correlation parameter estimations increases significantly for IT and MPT where as the bias doubles with an increase in SSD and ESE for DT and AT from the first simulation. The estimations for sensitivity and specificity are poor for IT and MPT but are accurate for DT and AT. AT again has the lowest biases, SSD, and ESE for the assay accuracies with only two exceptions. Our third simulation examines DT and AT with and without ordering the assay accuracies. The estimations are found in Table B.3. We find that SSD and ESE values can be slightly reduced with ordering implemented but the biases only improve on one intercept term and the larger valued assay accuracies.

The forth, fifth and sixth simulations are similar for the previous three simulations except that we use the logistic population and logistic model. The results in Table B.4 and Table B.5 are when the assay accuracies are known and unknown, respectively. The analysis is similar to the probit model with a similar prevalence and number of tests performed. Table B.6 shows the results for when the assay accuracies are unknown and ordered. The results show that the majority of SSD and ESE estimates improve with ordering and the bias under ordering is only better for the higher valued accuracies.

There are a few themes from the simulation results from both populations. The first theme is that AT requires around eight percent more tests than DT in our simulation setup but while AT demonstrate at least a 23% decrease in the number of tests performed compared to IT. The second theme is that IT and MPT struggle to estimate sensitivity and specificity with flat priors. Both IT and MPT provide reasonable regression coefficient estimates when the assay probabilities are known. The third theme is that enforcing an ordering on the assay accuracies shows only a trade off effect between bias and variance. Lastly, DT and AT demonstrate lower bias, SSD, and ESE parameter estimations with reasonable coverage probabilities over IT and MPT.

2.4 Data Application

Iowa's SHL is the state's public health and environmental laboratory. It provides services for disease detection, environmental monitoring, newborn and maternal screening, drinking water testing, and bioterrorism and chemical terrorism response and readiness. For STD screenings, medical providers collect samples from patients and the specimen are sent to the SHL facilities for testing. For chlamydia and gonorrhea screening, all urine samples are tested individually and specimens collected on swabs from females are tested using DT with the dual assay for retesting according to Tebbs, McMahan, and Bilder [47]. The testing protocol based on gender and specimen type is also common in Idaho according to Lewis, Lockary, and Kobic [29]. The stratification helps separate high prevalence populations from low prevalence populations such as men who get tested due to STD symptoms versus regular screenings for females during pregnancy. In this section, we examine clinical results for the 2014 Iowa clinical data for females. Included in the data set are results from individually testing urine samples and both individual and pool testing swab specimens. Our analysis of the data is presented in two tables: $\boldsymbol{\beta}$ estimation in Table B.9, ρ, sensitivity and specificity estimations are in Table B.10.

The data contains 13,862 female's test results for classification with individual covariates. The test results are composed of 4,316 individual urine specimens, 2,272 swab specimen master pools of size 4, 13 swab specimen master pools of size 3, 1 swab specimen master pool of size 2, and 417 individual swab specimens. We include nine patient covariates for modeling: one continuous and eight categorical. The continuous variable is the patient's age in years. The coefficient for age is $\beta_{d,1}$ for $d = 1, 2$. A density graph by specimen type can be found in Figure B.2. The majority of patient ages are found in teens to late thirties with the swab specimen type age distribution right-shifted compared to the urine specimen type age distribution. We reduce the categorical variable for race into three categories white (W), black (B), and other as black and white were the two majority classes. We one-hot encoded the category such that the class for other is the baseline and the majority classes were denoted by the indicators $\beta_{d,2}$ for W and $\beta_{d,3}$ for B for $d = 1, 2$. The rest of the categorical variables do not need one-hot encoding as they have two classes. The other indicators are patient-reported risks and symptoms for new sexual partner in the last 90 days (RNP, $\beta_{d,4}$), multiple partners in the last 90 days (RMP, $\beta_{d,5}$, sexual contact with a STD-positive partner (RC, $\beta_{d,6}$), symptoms of infection (S, $\beta_{d,7}$), specimen type swab (1) or urine (0) (ST, $\beta_{d,8}$), signs of cervicitis (SC, $\beta_{d,9}$) and signs of pelvic inflammatory disease (SP, $\beta_{d,10}$) for $d = 1, 2$. We provide descriptive statistics for the categorical covariates in Table B.7. We simultaneously estimate both the urine and swab samples using both the probit model and the logistic model. With the correlation term, sensitivities, specificities, and regression coefficients, we estimate 43 parameters.

To combine the testing types into one large dataset, we have separate parameters for the testing error of each type of specimen in the model. Moreover, we make informed estimates of sensitivity and specificity for each tuple of disease, test type, and group size. Hologic's performance data, used to establish their assay probabilities, provides the necessary counts to inform the priors on each sensitivity and specificity per disease and test type combination. We provide the data for female swab and urine specimens in Table B.8. Since there is no data based on group size, all the varying group sizes have the same prior. Additionally, we explore enforcing an ordering on the sensitivity and specificity based on group size and specimen type. For example, we have five assay types $l = 1, 2, 3, 4$ for the four group sizes of swab specimens and $l = 5$ for individual urine specimens. If ordering is in effect, we have the partial order of $S_{e(1),d} \geq S_{e(2),d} \geq S_{e(3),d} \geq S_{e(4),d}$, and $S_{p(1),d} \leq S_{p(2),d} \leq S_{p(3),d} \leq S_{p(4),d}$ for $d = 1, 2$. We chose the same priors from our simulations for the correlation parameter and the regression coefficients. We generated $100,000$ posterior draws and burned the first half and saved every tenth draw. This left us with $5,000$ posterior draws for each parameter. We computed the posterior mean, sample standard deviation of posterior draws, 95% highest posterior density interval (HPDI), and 95% equal-tailed interval. We find that the direction of the regression coefficients is in line with McMahan, Tebbs, Hanson, and Bilder [41]. From our estimations, We did not observe the dilution effect in sensitivity when sensitivity was not partially ordered. Also we note that for the probit model ST is not significantly different from zero for chlamydia with ordering but ST is with ordering. For the probit model, RMP was not significantly different from zero for the disease gonorrhea with ordering but RMP is significant for no ordering. For the logit model, we found that S and SC where not significantly different from zero for gonorrhea without ordering but are with ordering enforced.

We have proposed a Bayesian regression model to estimate generalized linear models with data from most group testing protocols for multiplex assays. Our implementation has the property that the marginal distributions following the probit and logit link functions. This property allows us to have estimate interpretable regression coefficients. There are some natural extensions of this work that may be of interests. One could include variable selection strategies for model reduction steps. The correlation between infections could conditionally depend on covariates. Non-linear relations could be present in the covariates. Additionally extensions to handle non-discriminating testing procedures could be of interest.

Chapter 3

GPU acceleration with NVIDIA CUDA

Graphic processing units (GPU) are supplemental hardware to CPUs that are used for graphics rendering. To achieve real-time graphics, GPUs are designed for highly parallelized tasks. Graphics rendering involves intricate model meshes which are composed of millions of triangles, shaders to paint the textures, color and lights on each triangle and matrix transformations applied to every pixel and vertex of the model to move it in space. While the demand for graphics performance fuels the GPU market, other numerically intensive tasks take advantage of the high throughput numerical operations. NVIDIA is a market leader in GPUs and supports general purpose computing on graphics processing units (GPGPU) with NVIDIA's CUDA (Compute Unified Device Architecture) Software Development Kit (SDK). We use the NVIDIA CUDA SDK for all of our GPU accelerated programs. This chapter is dedicated to introducing the SDK and hardware architecture. In Section 3.1, we address the programmer's model or an abstraction of the hardware. The programmer's model will cover the types of memory on the GPU as well as the thousands of computational cores or CUDA cores on the GPU. In Section 3.2 and 3.3, we setup a development environment and learn how to compile and execute code. In Section 3.4 and Section 3.5 we provide basic examples to springboard into our next project.

3.1 Programmer's Model

While we discuss GPU acceleration in this chapter, a GPU is not always needed. There are several metrics to evaluate needing a GPU. The first consideration is the size of the task. Sometimes tasks are too small or simple causing the overhead costs of using and programming a GPU to outweigh the benefits. On the other end of the spectrum, really large tasks sometimes do not warrant using a GPU. Large tasks spend time sending messages between computers and waiting on those communications. The cost of communication can drown out the benefits of using GPUs as well as add unnecessary complexity. After all these considerations, many tasks benefit from GPU acceleration.

The current GPU architecture has changed dramatically in the last couple of years [75] [76] [77]. NVIDIA GPU's hardware is composed of a medley of dedicated circuitry. CUDA cores are next to tensor cores and ray-tracing cores as well as traditional texture units. The general organization is the following hierarchy from top to bottom: Graphic Processing Clusters (GPCs), Texture Processing Clusters (TPCs), Streaming Multiprocessors (SMs), and finally CUDA cores. We focus on the architecture around the CUDA cores as they are most relevant to this dissertation. Within each CUDA core, there is a floating-point and an arithmetic logic unit that performs the computations. The CUDA core is abstractly associated with a thread or small set of independent instructions, and threads are organized into thread blocks. At the hardware level, the threads are organized into warps of 32 threads because there are some multiple of 32 CUDA cores within each SM, which plays an important part in scheduling tasks. In a few instances, we organize tasks in multiples of 32 to maximize the efficiencies of synchronization in a warp.

Synchronization is important because the NVIDIA GPUs implement a single instruction, multiple thread (SIMT) model. In a parallelized program, every thread executes the same set of instructions. At the warp level, instructions are executed in lockstep. For example, an instruction could be: add the numbers in the first two registers or grab a number in a memory location plus an offset. Machine commands are executed on the exact same clock cycle in the GPU in all 32 threads of the warp. Outside of the warps, the same code might be running a little off from one another. Therefore, the timing of memory writes and reads become an important consideration in this dissertation.

We consider a simple hardware setup of a single CPU and GPU. The programmer considers two locations: the host (CPU) and the device (GPU). Programming instructions must be sent to each location. In addition, data must be sent from the host to the device and vice versa. These data transfers between host and device are potential choke points. The data transfers can be either efficiently dispatched or separated from the computation by concurrent operations. Data transfers can be done asynchronously through the use of streams while other computations proceed.

The next important part of the architecture is the memory model. Reading and writing data takes time or clock cycles as there are different steps and pathways to complete the task. Memory can have multiple hardware layers from the fastest registers in the processor through multiple levels of cache to the system random access memory. Since we focus on the device, we assume the host only has global memory and the compiler and run time systems takes care of the rest. The device has memory on three different levels; global, local and shared that must be managed by the user. The easiest to use is global memory, every thread has read and write access to global memory. It must be managed such that read and write operations are not interleaved to create an unknown memory state. In contrast, local memory is only available to the thread and does not need as much care for scope and timing as it is isolated from being touched by other threads. Moreover accessing and fetching local memory is either similar to global memory or it is similar to a register on the chip. Local memory could be stored in registers, the fastest type of memory, but is usually isolated in the global memory of the device, the slowest type of memory. The compiler will try to optimize memory management based on the size of the data being retrieved and its usage. The guaranteed fastest memory is shared memory which trades accessibility and complexity for speed. Finally shared memory's scope is between global and local memory. It is available to all the threads in the same block and is comparable to registers and faster than global memory.

In the programmer's model, we think from the point of view of the host as the host orchestrates the work. We create variables and do computations all within host memory. To use the GPU, we send data and commands to the GPU to do work. There is a cost to copying the information over to the GPU and back to the CPU, so any GPU computation must outweigh the cost of transmission. To marginalize the overhead cost of sending data, we use data streams. We sequence the data to send it in chunks so that the GPU starts computing a chunk while the next is transferred. There are additional speed-ups from pinning host memory as it usually is automatically done to start a transfer. However we choose not to do this as our data transfers are usually at the beginning and end of the computation and the time savings do not scale with the models we implement. Once data is in the device, on occasion, we transfer to local or shared memory, which is data accessible only to a core or to the block. Shared data can be allocated statically or dynamically through the kernel function. We can move data once locally and then transfer it back at the end of a computation instead of repeatedly accessing global memory in a thread. This is because global data access is slower than local/shared data. Another memory issue that we come across is race conditions. The order in which each thread reads and writes to memory may not be deterministic in the code. For example, two threads, A and B, write to memory a value in the same place in memory at the same time. The unknown condition of the memory, either the value from thread A or thread B or a mangled combination of the two, is called a race condition. There are concepts such as atomic operations, semaphores, mutex, and other ways to deal with this concurrency issue.

Once the data is transferred to the GPU, it is in the device's global memory. There are thousands of cores on the GPU that have access to global memory. The simple cores are directed by a kernel program that is executed in parallel. Multiple kernels can be written for each part of the task. The threads that execute the kernels are able to self identify, which means they know their thread index, and block index, which in turn allows us to direct their tasks and which part of memory to use. For example, in Chapter 5, we have every thread compute a complex random number then the first thread selects which random number to use. In Section 3.4, we use the thread index to choose what numbers to add in memory.

3.2 Setup

The following setup is for the Palmetto Cluster at Clemson University, a high performance computing (HPC) cluster; however, the concepts are still applicable in other settings. The cluster is a set of computers called nodes. The nodes are networked with high speed connections. When we connect to the cluster via SSH, the terminal is at the login node. The login node runs on Red Had Enterprise Linux as well as the rest of the cluster. In the login node, we may read and write files and use other basic Linux utilities, but we cannot run our code. Past the login node, the rest of the cluster has several thousand nodes with different interconnections and hardware availability. The output of the whatsfree command shows there are NVIDIA K20, K40, P100, and V100 GPUs associated with various other hardware configurations. The exact hardware we use depends on the availability, the features, and the tools we need for our work.

The Palmetto Cluster uses the Portable Batch System (PBS) to request the resources and job scheduling. The qsub command submits a request to run a script or run an interactive session. Options of the command allow one to request the number of cores, amount of memory, the number and type of GPUs as well as the length of the request. Once the command is entered, the scheduler adds our request to the queue, and once available, the terminal switches to the available node. In the interactive environment, the terminal prompt changes indicated one is in the node.

After we login into a node, we can load modules. Modules are easy to install software resources The command module avail contains a condensed list of modules. In our case, we use the command to find the correct version of CUDA, gcc and R. Even though the toolkit comes with NVIDIA CUDA Compiler (NVCC), it requires a C++ compiler. The latest compilers do not always work with NVCC. On Windows, Visual C++ 14.1 in Visual Studio 2017 did not work with NVCC but Visual C++ 14.0 in Visual Studio 2015 did work at the time of the setup. Sometimes unexpected behavior or limited features can be from the wrong toolkit and compiler versions. To check support, go to the Installation Guides for CUDA linked on the landing page [78] or directly to the Linux guide [79].

3.3 Compiling

The NVCC compiles in two stages for the GPU. The two stages let the programmer choose between targeting the program to run on a specific hardware architecture or allowing the final stage to be completed by a yet to be determined architecture at the cost of a slower startup. The first stage is the virtual architecture which compiles a PTX file. The virtual architecture is designated by the compute_xx designation. The final stage generates the binaries and compiles to a Cubin for the machine code to be ran by the GPU. The second stage can be left to run-time compilation when the target architecture is unknown.

When compiling a GPU program to use in R, many flags are required during compilation. The goal of each flag is to appropriately produce a binary to be run by R with additional libraries included.

1. nvcc is the program to be called. Its location is found in the PATH variable in the linux environment and placed by the previous module add calls.

2. –O3 is the code optimization level. The compiler will make several passes at the code to search for ways to optimize it. This automatic process is not a guarantee that it will be different from previous levels, run faster, will be more efficient or error free. There are three levels with the second level the most commonly used, and the third level is the riskiest.

3. –arch=compute_35 is the virtual architecture for 3.5 which is for Kepler, K40. compute_60 is the 6.0 compute architecture for Pascal, P100, compute_70 and compute_72 is the compute architecture for Volta, V100. It is recommended to target the lowest architecture for the features we want.

4. –code=sm_35 targets the 3.5 architecture binaries for Kepler, K40. Similarly, the sm_60 is the for 6.0 compute architecture for Pascal, P100, sm_70 and sm_72 is the compute architecture for Volta, V100. If we want to use run-time compilation swap out sm_xx for compute_xx.

5. –G indicates debug mode.

6. –lcublas –lcurand –lcusolver links the CUDA cublas, curand and cusolver libraries in the toolkit. The modules include the paths to the libraries and are automatically looked up, otherwise a direct path is required.

7. ––shared –Xcompiler –fPIC is the set of commands to get a binary file that R can use. The first part, ––shared, builds a shareable library. The second part, –Xcompiler – fPIC, sets a flag for position independent code where memory locations are offset rather than absolute addresses so that the library can be loaded anywhere in program memory.

8. –o cuHorse.so tells the compiler what the output filename should be.

9. cuHorse.cu is the file or files to be compiled.

Further information can be found in the NVCC documentation [80].

3.4 A Simple Program

We now show a simple program for R that uses a GPU to demonstrate using CUDA SDK. Appendix C.1 contains links to the R code and CUDA code. The goal of the code is to add two vectors. Parallelization occurs when we return the componentwise sum where each core adds their respective component together.

The R script loads the binary gpu_program.so for the GPU, creates a wrapper function to call the GPU code, creates inputs, then calls the function and unloads the GPU code. Note the .**C** function call ensures that copies of all the data, not references, are sent to the external function. While copies of the data are created, only pointers or memory locations will be passed to our function. In C/C++, a pointer is a memory location declared with an **int** $*$a. The address in memory of the variable a is &a and to get the actual value at the memory location it can be dereferenced by $*$ operator or a[0].

The next part is the CUDA code that we saved as add_vector.cu. In the program we include all the necessary libraries, in this case the CUDA runtime library, we declare the kernel for the device and write the host program. One of the ways kernels are declared is by the __global__ declaration specifier. When writing a kernel, the point of view of the device should be taken. The kernel should first identify itself by thread id, block id, and block dimension. Distinguishing the kernel's identity allows the kernel to use different data or compute different tasks. The kernel from the first example uses the thread index to add the values at the corresponding indices in the vectors. That means for a vector of size 10, 10 threads will be launched to add two numbers.

Following the simple kernel function, we return to the point of view of the host. The first part Extern ''C'' is a directive for the functions that can be called by the library. Next is the function, called in R. The variable cudaError_t is usually returned for every basic CUDA function. We check the error after each device function call or we wrap them in another function to check them. Since we are not running critical computations, they can be ignored as they usually indicate a catastrophic failure.

The basic way to allocate global memory on the device is by using cudaMalloc. We give a double pointer address so that the function can return a new location for the pointer in addition to the number of bits we want to allocate. Next we have the data on the host and move it to the device using cudaMemcpy. It requires the device and host pointers, the size of the memory, and the direction of the transfer.

Finally we execute the kernel. We launch with n blocks and 1 thread per block by the <<<...>>> operator which is the execution configuration syntax. Either we call a double in this example or a quad-tuple that has the stream and memory management in it. After the kernel is called and all threads executed, we still have our results on the device. We then send the results back to the host and free all the device and host memory that was dynamically allocated.

3.5 A Second Program

In this section the code is similar to the code in the previous section with the exception of using an outside library. A link to the program is in Appendix C.2. The wrapper and initialization from R remain nearly identical except for the function name. The library, cuBLAS, contains reusable code for basic linear algebra subroutines (BLAS) for NVIDIA GPUs. For this program, there is no need to write a kernel function for adding two vectors, the function is part of the library. Instead, the program needs to be made aware of the library and where to find the code to run it.

To use the library, we introduce a handler. The handler stores the location of resources to using the library. There is a price for using the library, most of which is paid for in resources and time when the handler is initialized. We let cuBLAS manage all the threads, blocks, etc. by calling the function from the host. Later we must destroy the handler to free system resources back to the host. Additionally, cublasSetVector and cublasGetVector are are just wrappers for cudaMemcpy. However they have a nice increment function that allows us to grab the diagonal out of a matrix or any other subset.

The most subtle part of the function call is how alpha, incx, iny are set. They are all set to be 1 in the BLAS <t>axpy call. Note there is a mix of variables from the host cublas_handle, n [0], 1 and device, d_x, d_y. In the documentation, if a variable says host or device, the default setting is host. To toggle between where a variable exists, use cublasSetPointerMode, which is documented in the cuBLAS helper functions. For more information, we refer to the cuBLAS documentation [81]. We use these methods over the next chapters applied to more complex programs.

3.6 Performance

We run two different simulations to demonstrate the effectiveness of our two simple programs. In each simulation, we compare running R code (CPU only) versus the simple kernel (CUDA kernel) and the built-in library (CUDA library) programs. We repeat the simulations for numerical vectors of size 10^i for $i = 1, \ldots, 9$ and compute six summary statistics after running each program

100 times. We chose to scale up to vectors of size 10^9 because it was the last increment that fits within our GPU's memory. The first simulation demonstrates the overhead costs of running a GPU program. The data found in Table C.1 and Figure C.1 shows the cost of starting a GPU program and transferring data to and from the GPU outweighs the computation time by an order of magnitude. Moreover, the data demonstrates that the GPU libraries, while convenient, incur additional overhead costs. The second simulation demonstrates the acceleration of running a GPU program when we increase the numerical load by adding two vectors not once but 100 times. The results in Table C.2 and Figure C.2 show the CUDA kernel and CUDA library programs accelerate the computations as the vector size increases. When adding larger vectors, the CUDA kernel and CUDA library programs are over an order of magnitude faster and the CUDA library is optimized better than our custom kernel. The simulations demonstrate that we can see acceleration benefits from large computational tasks that stay on the GPU and using built-in libraries.

Chapter 4

GPU Acceleration for Logistic Regression with Penalty

In this chapter, we are interested in measuring the benefits of GPU acceleration. Applications for GPU acceleration are numerous but we focus on optimization, a common machine learning task. Optimization is a hard problem because many problems do not have closed form analytic solutions. In linear regression, the ordinary least squares solution is an example of an analytic solution. In particular, we demonstrate GPU acceleration on the logistic regression with ℓ^1 regularization as our objective function. We choose logistic regression as it has no analytic solution and it is a common link for binary data. For example, McMahan et. al. [41] use logistic regression to relate disease status to covariates in a group testing application.

The goal of our optimization is to find the regression coefficients that maximize the likelihood function or, with a simple negation, the equivalent problem of minimizing the objective function with the penalty term. A descent method is a common way to find the minimum of the objective function. We employ two types of descent methods: stochastic gradient descent and stochastic coordinate descent in different programming languages to see the performance improvements. A survey by Yuan et al. [82] includes other methods that we do not explore here.

We compare the formulation of Newton's method to stochastic descent methods to understand the origins of the descent methods and trade-offs. Newton's method is an iterative method for root finding with quadratic convergence. It has applications in optimization problems because finding critical values or potential extrema is a root finding problem. We wish to minimize the scalar function $F(\mathbf{x}) \in \mathbb{R}$ where $\mathbf{x} \in \mathbb{R}^d$ and $F(\mathbf{x})$ typically convex. We then wish to find critical points of the gradient $\nabla F(\mathbf{x}) = 0$ as potential solutions. Newton's method gives a sequence of points that converge to the solution, $\mathbf{x}_{t+1} = \mathbf{x}_t - \nabla^2 f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)$. Each iteration of Newton's method moves closer to the critical point. In particular, it moves in the general direction of $\nabla f(\mathbf{x}_t)$ where $\nabla^2 f(\mathbf{x}_t)^{-1}$ is a linear transformation that optimally scales and rotates the direction. One downside of Newton's method is that it can be expensive to compute $\nabla^2 f(\mathbf{x}_t)^{-1}$ on each iteration.

Gradient descent is an iterative method where each iteration moves in the direction of the gradient similar to Newton's method. Gradient descent is represented by the iterations $\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma_t \nabla f(\mathbf{x}_t)$ for some γ_t called the step size. There are two methods we explore similar to gradient descent; stochastic gradient descent and coordinate descent. Both these methods stem from ideas of Robbins and Monro [83] who first presented the idea to solve optimization problems stochastically. For gradient descent, the step size is called the learning rate η_t and for coordinate descent, the step size is denoted α_t.

In stochastic gradient descent, we rewrite $F(\mathbf{x})$ as $\sum_{i=1}^{n} F_i(\mathbf{x})$ where $F_i(\mathbf{x})$ is the ith record in the data set. The problem is formulated such that each update needs a single observation from the data. Thus with each observation in the data, we update the sequence until convergence with $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla F_i(\mathbf{x}_t)$ where η_t is called the learning rate. The stochastic gradient descent algorithm randomizes F_i then updates $\mathbf{x}$ for each i. However, there are two issue with this methodology. First, allowing the gradient to be evaluated with one data point creates instability in the direction and magnitude of the update. Second, an incorrectly selected learning rate can cause the algorithm to converge slowly. We solve the the first issue with batching, that is we take the average update of several data points. The second issue is solved using an adaptive learning rate. We use AdaGrad [84] because of its ease of implementation. AdaGrad tracks the diagonal of the outer product of the subgradient sequence to change the learning rate per parameter. It is a specialization of the standard gradient descent that gives frequent features lower learning rates.

In stochastic coordinate descent, we take vertical slices of data to update our solution instead of horizontal slices of data in stochastic gradient descent. We update for each coordinate in $\mathbf{x} \in \mathbb{R}^d$. With each iteration, we create a sequence that updates only one coordinate with $\mathbf{x}_{j,t+1} = \mathbf{x}_{j,t} - \alpha_t(\nabla F(\mathbf{x}_t))_j$ where α_t is the step size. The algorithm randomly updates the coordinates of the solution until convergence.

In Section 4.1, we derive the algorithms that implement stochastic coordinate descent and gradient descent to fit logistic regression models. In Section 4.2, we look at the performance of the algorithm with a comparison to GPU acceleration. For in-depth linear regression examples, see Appendix D.

4.1 Logistic Regression Derivation

In this section, our focus is on the algorithms for logistic regression. In particular, the model assumes that we have data, $\mathbf{X}$, and response, $\mathbf{Y}$. The data is normalized such that $\sum_{i=1}^{n} x_{ij} = 0$ and $\frac{1}{n}\sum_{i=1}^{n} x_{ij}^2 = 1$, so the probability of success is $P(Y_i = 1|\mathbf{x}_i) = (1 + e^{-\mathbf{x}_i^T \boldsymbol{\beta}})^{-1}$ and the probability of failure is $P(Y_i = 0|\mathbf{x}_i) = (1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}})^{-1}$. We have the following likelihood function that we wish to maximize as well as the scaled log likelihood function which is equivalent because the log function is monotonic:

$$\begin{aligned}\underset{\boldsymbol{\beta}}{\operatorname{argmax}}\, L(\boldsymbol{\beta}|\mathbf{X}) &= \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i} \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1}^{N} \log(p_i^{y_i}(1-p_i)^{1-y_i}). \end{aligned} \tag{4.1}$$

We frame the problem as a minimization problem by taking the negative of the scaled log likelihood and adding the penalty, we have the following:

$$\begin{aligned}\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, H(\boldsymbol{\beta}) &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, F(\boldsymbol{\beta}) + \lambda \left\|\boldsymbol{\beta}\right\|_{L_1} \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} y_i \log\left(\frac{p_i}{1-p_i}\right) + \log(1-p_i) + \lambda \sum_{j=1}^{d} |\beta_j|\,. \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} y_i\mu_i + \log(1-p_i) + \lambda \sum_{j=1}^{d} |\beta_j|\,, \end{aligned} \tag{4.2}$$

where $p_i = P(Y_i = 1|\mathbf{x}_i)$ and $\mu_i = \mathbf{x}_i^T\boldsymbol{\beta}$. Moreover we have the following useful terms:

$$\begin{aligned}\frac{\partial p}{\partial \beta_k} &= \frac{e^{\mu}x_k}{(1+e^{\mu})^2} = p(1-p)x_k \\ \frac{\partial(1-p)}{\partial \beta_k} &= -p(1-p)x_k. \end{aligned} \tag{4.3}$$

With these terms, we find the gradient and Hessian of the likelihood function. For the gradient, we use the definition of the subdifferential for the penalty term whereas the Hessian does not need a special case.

$$\begin{aligned}
(\nabla H(\boldsymbol{\beta}))_k &= (\nabla F(\boldsymbol{\beta}))_k + \lambda \frac{\partial}{\partial \beta_k} \|\boldsymbol{\beta}\|_{L_1} \\
&= -\frac{1}{N}\sum_{i=1}^{N} x_{ik}(y_i - p_i) + \lambda \begin{cases} \operatorname{sign}(\beta_k) & \text{if } \beta_k \neq 0 \\ [-1,1] & \text{if } \beta_k = 0 \end{cases} \\
(\nabla^2 H(\boldsymbol{\beta}))_{ij} &= (\nabla^2 F(\boldsymbol{\beta}))_{ij} \\
&= \frac{1}{N}\sum_{k=1}^{N} -x_{ki}x_{kj}p_k(1-p_k) \qquad (4.4)
\end{aligned}$$

Our goal is to ensure that for each iteration we decrease the function $F(\boldsymbol{\beta})$ in search of the minimum. In order to ensure this, we optimally use the step size in the stochastic coordinate descent iteration. That is, we use the condition of Shalev-Shwartz and Tewari [85] that $F(\boldsymbol{\beta} + \delta\beta_j \mathbf{e}_j) \leq F(\boldsymbol{\beta}) + \delta\beta_j(\nabla F(\boldsymbol{\beta}))_j + \frac{1}{2}\alpha(\delta\beta_j)^2$. This condition ensures that on the Tth iteration $\boldsymbol{\beta}^{(T)}$ is close to the minimizer, $\boldsymbol{\beta}^*$, $\left|E[F(\boldsymbol{\beta}^{(T)})] - F(\boldsymbol{\beta}^*)\right| < C/T$ for some C. Following Taylor's Theorem, we need to select α such that $\delta\beta_j \mathbf{e}_j' \nabla^2 F(\boldsymbol{\beta}) \delta\beta_j \mathbf{e}_j \leq \alpha(\delta\beta_j)^2$. When we find the max $\nabla^2 F(\boldsymbol{\beta})_{jj} = \frac{1}{N}\sum_{i=1}^{N} x_{ij}^2 p_i(1-p_i)$. We have $p_i(1-p_i)$ has a maximum of 1/4 at $p_i = 1/2$. Therefore $\alpha = \frac{1}{4}\frac{1}{N}\sum_{i=1}^{N} x_{ij}^2 = \frac{1}{4}$ from the regularization.

After dealing with the likelihood part of the function we address the penalty. The ℓ^1 penalty or LASSO term is carefully handled. The derivative of $|\beta_j|$ changes with the sign of β_j and includes all subderivatives when $\beta_j = 0$. When updating $\boldsymbol{\beta}_t$, we have to consider the sign changes of the components, $\beta_{j,t}$ and $\beta_{j,t+1}$. Suppose that $\beta_{j,t} > 0$, then $\beta_{j,t+1} = \beta_{j,t} - \gamma_t(\nabla H(\boldsymbol{\beta}_t)) > 0$ is correct so long as $\beta_{j,t+1}$ remains positive. Thus we have the condition that $\beta_{j,t} - \gamma_t \nabla F(\boldsymbol{\beta}_t) > \gamma_t \lambda$. Similarly $\beta_{j,t} < 0$ gives rise to the condition that $\beta_{j,t} - \gamma_t \nabla F(\boldsymbol{\beta}_t) < -\gamma_t\lambda$. Thus we are left with $\beta_{j,t} - \gamma_t \nabla F(\boldsymbol{\beta}_t) \in [-\gamma_t\lambda, \gamma_t\lambda]$ when $\beta_{j,t+1} = 0$. These three cases can be exactly represented by the soft-thresholding function:

$$s_\tau(w) = \operatorname{sign}(w)\max(0, |w| - \tau). \qquad (4.5)$$

For stochastic gradient descent, we have $\nabla F_i(\boldsymbol{\beta}_t) = -(y_i - p_i)\mathbf{x}_i$ and η_t is the learning rate. Hence our naive update step is $\boldsymbol{\beta}_{t+1} = s_{\eta_t\lambda}\left(\boldsymbol{\beta}_t + \eta_t(y_i - p_i)\mathbf{x}_i\right)$. We use the algorithm AdaGrad introduced by Duchi, Hazan, and Singer [84] to provide and adaptive learning rate. While AdaGrad has been improved upon by algorithms such as AdaDelta [86] and Adam [87], AdaGrad is simpler to implement. In addition, we batch our update to add stability. This means that we take a random subset of indices of size B and average the gradient. For the largest batch size, $B = N$, we have gradient descent which is slow. For the smallest batch size $B = 1$ is too sensitive to outliers. A large batch size approximates the gradient for a fast update. Furthermore we choose a common learning rate, $\eta_t = 1 \cdot t^{-1/2}$. Algorithm 4.1 represents our version of stochastic gradient descent.

```
Result: $\operatorname{argmin}_{\boldsymbol{\beta}} H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$
while not converged do
    Choose $I_t \subset [N]$ uniformly at random with $|I_t| = B$.
    Let $G_{t+1,ii} \leftarrow G_{t,ii} + (\frac{1}{B}\sum_{i \in I_t}((y_i - p_i)\mathbf{x}_i))_i^2$.
    Let $\eta_t \leftarrow t^{-1/2}/\sqrt{\epsilon + G_{t+1,ii}}$.
    Let $\boldsymbol{\beta}_{t+1} \leftarrow s_{\eta_t\lambda}\left(\boldsymbol{\beta}_t + \eta_t\frac{1}{B}\sum_{i \in I_t}((y_i - p_i)\mathbf{x}_i)\right)$.
end
```

Algorithm 4.1: AdaGrad Stochastic gradient descent with batching for logistic regression with ℓ^1 penalty.

For coordinate descent, we have $(\nabla F(\boldsymbol{\beta}_t))_k = -\frac{1}{N}\sum_{i=1}^{N} x_{ik}(y_i - p_i)$ and the step size $\alpha_t = \frac{1}{4}$ is sufficient. The update step is $\boldsymbol{\beta}_{k,t+1} = s_{\alpha_t\lambda}\left(\boldsymbol{\beta}_{k,t} + \alpha_t\frac{1}{N}\sum_{i=1}^{n} x_{ik}(y_i - p_i)\right)$. Shalev-Shwartz and Tewari [85] introduce a stochastic coordinate descent with convergence results. Algorithm 4.2 is our implementation.

```
Result: $\operatorname{argmin}_{\boldsymbol{\beta}} H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$
while not converged do
    Chose $k \in \{1, \ldots, d\}$ uniformly at random.
    Let $\beta_{k,t+1} \leftarrow s_{\alpha_t\lambda}\left(\boldsymbol{\beta}_{k,t} + \alpha_t(\frac{1}{N}\sum_{i=1}^{N} x_{ik}(y_i - p_i))\right)$.
end
```

Algorithm 4.2: Stochastic coordinate descent for logistic regression with ℓ^1 penalty.

In order to implement these methods in parallel, we made several attempts using existing literature. For SGD, we attempted the Hogwild approach of Recht, etc. [88] for asynchronous updates to the solution for coordinate descent. The asynchronous updates means that every time a thread starts an update it uses the latest version of $\boldsymbol{\beta}$. However, this implementation failed to converge. Instead we exploit parallel structures in Algorithm 4.3. For each processor, we compute the gradient of a random row and accumulate the gradients. In this algorithm, the number of threads equals the batch size. After we synchronize the threads with a wall, for the d components of $\boldsymbol{\beta}$ we update the outer product of the gradients, the step size, and combine it to update the parameter.

```
Result: $\text{argmin}_{\boldsymbol{\beta}} H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$
while not converged do
    do in parallel
        Choose $i \in \{1, \ldots, N\}$ uniformly at random.
        Compute $p_i$.
        Atomically update $W_j \leftarrow W_j + (y_i - p_i)x_{ij}$ for all $j$.
    end
    Synchronize.
    do in parallel
        Choose $j \in \{1, \ldots, d\}$.
        Let $G_{t+1,jj} \leftarrow G_{t,jj} + \left(\frac{1}{B} W_j\right)^2$.
        Let $\eta_t \leftarrow t^{-1/2} / \sqrt{\epsilon + G_{t+1,ii}}$.
        Let $\boldsymbol{\beta}_{j,t+1} \leftarrow s_{\eta_t \lambda}\left(\boldsymbol{\beta}_{j,t} + \eta_t \frac{1}{B} W_j\right)$.
    end
end
```
Algorithm 4.3: Parallel stochastic gradient descent with batching for logistic regression with ℓ^1 penalty.

We had similar problems with SCD. The shotgun, a parallelized shooting approach, by Bradley et al. [89] was difficult to implement due to the duplicate features. Algorithm 4.4 instead uses the same parallel structure but does not use the same update or duplicate features. We update μ_t using a parallelized matrix vector multiplication. Using μ_t, we perform several updates for random column vectors.

```
Result: $\text{argmin}_{\boldsymbol{\beta}} H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$
while not converged do
    do in parallel
        $\mu_t \leftarrow \mathbf{X}\boldsymbol{\beta}_t$.
    end
    do in parallel
        Choose $k \in \{1, \ldots, d\}$ uniformly at random.
        Let $\beta_{k,t+1} \leftarrow s_{\alpha_t \lambda}\left(\boldsymbol{\beta}_{k,t} + \alpha_t\left(\frac{1}{N}\sum_{i=1}^{N} x_{ik}(y_i - p_i)\right)\right)$.
    end
end
```
Algorithm 4.4: Parallel stochastic coordinate descent with batching for logistic regression with ℓ^1 penalty.

Both methods that we implemented, while effective, are general purpose algorithms. In contrast, the glmnet algorithm of Friedman et al. [90] is designed for logistic regression with penalty. The algorithm uses the Taylor expansion to find a weighted least squares approximation. Moreover, we need to solve for several λ with nearby solutions. Therefore each solution for λ can warm start the algorithm. We provide the derivation of the glmnet update from the following Taylor expansion:

$$F(\boldsymbol{\beta}) = F(\tilde{\boldsymbol{\beta}}) + (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^T \nabla F(\tilde{\boldsymbol{\beta}}) + \frac{1}{2}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^T \nabla^2 F(\tilde{\boldsymbol{\beta}})(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) + \text{Remainder}. \tag{4.6}$$

$$\begin{aligned}
&\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, F(\tilde{\boldsymbol{\beta}}) + (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^T \nabla F(\tilde{\boldsymbol{\beta}}) + \frac{1}{2}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^T \nabla^2 F(\tilde{\boldsymbol{\beta}})(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) \\
&\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \frac{1}{N} \sum_{i=1}^{N} y_i \tilde{\mu}_i + \log(1 - \tilde{p}_i) - (y_i - \tilde{p}_i)(\mu_i - \tilde{\mu}_i) - \frac{1}{2}\tilde{p}_i(1 - \tilde{p}_i)(\mu_i - \tilde{\mu}_i)^2 \\
&\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \frac{-1}{2N} \sum_{i=1}^{N} \tilde{p}_i(1 - \tilde{p}_i) \left[\left(\mu_i - \tilde{\mu}_i - \frac{y_i - \tilde{p}_i}{\tilde{p}_i(1 - \tilde{p}_i)} \right)^2 + \frac{-2y_i\tilde{\mu}_i - 2\log(1 - \tilde{p}_i)}{\tilde{p}_i(1 - \tilde{p}_i)} - \left(\frac{y - \tilde{p}_i}{\tilde{p}_i(1 - \tilde{p}_i)} \right)^2 \right]
\end{aligned} \tag{4.7}$$

The first term is a quadratic and the remainder we will ignore. We can then obtain a form for the weighted least squares problem

$$\begin{aligned}
\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \ell_Q(\boldsymbol{\beta}) &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \frac{-1}{2N} \sum_{i=1}^{N} \tilde{p}_i(1 - \tilde{p}_i) \left(\mu_i - \tilde{\mu}_i - \frac{y_i - \tilde{p}_i}{\tilde{p}_i(1 - \tilde{p}_i)} \right)^2 \\
\frac{\partial \ell_Q(\boldsymbol{\beta})}{\partial \beta_j} &= -\frac{1}{N} \sum_{i=1}^{N} \tilde{p}_i(1 - \tilde{p}_i) \left(\mu_i - \tilde{\mu}_i - \frac{y_i - \tilde{p}_i}{\tilde{p}_i(1 - \tilde{p}_i)} \right) x_{ij}
\end{aligned} \tag{4.8}$$

Thus we have

$$\beta_j = \frac{s_\lambda(\frac{1}{N}\sum_{i=1}^{N} \tilde{p}_i(1 - \tilde{p}_i) x_{ij} (\sum_{k \neq j} x_{ik}\beta_k - \tilde{\mu}_i - \frac{y_i - \tilde{p}_i}{\tilde{p}_i(1 - \tilde{p}_i)}))}{\frac{1}{N}\sum_{i=1}^{N} \tilde{p}_i(1 - \tilde{p}_i) x_{ij}^2} \tag{4.9}$$

Friedman et al. [90] proceed to describe the glmnet algorithm as the following:

To add parallelism, we split the algorithm into two parts similar to SCD. The first part is to update the mean in parallel, and the second part is to use the common update for each coordinate in parallel. We have the following parallel glmnet Algorithm 4.6.

```
Result: argmin_β $H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$.
while not converged do
    Let $\tilde{\mu}_i = \mathbf{x}_i^T \boldsymbol{\beta}$.
    for $j = 1 \ldots d$ do
        $\beta_j \leftarrow \frac{s_\lambda(\frac{1}{N}\sum_{i=1}^N \tilde{p}_i(1-\tilde{p}_i)x_{ij}(\sum_{k\neq j} x_{ik}\beta_k - \tilde{\mu}_i - \frac{y_i-\tilde{p}_i}{\tilde{p}_i(1-\tilde{p}_i)}))}{\frac{1}{N}\sum_{i=1}^N \tilde{p}_i(1-\tilde{p}_i)x_{ij}^2}$
    end
end
```

Algorithm 4.5: glmnet for logistic regression with ℓ^1 penalty.

```
Result: argmin_β $H(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^d$.
while not converged do
    do in parallel
        $\tilde{\mu}_i \leftarrow \mathbf{X}\boldsymbol{\beta}$.
    end
    do in parallel
        $\beta_j \leftarrow \frac{s_\lambda(\frac{1}{N}\sum_{i=1}^N \tilde{p}_i(1-\tilde{p}_i)x_{ij}(\sum_{k\neq j} x_{ik}\beta_k - \tilde{\mu}_i - \frac{y_i-\tilde{p}_i}{\tilde{p}_i(1-\tilde{p}_i)}))}{\frac{1}{N}\sum_{i=1}^N \tilde{p}_i(1-\tilde{p}_i)x_{ij}^2}$
    end
end
```

Algorithm 4.6: Parallel glmnet for logistic regression with ℓ^1 penalty.

4.2 Performance

In order to test the performance of the algorithms for logistic regression, we create synthetic simulations. Synthetic simulations allow us to create a test data set for a logistic regression. The test data set is limited by allowable matrix allocation sizes in R. We create a matrix, $\mathbf{X}$, of size $n \times p$ from the standard normal distribution. We create covariates, $\boldsymbol{\beta}$, of size p from the *Beta* distribution and set m of the coefficients as zero. We choose five testing scenarios found in Tables 4.1, 4.2, and 4.3. Note that from Section 3, we could not use vectors over size 10^9 thus our design matrix does not exceed that size as well.

A key consideration is the convergence condition for the algorithms as we do not have the

Case	n	p	m
1	10^4	10	2
2	$5 * 10^4$	50	10
3	10^5	100	20
4	$5 * 10^5$	500	100
5	10^6	1000	200

Table 4.1: Parameters for gradient descent algorithms.

Case	n	p	m
1	$4 \cdot 10^3$	100	20
2	$2 \cdot 10^4$	500	100
3	$4 \cdot 10^4$	1000	200
4	$2 \cdot 10^4$	5000	1000
5	$2 \cdot 10^5$	10^4	2000

Table 4.2: Parameters for coordinate descent algorithms.

Case	n	p	m
1	10^3	100	20
2	$5 * 10^3$	500	100
3	10^4	1000	200
4	$5 * 10^4$	5000	1000
5	10^5	10^4	2000

Table 4.3: Parameters for glmnet algorithms.

true solution in general. We chose a stopping condition such that the max coordinate wise update is less than an error tolerance or $\left\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t\right\|_\infty < 10^{-4}$. However, when one coordinate at a time is updated this condition is too aggressive as all but one coordinate is the same value. For SCD and glmnet, we track the last update to $\boldsymbol{\beta}_j$ for each coordinate and stop when all are less than our tolerance. For example, set $w_j = 1$ for all j, when j is updated set $w_j = |\beta_{j,t+1} - \beta_{j,t}|$. Therefore our stopping condition is $\|w_j\|_\infty < 10^{-4}$. We check this condition before the common mean $\boldsymbol{\mu}$ is computed.

We ran the algorithms on a single core CPU with a NVIDIA V100 GPU. We set the batch size and the number of processors to be $B = P = 4096$. Furthermore, we use 8 blocks of 512 threads to achieve our 4096 parallel processors. For each case, we simulated the data, selected a λ, and ran the relevant algorithms 5 times. For SGD, we chose a simulation that emphasizes a larger n than p as the parallelism is row wise. We wrote Algorithm 4.1 in R and C and implemented Algorith 4.3 in CUDA. Our results are in Figure 4.1. The results show for all problem sizes that parallelism improves the time to convergence. However, the performance improvement is not the same order of magnitude of P. This is caused by the simplicity of the gradient update relative to the synchronization and other overhead. A more complex update would yield better results. This issue carries over the SCD and glmnet algorithms. We select design matrix sizes that take advantage of the column wise parallelism. For SCD, we implement Algorithm 4.2 in R and C and Algorithm 4.4 in CUDA. When we used single threads, the largest runs failed to complete in the maximum server time. This means the stopping condition requires every column index to be iterated which takes more time when randomly selecting columns. We replaced the single core SCD that computes the mean and iterates over all columns. Figure 4.2, shows that the R implementation is still slow and we had to drop the largest case. We see the parallelism shows improvement after accounting for overhead. For glmnet, we implement Algorithm 4.5 in R and C and Algorithm 4.6 in CUDA. Figure 4.3, shows the overhead of parallelism slowing down the algorithm for smaller sizes. The uneven scaling with the design matrix size is indicative of the randomness of our convergence criteria with the smaller ratio of $n : p$ in the design matrix.

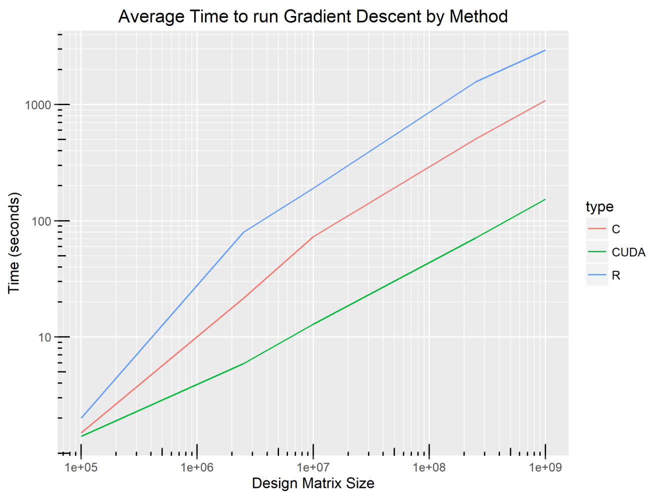


Figure 4.1: The graph shows the timings of three implementations of stochastic gradient descent for logistic regression with ℓ^1 penalty. The R and C implementations are written for a single processor and the CUDA implementation is written using a parallel algorithm.

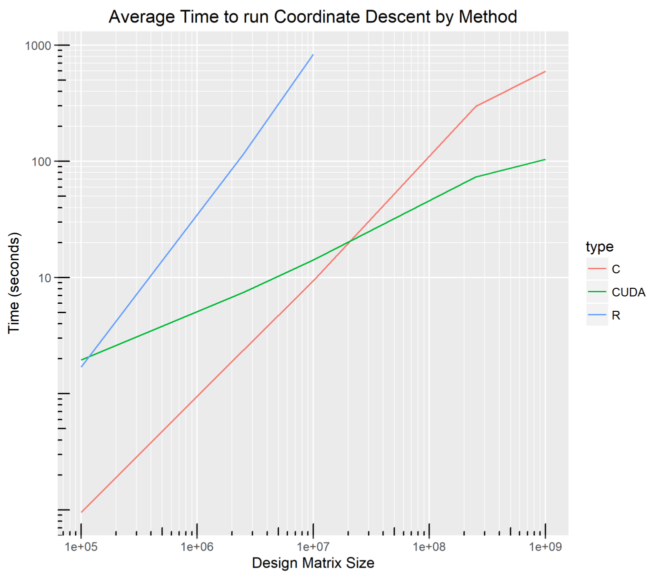


Figure 4.2: The graph shows the timings of three implementations of stochastic coordinate descent for logistic regression with ℓ^1 penalty. The R and C implementations are written for a single processor and the CUDA implementation is written using a parallel algorithm.

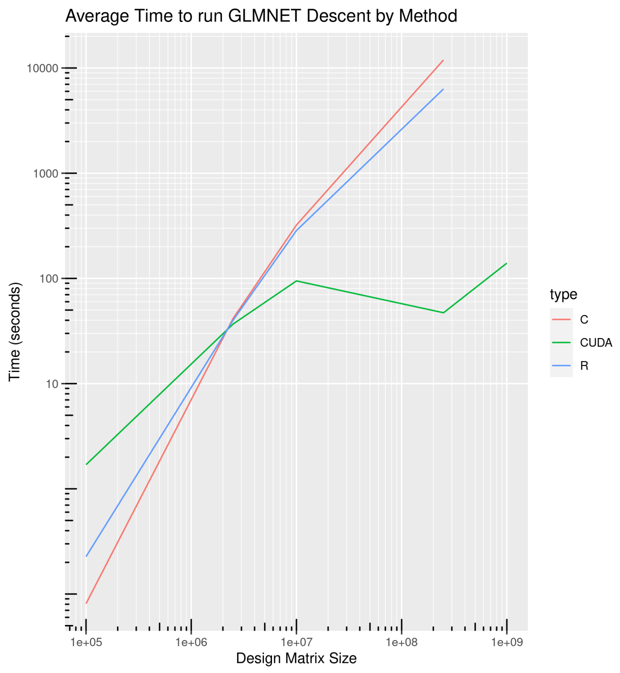


Figure 4.3: The graph shows the timings of three implementations of glmnet for logistic regression with ℓ^1 penalty. The R and C implementations are written for a single processor and the CUDA implementation is written using a parallel algorithm.

Chapter 5

Horseshoe Probit

In this chapter, we explore the benefits of GPU acceleration using Clemson's High Performance Cluster in a Bayesian statistical model. In particular, we benchmark the case study by Terenin, Dong, and Draper [67]. The case study revolves around a Bayesian probit regression with a horseshoe prior. The horseshoe prior was introduced by Carvalho, Polson, and Scott [91]. The binary response from Albert and Chib [55] and the horseshoe prior was used for a GPU-accelerated case study by Terenin, Dong, and Draper [67]. In Section 5.1, we establish the model and derive the posterior sampling algorithm. In Section 5.2, we build probability distributions from basic functions because the CUDA SDK has limit random number generating functionality. In Section 5.4, we discuss two programming features: randomness and streaming. These features have been used in other sections but a thorough discussion is necessary in this example. A portion of the definitions and theorems used in this chapter can be found in Appendix E. The code used in this chapter is found in Appendix F.

5.1 Model

This section will introduce the probit model with the horseshoe prior. We will further decompose the horseshoe prior into inverse gamma distributions to simplify our conversion to GPU acceleration. Lastly, we present the likelihood function. The probit model proposed by Terenin, Dong and Draper [67] is

$$\begin{aligned} y_i|z_i &= round[\Phi(z_i)], \\ z_i|x_i, \boldsymbol{\beta} &\sim N(x_i\boldsymbol{\beta}, 1), \\ z_i &\perp z_{-i}|\boldsymbol{\beta} \end{aligned} \tag{5.1}$$

where the priors from Carvalho, Polson and Scott [91] are

$$\begin{aligned} \beta_j|\lambda_j, \tau &\sim N(0, \lambda_j^2\tau^2), \\ \lambda_j &\sim C^+(0,1), \\ \tau &\sim C^+(0,1). \end{aligned} \tag{5.2}$$

The parameter estimation for this problem is easier than our earlier work in Chapter 2, but not trivial. In order to make a Gibbs sampler GPU ready, we augment the prior distributions. The Cauchy distribution is difficult to directly sample from so we further decompose $\lambda_j^2|\nu_j \sim IG(1/2, 1/\nu_j)$ and $\nu_j \sim IG(1/2, 1)$ and $\tau^2|\epsilon \sim IG(1/2, 1/\epsilon)$ and $\epsilon \sim IG(1/2, 1)$. The decomposition is outlined using the theorems in Appendix E.

5.1.1 Posterior Computation

Now that we have established our model along with the prior, we write the likelihood then the posterior and conditional posterior distributions. We are able to compute conditional posterior distributions for every parameter. We use a Gibbs sampler and avoid any Metropolis-Hasting steps since we have conditional posterior distributions for every parameter.

First we have the following augmented likelihood function for probit regression from Albert and Chib [55]:

$$\pi(y|Z, \boldsymbol{\beta}) = \prod_{i=1}^{N} \{1(Z_i > 0)1(y_i = 1) + 1(Z_i \leq 0)1(y_i = 0)\}\phi(Z_i|x_i\boldsymbol{\beta}, 1). \tag{5.3}$$

We combine the likelihood function and the priors to get the full the posterior as follows:

$$\pi(Z, \boldsymbol{\beta}|y) \propto \pi(\boldsymbol{\beta}) \prod_{i=1}^{N} \{1(Z_i > 0)1(y_i = 1) + 1(Z_i \leq 0)1(y_i = 0)\}\phi(Z_i|x_i\boldsymbol{\beta}, 1)$$

$$= \prod_{j=1}^{p} N(\beta_j|0,\lambda_j^2\tau^2) IG(\lambda_j^2|1/2,\nu_j^{-1}) IG(\nu_j|1/2,1) IG(\tau^2|1/2,\epsilon^{-1}) IG(\epsilon|1/2,1)$$
$$\times \prod_{i=1}^{N} \{1(Z_i>0)1(y_i=1)+1(Z_i\le 0)1(y_i=0)\} N(z_i|x_i\boldsymbol{\beta},1,y_i) \tag{5.4}$$

From the full posterior, we derive the full conditional posteriors as follows:

$$\begin{aligned}
&z_i \sim TN(x_i\boldsymbol{\beta},1,y_i)\\
&\tau^{-2} \sim IG\left(\frac{p+1}{2},\frac{1}{\epsilon}+\frac{1}{2}\sum\frac{\beta_j^2}{\lambda_j^2}\right)\\
&\epsilon \sim IG(1,\frac{1}{\tau^2}+1)\\
&\lambda_j^2 \sim IG(1,\frac{\beta_j^2}{2\tau^2}+\frac{1}{\nu_i})\\
&\nu_j \sim IG(1,\frac{1}{\lambda_j^2}+1)\\
&\beta \sim N(\Sigma X^T z,\Sigma)
\end{aligned} \tag{5.5}$$

where $\Sigma = \left(X^TX+\tau^{-2}diag(\lambda_1^{-2},\ldots,\lambda_p^{-2})\right)^{-1}$.

For inference, we create a Gibbs sampler with full conditional posteriors. From the above list of conditional posterior distributions, we see that we need to sample from a normal distribution, truncated normal distribution, exponential and inverse gamma distribution. Notice that many of the posterior distributions are listed as inverse gamma. When the shape is one for an inverse gamma distribution, it is equivalent to an exponential distribution. In the next section, we review sampling methods for those distributions to complete the Markov chain Monte Carlo method.

5.2 Distributions

From the previous section, we need to sample from a normal distribution, truncated normal distribution, and inverse gamma distribution. The CUDA toolkit offers a library for random number generation but not all of the aforementioned distributions. In exchange for performance and parallelism, there are several of the usual trade-offs. The programmer is responsible for a litany of tasks with respect to random number generation. These tasks include: selecting the random number generator, initializing and tracking the starting states for each thread, and transforming the basic distributions. The basic distributions available are: the standard uniform, standard normal, log normal and Poisson distributions. Therefore we spend some time building the distributions we need. We choose methods that extend the available distributions to our desired distributions utilizing the built-in sampling methods. When we sample from an exponential distribution, a gamma distribution, or a truncated normal distribution, they all must be derived from a sampling method based on a uniform or normal distribution.

5.2.1 Exponential

We begin with the exponential distribution. The following Lemma 5.2.1 gives us that the exponential distribution is a simple transformation of the uniform distribution:

Lemma 5.2.1. *If* $U \sim U(0,1)$*, then* $-\log U/\lambda \sim Exp(\lambda)$*.*

Proof. We have that $P(-\log U/\lambda \leq u) = P(U \geq e^{-\lambda u}) = 1 - F_U(e^{-\lambda u}) = 1 - e^{-\lambda u}$. □

5.2.2 Truncated Normal

In this subsection, we discuss how to build the truncated normal distribution for $(-\infty, 0]$ and $[0, \infty)$. In order to do this, we use part of the efficient sampler from Li and Ghosh [92]. To efficiently sample the truncated normal, we combine three samplers: a normal rejection sampler, a half-normal rejection sampler and a one-sided translated-exponential.

Li and Ghosh [92] method for a truncated normal distribution uses the accept reject algorithm that is outlined in Algorithm 5.1. With the accept reject algorithm, the expected number

```
Result: Sample from distribution $X$ with density $f$ using accept-reject algorithm
repeat
    1) Sample $y$ from $Y$ with density $g$
    2) Sample $u$ from $Unif(0,1)$
until $u < f(y)/Mg(y)$;
return $x$=$y$
```

Algorithm 5.1: Accept reject algorithm.

of iterations is M. Thus we wish to minimize M whenever possible, or else we waste too much time rejecting samples. Minimizing M is the reason there are three components to sampling the truncated normal distribution.

We are interested in sampling from the truncated normal distribution with mean $x\boldsymbol{\beta}$ and standard deviation 1 on $[0,\infty)$. When we negate the mean, we get the distribution for the support $(-\infty, 0]$. Hence we focus only on the problem for positive support.

The accept reject algorithm needs four components: a target distribution, a sample distribution, a uniform sampler and M. We begin by writing the target distribution for the truncated normal as follows:

Definition 1 (Truncated Normal Distribution)**.** *Let the mean and variance be μ and σ^2 respectively. Let $x \in (a,b)$. The density is*

$$TN(x|\mu,\sigma^2,a,b) = \frac{\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} = \frac{\phi(\frac{x-\mu}{\sigma})}{\sigma(\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma}))}$$

and the distribution is

$$F_X(x|\mu,\sigma^2,a,b) = \frac{\Phi(\frac{x-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}.$$

Furthermore, we simplify our target distribution under a linear transform to align our target distribution with the sample distribution. In our case the sample distribution is a standard normal distribution. We have the following lemma to transform the truncated normal:

Lemma 5.2.2. *If X is from $TN(\mu,\sigma^2,0,\infty)$ then $\frac{X-\mu}{\sigma}$ is from $TN(0,1,-\frac{\mu}{\sigma},\infty)$.*

Proof. We have that the density for X is

$$f_X(x) = \frac{\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\int_a^b \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx}. \tag{5.6}$$

Then the density of $(X-\mu)/\sigma$ is $f_X(x\sigma+\mu)\sigma$ or

$$f_{(X-\mu)/\sigma}(x) = \frac{\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}}{\Phi(\frac{\infty-\mu}{\sigma}) - \Phi(\frac{0-\mu}{\sigma})} = \frac{\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}}{1-\Phi(\frac{-\frac{\mu}{\sigma}-0}{1})}. \tag{5.7}$$

□

Let $-\mu/\sigma = a$. Then we optimize $TN(0,1,a,\infty)$ for three cases which gives the three sampling distributions that combine to the truncated normal.

1. If $a < 0$, then let $x \sim N(0,1)$ and let the accept reject sampler reject if $x \in (-\infty, a)$. We have that $TN(0,1,a,\infty) \le \frac{1}{1-\Phi(a)}N(0,1)$. Thus the probability of acceptance is $1/M = 1-\Phi(a)$.

2. If $0 \leq a < a^*$, then let $x \sim N(0,1)$ and take $|x|$ and let the accept reject sampler reject if $|x| < a$. The density of the half normal is $2\phi(x)1(x > 0)$. We have $TN(0,1,a,\infty) \leq \frac{1}{2(1-\Phi(a))}2\phi(x)1(x > 0)$. Thus the probability of acceptance is $1/M = 2(1-\Phi(a))$. Hence when $a > 0$ we use the half normal sampler as it is twice as efficient.

3. If $a^* < a$, then we sample from the translated exponential density $g_Y(y)$. The accept reject sampler condition rejects if $u > e^{-(y-\lambda)^2/2}$. The density for the translated exponential is $\lambda e^{-\lambda(y-a)}1(y > a)$. We can choose $\frac{1}{M} = \sqrt{2\pi}\lambda e^{-\frac{\lambda^2}{2}+\lambda a}(1-\Phi(a))$, giving $f/Mg = e^{-(x-\lambda)^2/2} \leq 1$. Hence for $x > a$ we may maximize $1/M$ with a choice of $\lambda = \frac{a+\sqrt{a^2+4}}{2}$.

We choose a^* for when the acceptance probability of the half normal and the translated exponential are equal. Thus we have $2(1-\Phi(a)) = \sqrt{2\pi}\frac{a+\sqrt{a^2+4}}{2}e^{-\frac{\left(\frac{a+\sqrt{a^2+4}}{2}\right)^2}{2}+a\frac{a+\sqrt{a^2+4}}{2}}(1-\Phi(a))$ or $a^* \approx 0.2570$. With our choice of a^*, it is the optimal value of a^* that minimizes M or the mean number of iterations needed to accept a value. Finally we have the truncated normal distribution in terms of the standard normal and exponential density.

5.2.3 Inverse-Gamma

Finally, we discuss how to build the inverse-gamma distribution. We will use Cheng's algorithm for sampling the gamma distribution from $G(\alpha, 1)$ [2]. We then divide by the rate parameter to get $G(\alpha, \beta)$.

```
**Result:** Sample from Gamma($\alpha, 1$) distribution
**repeat**
    1) Generate a pair of uniform random numbers, $U_1$ and $U_2$.
    2) Set $V = a \log U_1(1-U_1)$ and $X = \alpha e^V$
**until** $b + cV - X < log(U_1^2 U_2)$;
**return** $X$
```

Algorithm 5.2: Sample from the Gamma distribution by an algorithm introduced by Cheng [2].

To generate τ^{-2}, we do not have a high acceptance rate for the method from Cheng [2]. Terenin, Dong and Draper [67] suggests to generate blocks of 32 candidates, check them and use the first available one. This requires local memory between the 32 threads that is faster then the global memory.

Each thread proceeds to generate a candidate, checks if it works, and sets a local flag if true. Once every thread is done producing a candidate, the first thread sees if any of the other threads found a candidate. If an acceptable candidate is found, it exits the process and if not, it restarts the process. We include a wall to block all threads until the threads are finished each iteration. The wall is not necessary if all the threads are on a warp. Hence we generate 32 candidates in the time it takes for one candidate to be produced with a little overhead to sync and to find the correct one.

5.3 Algorithm

The parallel algorithm for the horseshoe probit E.1 is quite complicated because we add parallelism to all matrix operations as well as generate distributions in a parallel manner. For example, we use 8 CUDA functions to sample $\boldsymbol{\beta}$. The operations to copy a matrix, add matrices, multiply a matrix and a vector, dot product, solve dense linear systems, and solve upper triangular systems are built in functions that have been optimized for parallelism. Additionally, we wrote several custom CUDA kernels to independently sample all other distributions.

5.4 Implementation

The previous sections introduced the algorithms to sample from several distributions that are not built by the CUDA SDK. In this section, we learn how to implement them in the CUDA programming environment while taking advantage of the hardware whenever possible. In Subsection 5.4.1, we focus on setting up random number generation and the built in random distributions. In Subsection 5.4.2, we delve deeper into optimally streaming data in and out of the GPU device.

5.4.1 cuRand

Part of the CUDA SDK is the cuRand library. The cuRand functions allow the GPU to generate random numbers. There are a few lines to setup a random number generator. First we must select the type or method of random number generation and the seed. Since the GPU uses SIMD, we have to be careful that when there are multiple threads running that each thread is generating a different random numbers. To do this, we use a state of the random generator which is composed of the seed, the sequence, and the offset in the sequence. This way we are guaranteed that each state will generate a different random sequence. Moreover the states will remain where they are in the sequence and multiple subsequent calls to the state will produce different parts of the random sequence. This allows for it to be initialized once and not repeat itself. We choose to assign each thread a different sequence with no offset.

In the following code, we create the device states, choose our generator, select the random seed and destroy our generator.

```
curandGenerator_t gen;
curandState_t *d_states;
curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_PHILOX4_32_10);
curandSetPseudoRandomGeneratorSeed(gen, 1234ULL); //set to time
...
curandDestroyGenerator(gen);
```

Once the generator, and seed are selected, we can initialize a unique random number state for every thread. The state is used to generate random numbers from the builtin functions that will be unique for every thread.

```
/* Initialize random number generator */
__global__ void init(unsigned int seed, curandState_t *d_states) {
  int idx = threadIdx.x +blockDim.x * blockIdx.x;
  curand_init(seed, idx, 0, &d_states[idx]);
}
```

Later on in the main loop, we generate random samples two different ways. Either we generate random numbers using function calls in custom kernels or using function calls from the host device. When we use function calls in kernels, every kernel must be given a pointer to the state of the random sequence. We use the thread identifier to recall the state with the proper sequence. In the following code snippet, we used the states and thread index to sample z in the kernel. To sample a random number, each thread feeds its thread index into the appropriate state and then that state is used to draw the next random number.

```
 /* Sample normals */
__global__ void sample_z(curandState_t *d_states, int *d_Y, double *d_mu
    ↪ , double *d_z) {
int idx = threadIdx.x + blockDim.x * blockIdx.x;
...
d_s = curand_normal_double(&d_states[idx]);
...
}
```

Alternatively, from the host we can ask for a set of n random samples to be generated on the device. After the random samples are generated we can simply use a pointer to determine where they are in memory to read them later.

```
/* Sample 2p + 1 uniform random variables*/
curandGenerateUniformDouble(gen, d_uniform, p + p + 1);
```

5.4.2 Streaming

For the NVIDIA GPU, transferring data from device and host and floating point calculations in the core are not dependent operations. Therefore we can transfer memory data while other non-blocking processes can be done simultaneously. This process is called streaming.

We set up two streaming handles for this application. The following code sets up and destroys the streams:

```
cudaStream_t stream1;
cudaStream_t stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
...
cudaStreamDestroy(stream1);
cudaStreamDestroy(stream2);
```

We interleave the streams for several operations. For example allocating memory and copying into memory are separate operations. We move several variables onto the device. While we allocate up the memory for one variable, we begin queuing up memory copies.

```
/* Allocate space for tau_square_inverse */
cudaMalloc((void **)&d_tau_square_inverse, sizeof(double));
cudaMemcpyAsync(d_tau_square_inverse, &tau_square_inverse, sizeof(double
    ↪ ), cudaMemcpyHostToDevice,stream1);
```

```
/* Allocate space for v_inverse */
cudaMalloc((void **)&d_v_inverse, p * sizeof(double));
cublasSetVectorAsync(p, sizeof(double), v_inverse, 1, d_v_inverse, 1,
    ↪ stream2);
```

Another instance when streaming improves performance is when retrieving results from the device back to the host. After every update of $\boldsymbol{\beta}^{(t)}$, the results are sent back to the host while one can proceed with another computation. This reduces the memory needed on the device side as one does not have to store every iteration of $\boldsymbol{\beta}^{(t)}$. Additionally, it saves time at the end of the computation to move the entire chunk of memory back to the host.

```
/*Step viii: Compute Beta = Rs + mu */
cublasDaxpy(handle, p, d_one, d_s, 1, d_XTz, 1);
cublasGetVectorAsync(p, sizeof(double), d_XTz, 1, store + t * p , 1,
    ↪ stream1);
```

5.5 Performance

We use the following probit population model to evaluate our estimation methodology:

$$Y_i \sim \text{Bernoulli}(\Phi(x_i'\boldsymbol{\beta})) \tag{5.8}$$

where with x_i is a vector of size d generated from the standard normal, that is $x_{ij} \sim N(0,1)$, and $\boldsymbol{\beta}$ is a vector of size d with mostly zeros. We generate the data $\{(Y_i, X_i)\}$ for $i = 1, \ldots, N$. We choose $n = 10^4$ and $d = 10^3$. We set $\boldsymbol{\beta} = (1.3, 4, -1, 1.6, 5, -2, 0, \ldots, 0)'$.

We perform and retain 100,000 posterior draws. Even with a small example such as this we see significant differences in the languages. See Table 5.1. Although almost ever step uses parallelism, the most significant speed ups are from the Cholesky decomposition, inverting the correlation matrix, and sampling the truncated normal. Both the Cholesky decomposition and the solve are for matrices of size $d \times d$. Since we use a large $d = 10^3$), we can expect parallelism to gain an advantage. Moreover, a significant amount of time in the R solve method is used to check inputs and choose the solve method for every iteration. Whereas we choose the solve method without checks. Lastly, we sample $n = (10^4)$ independent truncated normal distributions. Independence allows us to divide n by the

Language	Min	1-Quartile	Mean	Median	3-Quartile	Max
R	1569.2530	1597.0731	1619.3852	1607.5422	1624.1556	1698.9023
CUDA	147.7432	148.2983	148.5039	148.2989	148.7514	149.4278

Table 5.1: The table shows the timings for five runs of the horseshoe probit MCMC in two implementations. The R implementation is written for a single processor and the CUDA implementation is written using a parallel algorithm. Timings are for 1,000 iterations, 10,000 observations with 100 covariates. All timings are displayed in seconds.

number of threads to perform the work instead of computing them sequentially. In Appendix F, we have the link to the code for our tests.

Chapter 6

Future Directions

There are a couple areas to improve upon for group testing. First, while our estimations are fast, they can be improved upon with GPU acceleration. The measurement for group testing can be switched to included dilution effects. The correlation of the disease can include risk factors. Lastly, there are several opportunities to improve variable selection and estimation.

Our methodology for group testing is a computationally intensive process and takes time to run. Our next steps are to increase its performance using GPU acceleration. In order to use GPU acceleration, we have to identify areas for parallelization. In general there are three areas that we can improve. If sections are independent, we can use threads to compute them in parallel. If we use matrix operations, we can use built in matrix libraries to accelerate the computation. If there is an aggregation function, we have opportunities to parallelize the aggregation.

For Markov chain Monte Carlo methods, every iteration depends on the state of the last iteration. This dependence prevents us from parallelizing the iterations. In addition there are global parameters that need to be estimated at every iteration such as $\mathbf{R}$ and $\boldsymbol{\beta}$ that must be sampled before continuing. Hence we focus on parallelization opportunities on a parameter by parameter basis.

We have three sets of global parameters $\mathbf{R}$, $\boldsymbol{\beta}$ and to a lesser extent the sensitivities and specificities. All global parameters have different opportunities for GPU acceleration. The Metropolis-Hastings steps in sampling $\mathbf{R}$ depend on a sequential process for each ρ_{rs} to be computed. We could simultaneously sample ρ_{rs} for all rs, but we would risk bad mixing and failing to converge to our target distribution. Hence in the case of $\mathbf{R}$ we do not have independent pieces that have parallelization opportunities at a high level. One can only benefit from GPU accelerated matrix operations and computing expensive aggregation functions. For the case of sampling the $\boldsymbol{\beta}$ distribution we simultaneously sample. Therefore, we can only take advantage of matrix operations and aggregation functions required to sample $\boldsymbol{\beta}$. When computing sensitivity and specificity there is an opportunity to compute the disease index and specimen type independently. Moreover if the ordering of the group size does not affect other sensitivities and specificities, that is we do not use a truncated beta distribution with upper and lower bounds, then we can independently compute the specificities and sensitivies by group size. Each instance were we have independence, we can use separate threads to compute in parallel.

At the individual levels, sampling can sometimes offer more opportunities because they are independent along the patient or disease dimension. When sampling γ, we can independently sample each patient. The most complicated sampling occurs when computing the true disease status. To parallelize this step, we have to identify disjoint sets of pools. A disjoint set of pools is where there is a patient can only be present in the pool set. This ensures that when a patients status is updated then it will be available to the other patients in the pool set when their status is updated. In our simulation and case study, we had many disjoint pools sets. Small disjoint sets of pools are a characteristic for our testing schemas. For instance in Dorfman group testing the group pool and the dependent individual pools would be set of pools disjoint from all other sets of pools. Hence we can parallellize easily. In a general setting, a simple computation can be performed before the MCMC to determine the number of threads. We assign pools that have common patients a counting index. For every available thread, we assign the next counting index. Beyond the independence within pools, we can add parallelism on the disease index. For each disease. we can independently compute the true disease status of the patients. This is because we only need to know the mean for the disease to compute the true disease status.

When performing group testing, one has to set a threshold for the classification specimen. Since we are interested in parameter estimation instead of classification, we can capture biomarker levels such as nucleic acid concentrations. This will allow us to use richer information as well as directly address the dilution effect and to prevent any issues with the threshold choice for the pool. The correlation between diseases is assumed to be universal, however there may be additional correlations in sub populations. We could explore was to add a dependence for every patient on the correlations.

Finally, there are several ways to improve estimation performance. During the exploration of this paper we experimented with variable selection. As more information is collected in medical records, more predictors are available to the model. Variable selection can bring forward and help isolate the most important risk factors. Spike and slab variable selection shows some promise but was computationally too expensive during early stages of this work. Additionally, we did not explore interactions or other higher order effects.

Appendices

Appendix A Group Testing Algorithms

Finally, we write the Metropolis-Hastings algorithms for both the probit and logistic distributions. Most of our parameters are updated once per iteration in the Monte Carlo Markov Chain. For each iteration, we use the superscript (t) as in $\widetilde{y}_{id}^{(t)}$ to indicate the tth iteration of $\widetilde{y}_{id}$. The complications of indexing our data structures mean parameters such as $\widetilde{z}_{jd}$ be updated $|\mathcal{P}_j|$ times every iteration. Thus the superscript (t^*) indicates an intermediate iteration between t and $t+1$. Additionally, $\mathbf{R}$ is updated $D(D-1)/2$ times; therefore, we use the (t^k) notation for the intermediate iteration $k \in [D(D-1)/2]$. We use $(\star)$ (star) to indicate the proposal on the (t^k) iteration. We sample ρ_{rs} on a uniform random walk with reflective boundaries -1 and 1 with step size $\delta = 0.05$. We write the probit model algorithm as follows:

1. Initialize $\mathbf{T}_i^{(0)} = \mathbf{0}$ and $\boldsymbol{\beta}^{(0)} = \mathbf{0}$ for the ith patient and set $\mathbf{R}^{(0)} = I_2$.
2. Let $g_{id}(x) = \left\{\prod_{j\in\mathcal{A}_i} \left(S_{e_j:d}^{Z_{jd}} \overline{S}_{e_j:d}^{1-Z_{jd}}\right)^{\mathbb{1}(x+\sum \widetilde{y}_{(-i)d}^{(t^*)})} \left(S_{p_j:d}^{1-Z_{jd}} \overline{S}_{p_j:d}^{Z_{jd}}\right)^{1-\mathbb{1}(x+\sum \widetilde{y}_{(-i)d}^{(t^*)})}\right\} p_{id0}^{1-x}(1-p_{id0})^x$ for $x \in \{0,1\}$.

   For all $d = 1, \dots, D$ and for all $i = 1, \dots, N$:

   (a) Sample $T_{id}^{(t+1)} \sim N(\tilde{\boldsymbol{\mu}}_{id}^{(t)}, \tilde{\Sigma}_{id}^{(t)}) I(T_{id}^{(t+1)} > 0)^{\widetilde{y}_{id}^{(t)}} I(T_{i1}^{(t+1)} \le 0)^{1-\widetilde{y}_{id}^{(t)}}$.

   (b) Sample $\widetilde{Y}_{id}^{(t+1)} \sim \text{Bernoulli}\left[g_{id}(1)/\left\{g_{id}(0) + g_{id}(1)\right\}\right]$.
3. Sample $\boldsymbol{\beta}^{(t+1)} \sim N_Q(\tilde{\boldsymbol{\mu}}^{(t)}, \tilde{\boldsymbol{\Sigma}}^{(t)})$ where $\tilde{\boldsymbol{\Sigma}}^{(t)} = \left\{\boldsymbol{\Sigma}_0^{-1} + \sum_{i=1}^N \mathbf{X}_i'(\mathbf{R}^{(t)})^{-1}\mathbf{X}_i\right\}^{-1}$ and $\tilde{\boldsymbol{\mu}}^{(t+1)} = \tilde{\boldsymbol{\Sigma}}^{(t)}\left\{\sum_{i=1}^N \mathbf{X}_i'(\mathbf{R}^{(t)})^{-1}\mathbf{T}_i^{(t+1)}\right\}$.
4. Sample $\rho_{rs}^{\star}$ from $[\rho_{rs}^{(t)} - \delta, \rho_{rs}^{(t)} + \delta]$. Set $\rho_{rs}^{(t+1)} = \rho_{rs}^{\star}$ with probability

$$\min\left\{1, \frac{\prod_{i=1}^N N(\boldsymbol{\mu}_i^{(t+1)}, \mathbf{R}^{\star})}{\prod_{i=1}^N N(\boldsymbol{\mu}_i^{(t+1)}, \mathbf{R}^{(t^k)})}\right\}.$$

   Otherwise $\rho_{rs}^{(t+1)} = \rho_{rs}^{(t)}$.
5. Go to step 2.

The logit algorithm is similar to the probit algorithm but with extra steps for ϕ and the importance sampling weights. Moreover the importance weights produce not a number results in R. This is caused by an infinite return value from the student-t quantile function. When we encounter an infinite return value we substitute a large positive number.

1. Initialize $\mathbf{T}_i^{(0)} = \mathbf{0}$ and $\boldsymbol{\beta}^{(0)} = \mathbf{0}$, $\phi_i^{(0)} \sim \Gamma(\tilde{\nu}/2, \tilde{\nu}/2)$ for the ith patient and set $\mathbf{R}^{(0)} = I_2$.

2. Sample $\phi_i^{(t+1)} \sim \Gamma\left[(\tilde{\nu}+2)/2, \left\{\tilde{\nu} + \tilde{\sigma}^{-2}(\mathbf{T}_i^{(t+1)} - \boldsymbol{\mu}_i^{(t)})'(\mathbf{R}^{(t)})^{-1}(\mathbf{T}_i^{(t+1)} - \boldsymbol{\mu}_i^{(t)})\right\}/2\right]$ for all i.

3. Let $g_{id}(x) = \left\{\prod_{j\in\mathcal{A}_i}\left(S_{e_j:d}^{Z_{jd}}\overline{S}_{e_j:d}^{1-Z_{jd}}\right)^{\mathbb{1}(x+\sum\tilde{y}_{(-i)d}^{(t^*)})}\left(S_{p_j:d}^{1-Z_{jd}}\overline{S}_{p_j:d}^{Z_{jd}}\right)^{1-\mathbb{1}(x+\sum\tilde{y}_{(-i)d}^{(t^*)})}\right\} p_{id0}^{1-x}(1-p_{id0})^x$ for $x \in \{0,1\}$.

   For all $d = 1,\ldots,D$ and for all $i = 1,\ldots,N$:

   (a) Sample $T_{id}^{(t+1)} \sim N(\tilde{\tilde{\boldsymbol{\mu}}}_{id}^{(t)}, \tilde{\tilde{\Sigma}}_{id}^{(t)})I(T_{id}^{(t+1)} > 0)^{\tilde{y}_{id}^{(t)}} I(T_{i1}^{(t+1)} \leq 0)^{1-\tilde{y}_{id}^{(t)}}$.

   (b) Sample $\widetilde{Y}_{id}^{(t+1)} \sim \text{Bernoulli}\left[g_{id}(1)/\left\{g_{id}(0)+g_{id}(1)\right\}\right]$.

4. Sample $\boldsymbol{\beta}^{(t+1)} \sim N_Q(\tilde{\tilde{\boldsymbol{\mu}}}^{(t)}, \tilde{\tilde{\boldsymbol{\Sigma}}}^{(t)})$ where $\tilde{\tilde{\boldsymbol{\Sigma}}}'^{(t)} = \left\{\boldsymbol{\Sigma}_0^{-1} + \tilde{\sigma}^{-2}\sum_{i=1}^N \phi_i^{(t+1)}\mathbf{X}_i'(\mathbf{R}^{(t)})^{-1}\mathbf{X}_i\right\}^{-1}$ and $\tilde{\tilde{\boldsymbol{\mu}}}^{(t+1)} = \tilde{\tilde{\boldsymbol{\Sigma}}}^{(t)}\left\{\tilde{\sigma}^{-2}\sum_{i=1}^N \phi_i^{(t+1)}\mathbf{X}_i'(\mathbf{R}^{(t)})^{-1}\mathbf{T}_i^{(t+1)}\right\}$.

5. Sample $\rho_{rs}^\star$ from $[\rho_{rs}^{(t)} - \delta, \rho_{rs}^{(t)} + \delta]$. Set $\rho_{rs}^{(t+1)} = \rho_{rs}^\star$ with probability

$$\min\left\{1, \frac{\prod_{i=1}^N N(\boldsymbol{\mu}_i^{(t+1)}, \tilde{\sigma}^2\phi_i^{(t+1)}\mathbf{R}^\star)}{\prod_{i=1}^N N(\boldsymbol{\mu}_i^{(t+1)}, \tilde{\sigma}^2\phi_i^{(t+1)}\mathbf{R}^{(t^k)})}\right\}.$$

   Otherwise $\rho_{rs}^{(t+1)} = \rho_{rs}^{(t)}$.

6. Compute $w^{(t+1)} \propto \prod_{i=1}^N L_{2,\tilde{\nu}}(\mathbf{T}_i^{(t+1)}; \boldsymbol{\mu}_i, \mathbf{R}^{(t+1)})/\mathcal{T}_{2,\tilde{\nu}}(\mathbf{T}_i^{(t+1)}; \boldsymbol{\mu}_i, \tilde{\sigma}^2\mathbf{R}^{(t+1)})$.

7. Go to step 2.

To add the sampling for testing accuracies, we add the following steps for independent test configurations.

1. Compute
$$S_{e(l):d}^{(t+1)} \sim f\left\{\sum_{j\in\mathcal{M}(l)} Z_{jd}\tilde{z}_{jd}^{(t+1)} + TP_{l,d}, \sum_{j\in\mathcal{M}(l)}(1-Z_{jd})\tilde{z}_{jd}^{(t+1)} + FN_{l,d}, 0, 1\right\}$$ for all d and l.

2. Compute
$$S_{p(l):d}^{(t+1)} \sim f\left\{\sum_{j\in\mathcal{M}(l)}(1-Z_{jd})(1-\tilde{z}_{jd}^{(t+1)}) + TN_{l,d}, \sum_{j\in\mathcal{M}(l)} Z_{jd}(1-\tilde{z}_{jd}^{(t+1)}) + FP_{l,d}, 0, 1\right\}$$ for all d and l.

If we want to include a dilution effect, we instead add the following:

1. Compute
$S_{e(l):d}^{(t+1)} \sim f\left\{\sum_{j\in\mathcal{M}(l)} Z_{jd}\widetilde{z}_{jd}^{(t+1)} + TP_{l,d}, \sum_{j\in\mathcal{M}(l)} (1-Z_{jd})\widetilde{z}_{jd}^{(t+1)} + FN_{l,d}, A_{e(l):d}, B_{e(l):d}\right\}$ for all d and l.

2. Compute
$S_{p(l):d}^{(t+1)} \sim f\left\{\sum_{j\in\mathcal{M}(l)} (1-Z_{jd})(1-\widetilde{z}_{jd}^{(t+1)}) + TN_{l,d}, \sum_{j\in\mathcal{M}(l)} Z_{jd}(1-\widetilde{z}_{jd}^{(t+1)}) + FP_{l,d}, A_{p(l):d}, B_{p(l):d}\right\}$
for all d and l.

Appendix B Simulation Data

Scenario		Test Rows and Columns						Re-Test Positive Column				
		NN	PP	NN	NN	NN		NN	PP	NN	NN	NN
	NN	○	○	○	○	○	NN	○	NN	○	○	○
	NN	○	○	○	○	○	NN	○	NN	○	○	○
1	NN	○	○	○	○	○	NN	○	NP	○	○	○
	NN	○	○	○	○	○	NN	○	PN	○	○	○
	NN	○	○	○	○	○	NN	○	NN	○	○	○

Scenario		Test Rows and Columns						Re-Test Intersections				
		NN	PP	NN	NN	NN		NN	PP	NN	NN	NN
	NN	○	○	○	○	○	NN	○	○	○	○	○
	PN	○	○	○	○	○	PN	○	NN	○	○	○
2	NP	○	○	○	○	○	NP	○	NP	○	○	○
	NN	○	○	○	○	○	NN	○	○	○	○	○
	NN	○	○	○	○	○	NN	○	○	○	○	○

Figure B.1: Each table represents a part of the process for testing a square five by five array. Each row represents the different scenarios for the last stage. The first table in each row is the result of testing rows and columns. The results of the row and column are labeled as row and column headers with N and P for negative and positive for each of the two diseases. The second table in each row represents which individuals are tested and their test results are labeled on the interior of the table. In the first scenario, only a column test positive, then the testing procedure dictates that every individual is tested in the column. In the second scenario, a column and two rows test positive, then the testing procedure dictates that every individual in the intersection is tested. In both cases, we classify the remaining individuals as negative for both diseases.

The following tables contain the data for the synthetic calibration and Iowa clinical results.

Parameter		IT	MPT	DT	AT
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	5008	1252	3573.4	3863.6
$\boldsymbol{\beta}_{10} = -2.1$	Bias (CP95)	-0.0529 (0.94)	0.0844 (0.87)	-0.0166 (0.96)	-0.0138 (0.95)
	SSD (ESE)	0.1301 (0.1276)	0.1404 (0.1275)	0.0787 (0.0778)	0.0670 (0.0696)
$\boldsymbol{\beta}_{11} = 0.5$	Bias (CP95)	0.0184 (0.93)	-0.0191 (0.92)	0.0061 (0.96)	0.0069 (0.96)
	SSD (ESE)	0.0637 (0.0619)	0.0721 (0.0703)	0.0435 (0.0445)	0.0396 (0.0404)
$\boldsymbol{\beta}_{12} = 0.5$	Bias (CP95)	0.0289 (0.96)	-0.0044 (0.95)	0.0130 (0.95)	0.0103 (0.95)
	SSD (ESE)	0.1209 (0.1195)	0.1728 (0.1568)	0.0866 (0.0839)	0.0777 (0.0768)
$\boldsymbol{\beta}_{20} = -1.7$	Bias (CP95)	-0.0120 (0.94)	0.0158 (0.94)	-0.0072 (0.94)	-0.0067 (0.92)
	SSD (ESE)	0.0630 (0.0597)	0.0875 (0.0845)	0.0521 (0.0497)	0.0504 (0.0477)
$\boldsymbol{\beta}_{21} = 0.5$	Bias (CP95)	0.0035 (0.95)	0.0018 (0.95)	0.0014 (0.96)	0.0010 (0.95)
	SSD (ESE)	0.0347 (0.0357)	0.0563 (0.0549)	0.0306 (0.0313)	0.0296 (0.0301)
$\boldsymbol{\beta}_{22} = 0.5$	Bias (CP95)	0.0054 (0.93)	0.0095 (0.95)	0.0040 (0.94)	0.0043 (0.92)
	SSD (ESE)	0.0696 (0.0651)	0.1147 (0.1117)	0.0621 (0.0582)	0.0602 (0.0564)
$\rho = 0.3$	Bias (CP95)	-0.0045 (0.94)	-0.0256 (0.94)	-0.0042 (0.94)	-0.0035 (0.95)
	SSD (ESE)	0.0676 (0.0647)	0.0837 (0.0852)	0.0526 (0.0516)	0.0483 (0.0477)

Table B.1: Known assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by probit population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. Master pool testing (MPT), DT and AT uses master pools of size 4.

Parameter		IT	MPT	DT	AT
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	5008	1252	3578.2	3864.1
$\beta_{10} = -2.1$	Bias (CP95)	-0.1548 (0.91)	-1.0792 (0.73)	0.0062 (0.96)	-0.0059 (0.96)
	SSD (ESE)	0.8121 (0.3567)	2.2442 (0.7457)	0.0874 (0.0947)	0.0700 (0.0744)
$\beta_{11} = 0.5$	Bias (CP95)	0.3086 (0.76)	0.4721 (0.58)	0.0053 (0.95)	0.0024 (0.95)
	SSD (ESE)	0.4574 (0.2237)	0.7977 (0.3271)	0.0454 (0.0458)	0.0407 (0.0406)
$\beta_{12} = 0.5$	Bias (CP95)	0.3704 (0.83)	1.0090 (0.69)	0.0079 (0.97)	0.0040 (0.96)
	SSD (ESE)	0.7437 (0.2949)	1.9803 (0.6553)	0.0827 (0.0851)	0.0766 (0.0769)
$\beta_{20} = -1.7$	Bias (CP95)	0.0580 (0.94)	-0.2887 (0.91)	0.0080 (0.97)	-0.0013 (0.96)
	SSD (ESE)	0.1555 (0.1597)	0.4013 (0.3138)	0.0533 (0.0561)	0.0482 (0.0490)
$\beta_{21} = 0.5$	Bias (CP95)	0.1356 (0.79)	0.2536 (0.65)	0.0020 (0.95)	0.0003 (0.93)
	SSD (ESE)	0.0909 (0.1101)	0.2321 (0.1787)	0.0323 (0.0318)	0.0322 (0.0302)
$\beta_{22} = 0.5$	Bias (CP95)	0.1343 (0.88)	0.2946 (0.78)	0.0018 (0.96)	-0.0004 (0.95)
	SSD (ESE)	0.1145 (0.1296)	0.3244 (0.2623)	0.0570 (0.0586)	0.0549 (0.0564)
$\rho = 0.3$	Bias (CP95)	0.2173 (0.70)	0.1689 (0.69)	-0.0106 (0.95)	-0.0071 (0.95)
	SSD (ESE)	0.1369 (0.1463)	0.3227 (0.2008)	0.0543 (0.0526)	0.0500 (0.0483)
$S_{e(1):1} = 0.94$	Bias (CP95)	-0.3728 (0.72)		-0.0218 (0.97)	-0.0051 (0.95)
	SSD (ESE)	0.1135 (0.1734)		0.0345 (0.0442)	0.0233 (0.0240)
$S_{e(2):1} = 0.90$	Bias (CP95)		-0.3165 (0.76)	-0.0142 (0.96)	-0.0040 (0.94)
	SSD (ESE)		0.1596 (0.1393)	0.0454 (0.0523)	0.0233 (0.0233)
$S_{e(1):2} = 0.98$	Bias (CP95)	-0.2685 (0.55)		-0.0101 (0.99)	-0.0028 (0.97)
	SSD (ESE)	0.0863 (0.1382)		0.0113 (0.0172)	0.0089 (0.0097)
$S_{e(2):2} = 0.96$	Bias (CP95)		-0.1850 (0.81)	-0.0114 (0.97)	-0.0021 (0.95)
	SSD (ESE)		0.0933 (0.1029)	0.0219 (0.0265)	0.0104 (0.0107)
$S_{p(1):1} = 0.93$	Bias (CP95)	-0.0053 (0.87)		-0.0002 (0.95)	-0.0003 (0.93)
	SSD (ESE)	0.0090 (0.0089)		0.0074 (0.0077)	0.0093 (0.0089)
$S_{p(2):1} = 0.95$	Bias (CP95)		-0.0994 (0.59)	0.0031 (0.97)	0.0002 (0.95)
	SSD (ESE)		0.1595 (0.0380)	0.0113 (0.0125)	0.0070 (0.0071)
$S_{p(1):2} = 0.97$	Bias (CP95)	-0.0059 (0.90)		0.0002 (0.97)	-0.0010 (0.95)
	SSD (ESE)	0.0091 (0.0096)		0.0059 (0.0064)	0.0073 (0.0075)
$S_{p(2):2} = 0.98$	Bias (CP95)		-0.0874 (0.78)	0.0019 (0.98)	-0.0003 (0.97)
	SSD (ESE)		0.0834 (0.0474)	0.0065 (0.0084)	0.0051 (0.0053)

Table B.2: Unknown assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by probit population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. Master pool testing (MPT), DT and AT uses master pools of size 4.

Parameter		DT no order	DT order	AT no order	AT order
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	3573.3	3573.3	3864.4	3864.4
$\beta_{10} = -2.1$	Bias (CP95)	0.0069 (0.96)	0.0135 (0.96)	-0.0053 (0.95)	-0.0028 (0.95)
	SSD (ESE)	0.0922 (0.0950)	0.0920 (0.0952)	0.0735 (0.0742)	0.0735 (0.0743)
$\beta_{11} = 0.5$	Bias (CP95)	0.0041 (0.95)	0.0035 (0.94)	0.0017 (0.95)	0.0017 (0.95)
	SSD (ESE)	0.0462 (0.0457)	0.0463 (0.0456)	0.0398 (0.0405)	0.0398 (0.0405)
$\beta_{12} = 0.5$	Bias (CP95)	0.0102 (0.95)	0.0095 (0.95)	0.0033 (0.95)	0.0032 (0.95)
	SSD (ESE)	0.0861 (0.0851)	0.0856 (0.0849)	0.0762 (0.0767)	0.0762 (0.0769)
$\beta_{20} = -1.7$	Bias (CP95)	0.0025 (0.97)	0.0058 (0.97)	-0.0046 (0.96)	-0.0037 (0.97)
	SSD (ESE)	0.0533 (0.0562)	0.0529 (0.0565)	0.0479 (0.0491)	0.0477 (0.0492)
$\beta_{21} = 0.5$	Bias (CP95)	0.0026 (0.96)	0.0033 (0.96)	0.0013 (0.95)	0.0017 (0.95)
	SSD (ESE)	0.0313 (0.0319)	0.0313 (0.0319)	0.0295 (0.0303)	0.0296 (0.0303)
$\beta_{22} = 0.5$	Bias (CP95)	0.0071 (0.96)	0.0076 (0.96)	0.0041 (0.96)	0.0044 (0.96)
	SSD (ESE)	0.0583 (0.0587)	0.0584 (0.0588)	0.0565 (0.0565)	0.0566 (0.0566)
$\rho = 0.3$	Bias (CP95)	-0.0064 (0.95)	-0.0090 (0.95)	-0.0045 (0.95)	-0.0068 (0.94)
	SSD (ESE)	0.0518 (0.0525)	0.0518 (0.0525)	0.0485 (0.0482)	0.0487 (0.0482)
$S_{e(1):1} = 0.94$	Bias (CP95)	-0.0213 (0.99)	-0.0075 (0.99)	-0.0050 (0.96)	0.0008 (0.97)
	SSD (ESE)	0.0331 (0.0446)	0.0267 (0.0374)	0.0225 (0.0240)	0.0181 (0.0206)
$S_{e(2):1} = 0.90$	Bias (CP95)	-0.0180 (0.97)	-0.0376 (0.93)	-0.0024 (0.94)	-0.0073 (0.95)
	SSD (ESE)	0.0469 (0.0526)	0.0399 (0.0485)	0.0232 (0.0232)	0.0208 (0.0222)
$S_{e(1):2} = 0.98$	Bias (CP95)	-0.0091 (0.99)	-0.0065 (0.99)	-0.0033 (0.96)	-0.0015 (0.96)
	SSD (ESE)	0.0111 (0.0169)	0.0103 (0.0146)	0.0090 (0.0098)	0.0075 (0.0083)
$S_{e(2):2} = 0.96$	Bias (CP95)	-0.0115 (0.96)	-0.0199 (0.95)	-0.0025 (0.93)	-0.0048 (0.94)
	SSD (ESE)	0.0226 (0.0264)	0.0184 (0.0246)	0.0109 (0.0107)	0.0098 (0.0103)
$S_{p(1):1} = 0.93$	Bias (CP95)	0.0006 (0.95)	0.0014 (0.95)	-0.0011 (0.96)	-0.0013 (0.97)
	SSD (ESE)	0.0079 (0.0077)	0.0074 (0.0074)	0.0088 (0.0089)	0.0080 (0.0085)
$S_{p(2):1} = 0.95$	Bias (CP95)	0.0029 (0.98)	0.0018 (0.99)	-0.0004 (0.95)	0.0001 (0.95)
	SSD (ESE)	0.0107 (0.0126)	0.0088 (0.0107)	0.0069 (0.0071)	0.0065 (0.0069)
$S_{p(1):2} = 0.97$	Bias (CP95)	-0.0003 (0.95)	-0.0004 (0.96)	-0.0009 (0.95)	-0.0022 (0.97)
	SSD (ESE)	0.0064 (0.0064)	0.0058 (0.0061)	0.0076 (0.0075)	0.0064 (0.0070)
$S_{p(2):2} = 0.98$	Bias (CP95)	0.0017 (0.99)	0.0026 (0.98)	-0.0001 (0.97)	0.0009 (0.97)
	SSD (ESE)	0.0064 (0.0084)	0.0052 (0.0069)	0.0050 (0.0053)	0.0046 (0.0049)

Table B.3: Unknown assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Assumes ordering for assay accuracies. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by probit population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. DT and AT uses master pools of size 4.

Parameter		IT	MPT	DT	AT
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	5008	1252	3566.3	3860.8
$\beta_{10} = -3.3$	Bias (CP95)	-0.0845 (0.94)	0.1335 (0.82)	-0.0261 (0.94)	-0.0172 (0.91)
	SSD (ESE)	0.2131 (0.2048)	0.2433 (0.2175)	0.1340 (0.1291)	0.1286 (0.1197)
$\beta_{11} = 0.5$	Bias (CP95)	0.0184 (0.93)	-0.0512 (0.91)	0.0033 (0.94)	0.0047 (0.95)
	SSD (ESE)	0.1092 (0.1076)	0.1436 (0.1378)	0.0799 (0.0779)	0.0739 (0.0725)
$\beta_{12} = 0.5$	Bias (CP95)	0.0366 (0.94)	0.0106 (0.93)	0.0148 (0.94)	0.0094 (0.92)
	SSD (ESE)	0.2355 (0.2272)	0.3420 (0.3144)	0.1623 (0.1581)	0.1569 (0.1473)
$\beta_{20} = -2.55$	Bias (CP95)	-0.0142 (0.93)	0.0322 (0.90)	-0.0065 (0.93)	-0.0057 (0.92)
	SSD (ESE)	0.0970 (0.0934)	0.1488 (0.1431)	0.0834 (0.0819)	0.0811 (0.0793)
$\beta_{21} = 0.5$	Bias (CP95)	0.0049 (0.95)	-0.0095 (0.92)	0.0029 (0.95)	0.0031 (0.94)
	SSD (ESE)	0.0552 (0.0571)	0.0985 (0.0985)	0.0508 (0.0516)	0.0511 (0.0502)
$\beta_{22} = 0.5$	Bias (CP95)	0.0048 (0.94)	0.0085 (0.92)	0.0000 (0.95)	-0.0019 (0.95)
	SSD (ESE)	0.1140 (0.1129)	0.2256 (0.2111)	0.1032 (0.1024)	0.0997 (0.0996)
$\rho = 0.3$	Bias (CP95)	0.0023 (0.92)	-0.0301 (0.92)	-0.0009 (0.94)	-0.0023 (0.95)
	SSD (ESE)	0.0706 (0.0680)	0.0907 (0.0887)	0.0546 (0.0529)	0.0494 (0.0488)

Table B.4: Known assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by logistic population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. Master pool testing (MPT), DT and AT uses master pools of size 4.

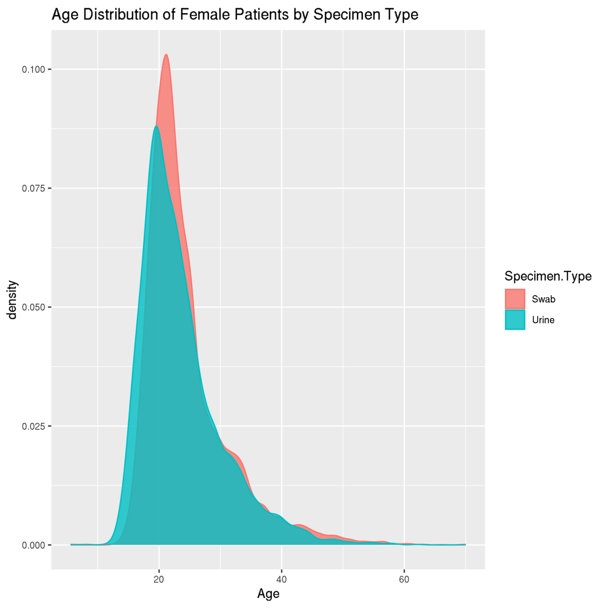
Parameter		IT	MPT	DT	AT
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	5008	1252	3566.5	3856.4
$\beta_{10} = -3.3$	Bias (CP95)	0.1087 (0.88)	-0.2441 (0.58)	0.0588 (0.93)	-0.0088 (0.94)
	SSD (ESE)	1.5659 (0.8944)	6.1371 (1.7864)	0.1626 (0.1849)	0.1360 (0.1321)
$\beta_{11} = 0.5$	Bias (CP95)	0.3321 (0.89)	-0.2027 (0.55)	-0.0036 (0.93)	-0.0013 (0.93)
	SSD (ESE)	0.9085 (0.3071)	3.9308 (1.0766)	0.0754 (0.0793)	0.0730 (0.0719)
$\beta_{12} = 0.5$	Bias (CP95)	0.4684 (0.91)	1.2416 (0.60)	0.0167 (0.95)	0.0080 (0.94)
	SSD (ESE)	1.3174 (0.5461)	6.5378 (2.0586)	0.1640 (0.1601)	0.1545 (0.1463)
$\beta_{20} = -2.55$	Bias (CP95)	0.1405 (0.93)	-2.3682 (0.67)	0.0276 (0.90)	-0.0063 (0.94)
	SSD (ESE)	0.4766 (0.4876)	4.8239 (1.4378)	0.0998 (0.0997)	0.0798 (0.0830)
$\beta_{21} = 0.5$	Bias (CP95)	0.2451 (0.85)	0.9965 (0.44)	-0.0038 (0.94)	-0.0020 (0.92)
	SSD (ESE)	0.1919 (0.2057)	3.0176 (0.7572)	0.0543 (0.0521)	0.0536 (0.0503)
$\beta_{22} = 0.5$	Bias (CP95)	0.2638 (0.85)	2.1825 (0.62)	0.0028 (0.94)	0.0041 (0.95)
	SSD (ESE)	0.2717 (0.2732)	4.7859 (1.4853)	0.1061 (0.1028)	0.0977 (0.0996)
$\rho = 0.3$	Bias (CP95)	0.3739 (0.36)	0.0335 (0.57)	-0.0033 (0.95)	-0.0047 (0.93)
	SSD (ESE)	0.1323 (0.1536)	0.5005 (0.2745)	0.0531 (0.0537)	0.0510 (0.0490)
$S_{e(1):1} = 0.94$	Bias (CP95)	-0.3958 (0.73)		-0.0371 (0.98)	-0.0054 (0.97)
	SSD (ESE)	0.1217 (0.1994)		0.0375 (0.0552)	0.0233 (0.0262)
$S_{e(2):1} = 0.90$	Bias (CP95)		-0.4526 (0.53)	-0.0261 (0.97)	-0.0025 (0.93)
	SSD (ESE)		0.2145 (0.1198)	0.0514 (0.0641)	0.0268 (0.0255)
$S_{e(1):2} = 0.98$	Bias (CP95)	-0.3536 (0.33)		-0.0119 (0.99)	-0.0031 (0.96)
	SSD (ESE)	0.1093 (0.1730)		0.0104 (0.0189)	0.0094 (0.0102)
$S_{e(2):2} = 0.96$	Bias (CP95)		-0.3228 (0.60)	-0.0180 (0.95)	-0.0018 (0.95)
	SSD (ESE)		0.1817 (0.1112)	0.0268 (0.0315)	0.0108 (0.0112)
$S_{p(1):1} = 0.93$	Bias (CP95)	-0.0039 (0.88)		0.0010 (0.96)	-0.0000 (0.95)
	SSD (ESE)	0.0147 (0.0162)		0.0079 (0.0085)	0.0088 (0.0090)
$S_{p(2):1} = 0.95$	Bias (CP95)		-0.1945 (0.47)	0.0065 (0.98)	0.0001 (0.92)
	SSD (ESE)		0.1891 (0.0806)	0.0115 (0.0148)	0.0081 (0.0076)
$S_{p(1):2} = 0.97$	Bias (CP95)	-0.0179 (0.83)		0.0014 (0.95)	-0.0017 (0.94)
	SSD (ESE)	0.0147 (0.0172)		0.0069 (0.0072)	0.0076 (0.0077)
$S_{p(2):2} = 0.98$	Bias (CP95)		-0.2084 (0.37)	0.0026 (0.99)	-0.0003 (0.97)
	SSD (ESE)		0.1518 (0.0639)	0.0057 (0.0090)	0.0051 (0.0056)

Table B.5: Unknown assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by logistic population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. Master pool testing (MPT), DT and AT uses master pools of size 4.

Parameter		DT no order	DT order	AT no order	AT order
	Prevalence	$d_1 = 0.05$ $d_2 = 0.10$			
	Average number of tests	3570.5	3570.5	3861.6	3861.6
$\beta_{10} = -3.3$	Bias (CP95)	0.0665 (0.92)	0.0828 (0.92)	-0.0084 (0.94)	-0.0019 (0.94)
	SSD (ESE)	0.1667 (0.1848)	0.1670 (0.1882)	0.1377 (0.1330)	0.1388 (0.1324)
$\beta_{11} = 0.5$	Bias (CP95)	0.0035 (0.93)	0.0013 (0.93)	0.0019 (0.92)	0.0024 (0.94)
	SSD (ESE)	0.0846 (0.0793)	0.0834 (0.0784)	0.0731 (0.0724)	0.0726 (0.0726)
$\beta_{12} = 0.5$	Bias (CP95)	0.0020 (0.93)	-0.0014 (0.93)	-0.0049 (0.94)	-0.0039 (0.94)
	SSD (ESE)	0.1655 (0.1582)	0.1641 (0.1575)	0.1521 (0.1472)	0.1524 (0.1470)
$\beta_{20} = -2.55$	Bias (CP95)	0.0259 (0.91)	0.0343 (0.90)	-0.0060 (0.91)	-0.0041 (0.91)
	SSD (ESE)	0.0959 (0.1003)	0.0949 (0.1008)	0.0910 (0.0830)	0.0903 (0.0833)
$\beta_{21} = 0.5$	Bias (CP95)	0.0042 (0.92)	0.0044 (0.91)	0.0030 (0.92)	0.0034 (0.94)
	SSD (ESE)	0.0553 (0.0515)	0.0548 (0.0519)	0.0531 (0.0505)	0.0539 (0.0504)
$\beta_{22} = 0.5$	Bias (CP95)	0.0066 (0.93)	0.0067 (0.91)	0.0067 (0.92)	0.0080 (0.92)
	SSD (ESE)	0.1068 (0.1034)	0.1068 (0.1025)	0.1077 (0.1004)	0.1071 (0.0999)
$\rho = 0.3$	Bias (CP95)	-0.0095 (0.92)	-0.0117 (0.92)	-0.0079 (0.91)	-0.0098 (0.91)
	SSD (ESE)	0.0554 (0.0535)	0.0554 (0.0536)	0.0534 (0.0494)	0.0546 (0.0497)
$S_{e(1):1} = 0.94$	Bias (CP95)	-0.0377 (0.98)	-0.0173 (1.00)	-0.0040 (0.95)	0.0027 (0.98)
	SSD (ESE)	0.0368 (0.0558)	0.0299 (0.0454)	0.0250 (0.0261)	0.0201 (0.0222)
$S_{e(2):1} = 0.90$	Bias (CP95)	-0.0278 (0.98)	-0.0547 (0.92)	-0.0017 (0.94)	-0.0080 (0.94)
	SSD (ESE)	0.0528 (0.0652)	0.0454 (0.0590)	0.0265 (0.0256)	0.0238 (0.0245)
$S_{e(1):2} = 0.98$	Bias (CP95)	-0.0108 (1.00)	-0.0092 (0.98)	-0.0035 (0.94)	-0.0015 (0.96)
	SSD (ESE)	0.0092 (0.0186)	0.0104 (0.0163)	0.0094 (0.0102)	0.0076 (0.0085)
$S_{e(2):2} = 0.96$	Bias (CP95)	-0.0174 (0.97)	-0.0263 (0.93)	-0.0012 (0.94)	-0.0039 (0.96)
	SSD (ESE)	0.0269 (0.0314)	0.0212 (0.0287)	0.0111 (0.0111)	0.0098 (0.0106)
$S_{p(1):1} = 0.93$	Bias (CP95)	0.0017 (0.96)	0.0032 (0.96)	-0.0010 (0.95)	-0.0009 (0.96)
	SSD (ESE)	0.0078 (0.0086)	0.0074 (0.0081)	0.0089 (0.0091)	0.0082 (0.0086)
$S_{p(2):1} = 0.95$	Bias (CP95)	0.0074 (0.97)	0.0048 (0.98)	-0.0006 (0.94)	0.0001 (0.96)
	SSD (ESE)	0.0111 (0.0149)	0.0092 (0.0123)	0.0077 (0.0076)	0.0073 (0.0074)
$S_{p(1):2} = 0.97$	Bias (CP95)	0.0014 (0.95)	0.0013 (0.97)	-0.0014 (0.94)	-0.0026 (0.96)
	SSD (ESE)	0.0066 (0.0072)	0.0060 (0.0067)	0.0079 (0.0077)	0.0068 (0.0072)
$S_{p(2):2} = 0.98$	Bias (CP95)	0.0026 (0.99)	0.0039 (0.98)	-0.0004 (0.93)	0.0008 (0.93)
	SSD (ESE)	0.0054 (0.0090)	0.0047 (0.0074)	0.0057 (0.0056)	0.0052 (0.0052)

Table B.6: Unknown assay accuracy probabilities $S_{e(1):1} = 0.94$, $S_{e(2):1} = 0.90$, $S_{e(1):2} = 0.98$, $S_{e(2):2} = 0.96$ and $S_{p(1):1} = 0.93$, $S_{p(2):1} = 0.95$, $S_{p(1):2} = 0.97$, $S_{p(2):2} = 0.98$ (imperfect testing) and unknown $\rho = 0.3$. Average bias of 500 posterior mean estimates (Bias), sample standard deviation of 500 posterior mean estimates (SSD), average of 500 estimates of the posterior standard deviation (ESE), and empirical coverage probability (CP95) of 95% equal-tail credible intervals are given for each estimated parameter when possible. The total number of individuals is $N = 5008$ with disease status generated by logistic population model. Prevalence of each disease in the population are given. For Dorfman testing (DT) and array testing (AT) the average number of tests are shown. DT and AT uses master pools of size 4.

Figure B.2: The patient age distribution for the 2014 Female Iowa SHL data.



Parameter	# of Patients	Race W	Race B	RNP	RMP	RC	S	ST	SC	SP
Count	13,862	11,205	1,506	6,215	1,769	703	3,873	9,546	731	120

Table B.7: Patient categorical covariates for 2014 Female Iowa SHL with positive encoding counts. Race has three categories white (W), black (B) and other and is one-hot encoded. A positive indicator or yes response to other risk factors are counted for each encoding.

Disease Specimen	CT Swab	CT Urine	NG Swab	NG Urine
Se_α (TP)	192	197	126	116
Se_β (FN)	12	11	1	11
Sp_α (TN)	1154	1170	1335	1347
Sp_β (FP)	28	13	17	10

Table B.8: The informed prior parameters for the beta distribution for sensitivity and specificity for each disease and specimen type. The data counts are from the female urine and swab performance tables for CT and NG found in the data sheet for the Aptima Combo 2 Assay [1].

		Probit		Probit		Logistic		Logistic	
		No *Se*, *Sp* ordering		*Se*, *Sp* ordering		No *Se*, *Sp* ordering		*Se*, *Sp* ordering	
$\boldsymbol{\beta}_{1,0}$ (Intercept)	Mean (ETI95)	−0.744	(−0.931, −0.560)	−0.736	(−0.922, −0.546)	−1.089	(−1.324, −0.740)	−1.077	(−1.296, −0.768)
	ESE (HPD95)	0.093	(−0.935, −0.566)	0.095	(−0.927, −0.554)	0.236	(−1.386, −0.897)	0.167	(−1.296, −0.768)
$\boldsymbol{\beta}_{1,1}$ (Age)	Mean (ETI95)	−0.034	(−0.040, −0.028)	−0.034	(−0.040, −0.028)	−0.070	(−0.078, −0.058)	−0.071	(−0.079, −0.061)
	ESE (HPD95)	0.003	(−0.040, −0.028)	0.003	(−0.040, −0.028)	0.005	(−0.081, −0.063)	0.006	(−0.079, −0.061)
$\boldsymbol{\beta}_{1,2}$ (Race W)	Mean (ETI95)	−0.162	(−0.274, −0.049)	−0.164	(−0.275, −0.056)	−0.328	(−0.476, −0.134)	−0.283	(−0.486, −0.171)
	ESE (HPD95)	0.057	(−0.271, −0.047)	0.057	(−0.279, −0.061)	0.082	(−0.514, −0.227)	0.090	(−0.486, −0.171)
$\boldsymbol{\beta}_{1,3}$ (Race B)	Mean (ETI95)	0.014	(−0.131, 0.155)	0.015	(−0.125, 0.155)	−0.010	(−0.197, 0.228)	0.011	(−0.197, 0.188)
	ESE (HPD95)	0.072	(−0.130, 0.156)	0.072	(−0.133, 0.146)	0.104	(−0.244, 0.120)	0.119	(−0.197, 0.188)
$\boldsymbol{\beta}_{1,4}$ (RNP)	Mean (ETI95)	0.140	(0.074, 0.206)	0.140	(0.074, 0.204)	0.311	(0.174, 0.384)	0.251	(0.176, 0.372)
	ESE (HPD95)	0.034	(0.073, 0.204)	0.034	(0.074, 0.203)	0.076	(0.152, 0.328)	0.063	(0.176, 0.372)
$\boldsymbol{\beta}_{1,5}$ (RMP)	Mean (ETI95)	0.164	(0.072, 0.256)	0.164	(0.073, 0.257)	0.231	(0.163, 0.449)	0.321	(0.165, 0.426)
	ESE (HPD95)	0.047	(0.073, 0.256)	0.047	(0.075, 0.258)	0.077	(0.133, 0.376)	0.062	(0.165, 0.426)
$\boldsymbol{\beta}_{1,6}$ (RC)	Mean (ETI95)	0.732	(0.616, 0.846)	0.737	(0.622, 0.857)	1.349	(1.191, 1.518)	1.405	(1.204, 1.502)
	ESE (HPD95)	0.059	(0.615, 0.844)	0.059	(0.622, 0.857)	0.093	(1.150, 1.437)	0.087	(1.204, 1.502)
$\boldsymbol{\beta}_{1,7}$ (S)	Mean (ETI95)	0.132	(0.055, 0.207)	0.134	(0.055, 0.211)	0.261	(0.145, 0.380)	0.290	(0.150, 0.365)
	ESE (HPD95)	0.039	(0.057, 0.209)	0.039	(0.058, 0.213)	0.074	(0.114, 0.318)	0.063	(0.150, 0.365)
$\boldsymbol{\beta}_{1,8}$ (ST)	Mean (ETI95)	0.097	(0.005, 0.194)	0.091	(−0.004, 0.191)	0.265	(0.050, 0.353)	0.178	(0.050, 0.323)
	ESE (HPD95)	0.049	(0.004, 0.192)	0.050	(−0.007, 0.186)	0.149	(0.022, 0.270)	0.087	(0.050, 0.323)
$\boldsymbol{\beta}_{1,9}$ (SC)	Mean (ETI95)	0.087	(−0.054, 0.223)	0.086	(−0.049, 0.223)	0.130	(−0.036, 0.390)	0.249	(−0.020, 0.359)
	ESE (HPD95)	0.071	(−0.056, 0.219)	0.070	(−0.051, 0.219)	0.159	(−0.086, 0.283)	0.148	(−0.020, 0.359)
$\boldsymbol{\beta}_{1,10}$ (SP)	Mean (ETI95)	0.103	(−0.243, 0.426)	0.100	(−0.246, 0.419)	0.113	(−0.362, 0.678)	0.320	(−0.383, 0.602)
	ESE (HPD95)	0.171	(−0.216, 0.449)	0.173	(−0.238, 0.427)	0.257	(−0.479, 0.438)	0.301	(−0.383, 0.602)
$\boldsymbol{\beta}_{2,0}$ (Intercept)	Mean (ETI95)	−2.501	(−2.903, −2.133)	−2.511	(−2.919, −2.140)	−4.786	(−5.841, −4.127)	−4.900	(−5.768, −4.311)
	ESE (HPD95)	0.195	(−2.881, −2.120)	0.197	(−2.907, −2.131)	0.432	(−6.069, −4.535)	0.356	(−5.768, −4.311)
$\boldsymbol{\beta}_{2,1}$ (Age)	Mean (ETI95)	−0.007	(−0.018, 0.003)	−0.007	(−0.018, 0.003)	−0.025	(−0.046, 0.001)	−0.025	(−0.042, −0.002)
	ESE (HPD95)	0.005	(−0.018, 0.003)	0.005	(−0.018, 0.003)	0.010	(−0.051, −0.011)	0.017	(−0.042, −0.002)
$\boldsymbol{\beta}_{2,2}$ (Race W)	Mean (ETI95)	−0.132	(−0.375, 0.126)	−0.133	(−0.362, 0.119)	−0.395	(−0.784, 0.221)	−0.305	(−0.794, 0.148)
	ESE (HPD95)	0.128	(−0.380, 0.117)	0.123	(−0.369, 0.105)	0.232	(−0.880, −0.059)	0.313	(−0.794, 0.148)
$\boldsymbol{\beta}_{2,3}$ (Race B)	Mean (ETI95)	0.296	(0.020, 0.581)	0.298	(0.021, 0.587)	0.609	(0.119, 1.281)	0.592	(0.122, 1.191)
	ESE (HPD95)	0.145	(0.026, 0.585)	0.143	(0.009, 0.571)	0.321	(0.004, 0.971)	0.365	(0.122, 1.191)
$\boldsymbol{\beta}_{2,4}$ (RNP)	Mean (ETI95)	0.061	(−0.082, 0.206)	0.058	(−0.081, 0.203)	0.098	(−0.142, 0.482)	0.095	(−0.139, 0.450)
	ESE (HPD95)	0.074	(−0.090, 0.198)	0.073	(−0.079, 0.206)	0.194	(−0.224, 0.319)	0.288	(−0.100, 0.486)
$\boldsymbol{\beta}_{2,5}$ (RMP)	Mean (ETI95)	0.196	(0.002, 0.376)	0.185	(−0.020, 0.371)	0.310	(0.095, 0.877)	0.502	(0.062, 0.749)
	ESE (HPD95)	0.096	(0.004, 0.376)	0.099	(−0.016, 0.375)	0.338	(0.014, 0.678)	0.169	(0.062, 0.749)
$\boldsymbol{\beta}_{2,6}$ (RC)	Mean (ETI95)	0.923	(0.742, 1.103)	0.924	(0.743, 1.107)	2.163	(1.877, 2.561)	2.154	(1.864, 2.506)
	ESE (HPD95)	0.092	(0.737, 1.098)	0.092	(0.747, 1.109)	0.185	(1.799, 2.390)	0.122	(1.864, 2.506)
$\boldsymbol{\beta}_{2,7}$ (S)	Mean (ETI95)	0.137	(−0.024, 0.291)	0.146	(−0.012, 0.303)	0.203	(−0.030, 0.654)	0.503	(0.005, 0.641)
	ESE (HPD95)	0.081	(−0.030, 0.284)	0.081	(−0.006, 0.310)	0.186	(−0.105, 0.486)	0.178	(0.005, 0.641)
$\boldsymbol{\beta}_{2,8}$ (ST)	Mean (ETI95)	0.158	(−0.040, 0.380)	0.162	(−0.033, 0.386)	0.461	(0.031, 1.072)	0.358	(0.080, 0.937)
	ESE (HPD95)	0.106	(−0.049, 0.363)	0.107	(−0.037, 0.380)	0.240	(−0.052, 0.753)	0.187	(0.080, 0.937)
$\boldsymbol{\beta}_{2,9}$ (SC)	Mean (ETI95)	0.162	(−0.077, 0.388)	0.165	(−0.075, 0.395)	0.420	(−0.010, 0.878)	0.611	(0.008, 0.826)
	ESE (HPD95)	0.121	(−0.075, 0.390)	0.121	(−0.071, 0.397)	0.219	(−0.118, 0.661)	0.314	(0.008, 0.826)
$\boldsymbol{\beta}_{2,10}$ (SP)	Mean (ETI95)	0.185	(−0.409, 0.701)	0.183	(−0.406, 0.688)	0.265	(−0.809, 1.463)	0.448	(−0.684, 1.332)
	ESE (HPD95)	0.283	(−0.361, 0.743)	0.278	(−0.344, 0.743)	0.566	(−1.139, 0.991)	0.681	(−0.532, 1.433)

Table B.9: The regression parameter estimates for all stratum of the Iowa test set using the Bayesian probit and logistic model. The sensitivity and specificity used the Hologic data to inform beta distributed priors. Ordering was optionally enforced with regards to the size of the pool. The $\boldsymbol{\beta}$ parameters had a normal prior with zero mean and a diffuse covariance matrix. The parameter ρ had a uniform prior. The posterior mean estimate, estimated posterior standard deviation, 95% highest posterior density interval and 95% equal-tailed interval are given for each predictor for each posterior distribution. The diseases are chlamydia ($d = 1$) and gonorrhea ($d = 2$).

		Probit		Probit		Logistic		Logistic	
		No Se, Sp ordering		Se, Sp ordering		No Se, Sp ordering		Se, Sp ordering	
ρ	Mean (ETI95)	0.484	(0.390, 0.566)	0.487	(0.406, 0.561)	0.399	(0.321, 0.480)	0.384	(0.321, 0.473)
	ESE (HPD95)	0.043	(0.400, 0.571)	0.040	(0.411, 0.565)	0.032	(0.303, 0.436)	0.040	(0.321, 0.473)
$S_{e(1),1}$(Swab, 1)	Mean (ETI95)	0.985	(0.975, 0.992)	0.985	(0.977, 0.992)	0.985	(0.979, 0.992)	0.986	(0.980, 0.991)
	ESE (HPD95)	0.004	(0.977, 0.993)	0.004	(0.978, 0.993)	0.002	(0.977, 0.989)	0.004	(0.980, 0.991)
$S_{e(2),1}$(Swab, 2)	Mean (ETI95)	0.943	(0.908, 0.970)	0.975	(0.965, 0.983)	0.949	(0.917, 0.966)	0.977	(0.968, 0.981)
	ESE (HPD95)	0.016	(0.911, 0.972)	0.005	(0.965, 0.984)	0.013	(0.917, 0.964)	0.004	(0.968, 0.981)
$S_{e(3),1}$(Swab, 3)	Mean (ETI95)	0.943	(0.908, 0.970)	0.972	(0.962, 0.980)	0.949	(0.919, 0.968)	0.974	(0.965, 0.978)
	ESE (HPD95)	0.016	(0.911, 0.971)	0.005	(0.963, 0.981)	0.015	(0.912, 0.958)	0.004	(0.966, 0.979)
$S_{e(4),1}$(Swab, 4)	Mean (ETI95)	0.986	(0.977, 0.993)	0.970	(0.959, 0.978)	0.986	(0.979, 0.992)	0.971	(0.962, 0.977)
	ESE (HPD95)	0.004	(0.978, 0.993)	0.005	(0.960, 0.979)	0.003	(0.979, 0.991)	0.004	(0.963, 0.977)
$S_{e(5),1}$(Urine, 1)	Mean (ETI95)	0.940	(0.904, 0.968)	0.940	(0.905, 0.967)	0.936	(0.914, 0.964)	0.944	(0.916, 0.961)
	ESE (HPD95)	0.016	(0.908, 0.970)	0.016	(0.908, 0.968)	0.013	(0.914, 0.962)	0.014	(0.919, 0.964)
$S_{e(1),2}$(Swab, 1)	Mean (ETI95)	0.944	(0.910, 0.971)	0.951	(0.927, 0.971)	0.941	(0.920, 0.968)	0.956	(0.934, 0.967)
	ESE (HPD95)	0.015	(0.912, 0.972)	0.011	(0.927, 0.971)	0.023	(0.920, 0.965)	0.009	(0.934, 0.967)
$S_{e(2),2}$(Swab, 2)	Mean (ETI95)	0.907	(0.852, 0.951)	0.933	(0.908, 0.955)	0.912	(0.868, 0.945)	0.931	(0.915, 0.949)
	ESE (HPD95)	0.025	(0.857, 0.953)	0.012	(0.911, 0.957)	0.017	(0.858, 0.930)	0.010	(0.915, 0.949)
$S_{e(3),2}$(Swab, 3)	Mean (ETI95)	0.907	(0.851, 0.951)	0.922	(0.895, 0.944)	0.913	(0.866, 0.946)	0.925	(0.903, 0.939)
	ESE (HPD95)	0.026	(0.854, 0.953)	0.013	(0.896, 0.945)	0.029	(0.855, 0.929)	0.011	(0.903, 0.939)
$S_{e(4),2}$(Swab, 4)	Mean (ETI95)	0.945	(0.910, 0.971)	0.912	(0.882, 0.937)	0.949	(0.920, 0.968)	0.916	(0.891, 0.932)
	ESE (HPD95)	0.016	(0.913, 0.972)	0.014	(0.885, 0.938)	0.015	(0.913, 0.958)	0.010	(0.894, 0.934)
$S_{e(5),2}$(Urine, 1)	Mean (ETI95)	0.985	(0.957, 0.998)	0.985	(0.957, 0.998)	0.987	(0.968, 0.997)	0.984	(0.968, 0.996)
	ESE (HPD95)	0.011	(0.963, 1.000)	0.011	(0.963, 1.000)	0.010	(0.968, 0.996)	0.012	(0.975, 0.999)
$S_{p(1),1}$(Swab, 1)	Mean (ETI95)	0.998	(0.997, 0.999)	0.995	(0.993, 0.996)	0.998	(0.997, 0.999)	0.995	(0.994, 0.996)
	ESE (HPD95)	0.001	(0.997, 0.999)	0.001	(0.993, 0.996)	0.000	(0.997, 0.999)	0.000	(0.994, 0.996)
$S_{p(2),1}$(Swab, 2)	Mean (ETI95)	0.988	(0.982, 0.994)	0.995	(0.994, 0.996)	0.990	(0.983, 0.993)	0.995	(0.994, 0.996)
	ESE (HPD95)	0.003	(0.982, 0.994)	0.001	(0.994, 0.996)	0.002	(0.982, 0.991)	0.001	(0.994, 0.996)
$S_{p(3),1}$(Swab, 3)	Mean (ETI95)	0.988	(0.981, 0.994)	0.995	(0.994, 0.997)	0.988	(0.983, 0.993)	0.995	(0.994, 0.996)
	ESE (HPD95)	0.003	(0.982, 0.994)	0.001	(0.994, 0.997)	0.003	(0.982, 0.991)	0.001	(0.994, 0.996)
$S_{p(4),1}$(Swab, 4)	Mean (ETI95)	0.995	(0.992, 0.997)	0.996	(0.995, 0.998)	0.995	(0.993, 0.997)	0.996	(0.995, 0.997)
	ESE (HPD95)	0.001	(0.992, 0.997)	0.001	(0.995, 0.998)	0.002	(0.992, 0.996)	0.001	(0.995, 0.997)
$S_{p(5,1}$(Urine, 1)	Mean (ETI95)	0.979	(0.972, 0.986)	0.979	(0.972, 0.986)	0.978	(0.974, 0.985)	0.979	(0.974, 0.984)
	ESE (HPD95)	0.004	(0.972, 0.987)	0.004	(0.972, 0.986)	0.004	(0.973, 0.983)	0.004	(0.974, 0.984)
$S_{p(1),2}$(Swab, 1)	Mean (ETI95)	0.999	(0.998, 0.999)	0.997	(0.996, 0.998)	0.999	(0.998, 0.999)	0.997	(0.996, 0.998)
	ESE (HPD95)	0.000	(0.998, 0.999)	0.000	(0.996, 0.998)	0.000	(0.998, 0.999)	0.000	(0.996, 0.998)
$S_{p(2),2}$(Swab, 2)	Mean (ETI95)	0.992	(0.987, 0.996)	0.997	(0.996, 0.998)	0.990	(0.988, 0.996)	0.997	(0.997, 0.998)
	ESE (HPD95)	0.002	(0.987, 0.996)	0.000	(0.996, 0.998)	0.002	(0.988, 0.995)	0.000	(0.997, 0.998)
$S_{p(3),2}$(Swab, 3)	Mean (ETI95)	0.992	(0.986, 0.996)	0.997	(0.997, 0.998)	0.992	(0.988, 0.995)	0.998	(0.997, 0.998)
	ESE (HPD95)	0.002	(0.987, 0.996)	0.000	(0.997, 0.998)	0.001	(0.988, 0.995)	0.000	(0.997, 0.998)
$S_{p(4),2}$(Swab, 4)	Mean (ETI95)	0.997	(0.995, 0.998)	0.998	(0.997, 0.999)	0.997	(0.995, 0.998)	0.998	(0.997, 0.999)
	ESE (HPD95)	0.001	(0.995, 0.998)	0.000	(0.997, 0.999)	0.001	(0.995, 0.998)	0.000	(0.997, 0.999)
$S_{p(5,2}$(Urine, 1)	Mean (ETI95)	0.994	(0.991, 0.996)	0.994	(0.991, 0.996)	0.993	(0.992, 0.996)	0.993	(0.992, 0.995)
	ESE (HPD95)	0.001	(0.991, 0.996)	0.001	(0.991, 0.996)	0.001	(0.991, 0.995)	0.001	(0.992, 0.996)

Table B.10: The scale, sensitivity and specificity parameter estimates for all stratum of the Iowa test set using the Bayesian probit and logistic model. The sensitivity and specificity used the Hologic data to inform beta distributed priors. Ordering was optionally enforced with regards to the size of the pool. With ordering, we used the following partial ordering corresponding to pool size $S_{e(1),d} \geq S_{e(2),d} \geq S_{e(3),d} \geq S_{e(4),d}$, and $S_{p(1),d} \leq S_{p(2),d} \leq S_{p(3),d} \leq S_{p(4),d}$ for $d = 1, 2$. The $\boldsymbol{\beta}$ parameters had a normal prior with zero mean and a diffuse covariance matrix. The parameter ρ had a uniform prior. The posterior mean estimate, estimated posterior standard deviation, 95% highest posterior density interval and 95% equal-tailed interval are given for each predictor for each posterior distribution. The diseases are chlamydia ($d = 1$) and gonorrhea ($d = 2$).

Appendix C Introductory CUDA Code Repositories

C.1 Simple Program

The R script is found in the cuda_examples github repository of pcubre with the file name add_vector.R. is the framework to interact between R and C/C++. We dynamically load the library, provide a function signature to the library, and unload the library.

The CUDA program is found in the cuda_examples github repository of pcubre under the name add_vector.cu. It has three parts: load headers, a device kernel, and a host program. The device kernel finds its own index and then performs the appropriate addition based on the index. The host program allocates memory on the device, copies data to the device, launches the kernel, moves data back to the host and frees device memory.

C.2 Second Program

The CUDA program found cuda_examples github repository of pcubre under the name add_vector_2.cu. is a small program using additional libraries. It has two parts: load headers, and a host program. The host program is different from the earlier program as uses a handler to set up the library and moves memory between the device and host via matrix and vector functions.

C.3 Simulations

Vector Size	Language	Min	1-Quartile	Mean	Median	3-Quartile	Max	Evaluations
10^1	R	0.0003	0.0013	0.0022	0.0018	0.0033	0.0096	100
10^1	CUDA kernel	0.2494	0.2607	0.2946	0.2798	0.2884	1.5244	100
10^1	CUDA library	0.6227	0.6353	0.6499	0.6412	0.6465	1.3791	100
10^2	R	0.0005	0.0008	0.0033	0.0023	0.0061	0.0069	100
10^2	CUDA kernel	0.2464	0.2574	0.2729	0.2755	0.2849	0.3086	100
10^2	CUDA library	0.6274	0.6427	0.6660	0.6512	0.6570	1.7869	100
10^3	R	0.0027	0.0041	0.0057	0.0048	0.0071	0.0163	100
10^3	CUDA kernel	0.2632	0.2780	0.2912	0.2899	0.2995	0.5122	100
10^3	CUDA library	0.6441	0.6632	0.6770	0.6744	0.6868	0.8135	100
10^4	R	0.0233	0.0274	0.0378	0.0411	0.0466	0.0535	100
10^4	CUDA kernel	0.4018	0.5473	0.5387	0.5554	0.5673	0.6479	100
10^4	CUDA library	0.7702	0.9403	0.9504	0.9616	0.9690	2.2898	100
10^5	R	0.2279	0.2309	0.2599	0.2326	0.2364	1.4865	100
10^5	CUDA kernel	1.6316	1.6719	1.7667	1.6797	1.6879	3.0429	100
10^5	CUDA library	1.8702	1.8967	2.0438	1.9076	1.9263	4.1852	100
10^6	R	2.2768	2.3026	3.0070	2.3203	2.3871	25.8529	100
10^6	CUDA kernel	13.3620	13.7002	16.3052	14.5700	15.5582	44.3450	100
10^6	CUDA library	12.5322	12.9081	16.0420	13.8855	15.1548	44.3460	100
10^7	R	27.4952	28.2571	29.9086	29.3003	30.8850	54.2894	100
10^7	CUDA kernel	183.8623	187.8381	195.1081	190.4356	198.7771	239.1906	100
10^7	CUDA library	167.3054	171.2911	177.7724	175.2761	182.1034	211.5703	100
10^8	R	298.7344	299.1144	306.8947	299.7457	307.9471	539.6474	100
10^8	CUDA kernel	1846.0654	1871.2546	1898.4229	1894.1307	1913.5316	1986.4911	100
10^8	CUDA library	1690.8641	1719.6367	1748.3970	1744.5908	1773.2190	1842.2082	100
10^9	R	2989.5161	3395.3064	3935.1477	4158.1613	4428.6801	5648.8770	100
10^9	CUDA kernel	18425.2621	18504.2436	19187.3646	18687.5881	19391.7940	27757.4831	100
10^9	CUDA library	16905.1594	16985.2045	17439.2973	17152.4995	17788.5568	22584.4471	100

Table C.1: Microsecond timings for 100 evaluations of functions that add two vectors. The both vectors have dimension 10^i for $i = 1, \ldots, 9$. The three functions where called from R and utilized either R vector addition, the Nvidia cuBLAS library or a custom kernel implementation. They were added on a Palmetto HPC node with a V100 and 64gb of memory.

Vector Size	Language	Min	1-Quartile	Mean	Median	3-Quartile	Max	Evaluations
10^1	R	11.4250	13.0565	16.7017	15.5190	16.6115	59.8930	100
10^1	CUDA kernel	298.5260	308.6900	362.8955	329.5865	337.0180	3164.5810	100
10^1	CUDA library	784.3160	793.5230	834.7403	797.2535	803.4535	1800.4250	100
10^2	R	32.2200	36.2460	41.4432	41.8470	45.4565	64.0570	100
10^2	CUDA kernel	300.4320	310.9335	344.7658	329.2185	339.1205	1321.0960	100
10^2	CUDA library	759.4510	815.6710	862.9754	822.5620	832.9065	2050.0010	100
10^3	R	214.0760	376.3590	369.5524	387.1125	395.1855	411.3300	100
10^3	CUDA kernel	313.4450	336.1470	373.6682	346.4940	361.3675	1562.3670	100
10^3	CUDA library	814.8790	845.6790	910.5580	867.9925	889.1225	2259.6360	100
10^4	R	1936.4810	1946.0275	2091.2105	1950.0345	1953.5680	4431.4660	100
10^4	CUDA kernel	664.2980	686.7515	714.5434	698.6620	707.5625	1741.4890	100
10^4	CUDA library	927.5570	944.4415	996.3053	959.2785	976.0510	2134.8460	100
10^5	R	19197.7830	20373.8270	20649.0109	20391.0320	20425.0085	44907.5530	100
10^5	CUDA kernel	4579.0040	4612.6085	4768.5866	4655.5365	4793.0140	6053.4960	100
10^5	CUDA library	1997.4180	2196.2820	2430.7635	2327.2155	2512.5320	3996.6350	100
10^6	R	196482.0950	199265.1430	206804.1463	202277.2085	209849.7790	244242.4630	100
10^6	CUDA kernel	38997.2570	39674.5730	41981.3113	40738.0560	42383.2545	65577.7560	100
10^6	CUDA library	14832.3970	15278.4140	17632.3712	16187.4545	17009.1640	42120.4920	100
10^7	R	2534010.7790	2587059.1425	2601346.4783	2600044.5515	2613744.6295	2691286.0410	100
10^7	CUDA kernel	444139.5800	447063.4960	457329.8523	450491.4200	467333.3240	490134.9900	100
10^7	CUDA library	193418.9970	195866.3945	206724.4619	201731.5470	216095.8375	237053.7630	100
10^8	R	22992094.6780	23023521.5920	23479118.8865	23042662.8500	23075528.4960	33431615.0600	100
10^8	CUDA kernel	4431228.8120	4451962.2615	4478375.7105	4468270.3020	4472830.3840	5029489.3510	100
10^8	CUDA library	1907584.4630	1926139.1810	1963621.5519	1939344.4990	1947535.3540	2861840.3780	100
10^9	R	206015005.4990	209943465.9835	220938231.9035	215718359.0965	233421430.0020	273024560.9830	100
10^9	CUDA kernel	45252926.2350	47733750.0645	48583731.8736	48116913.2375	49526135.6595	53804125.3580	100
10^9	CUDA library	20002668.8770	22343227.2525	22643012.1014	22483137.1600	22928511.6130	26978105.7540	100

Table C.2: Microsecond timings for 100 evaluations of functions that sequentially adds two vectors 100 times. The both vectors have dimension 10^i for $i = 1, \ldots, 9$. The three functions where called from R and utilized either R vector addition, the Nvidia cuBLAS library or a custom kernel implementation. They were added on a Palmetto HPC node with a V100 and 64gb of memory.

Figure C.1: Average times for 100 evaluations of functions that add two vectors in microseconds. The axes scale is log_{10}. The both vectors have dimension 10^i for $i = 1, \ldots, 9$. The three functions where called from R and utilized either R vector addition, the Nvidia cuBLAS library or a custom kernel implementation. They were added on a Palmetto HPC node with a V100 and 64gb of memory.

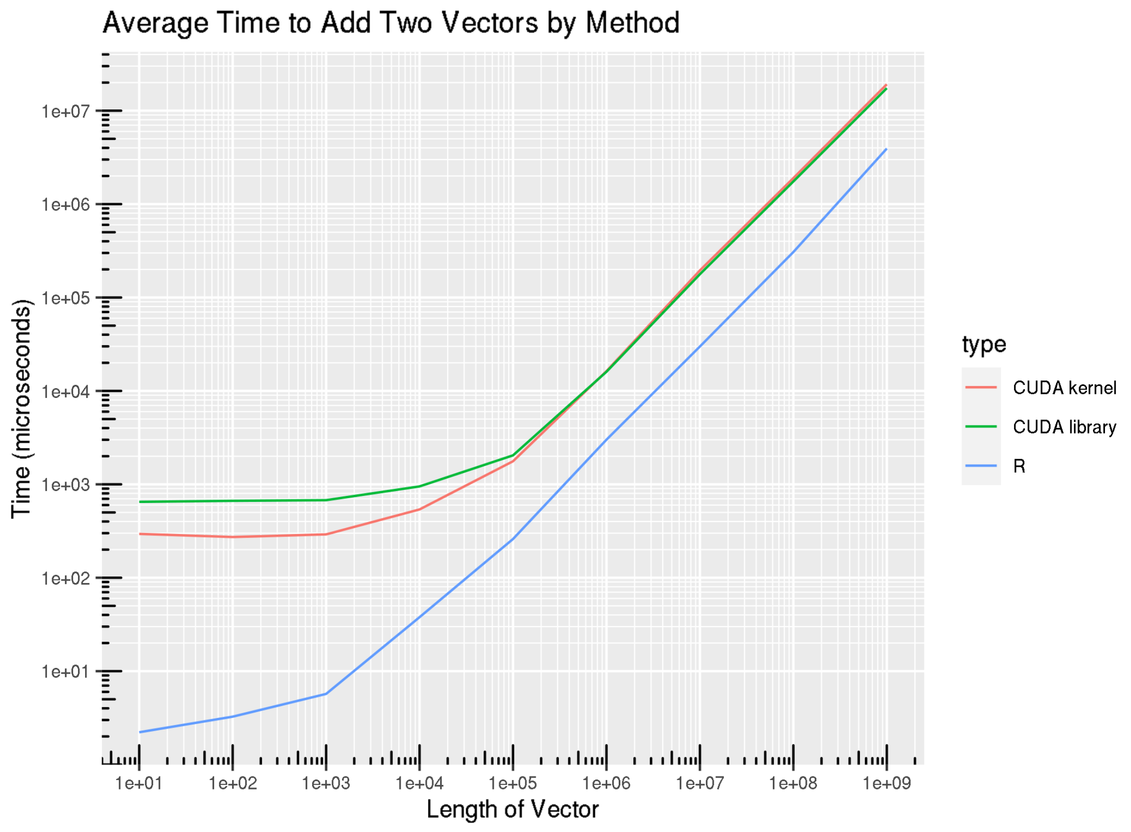
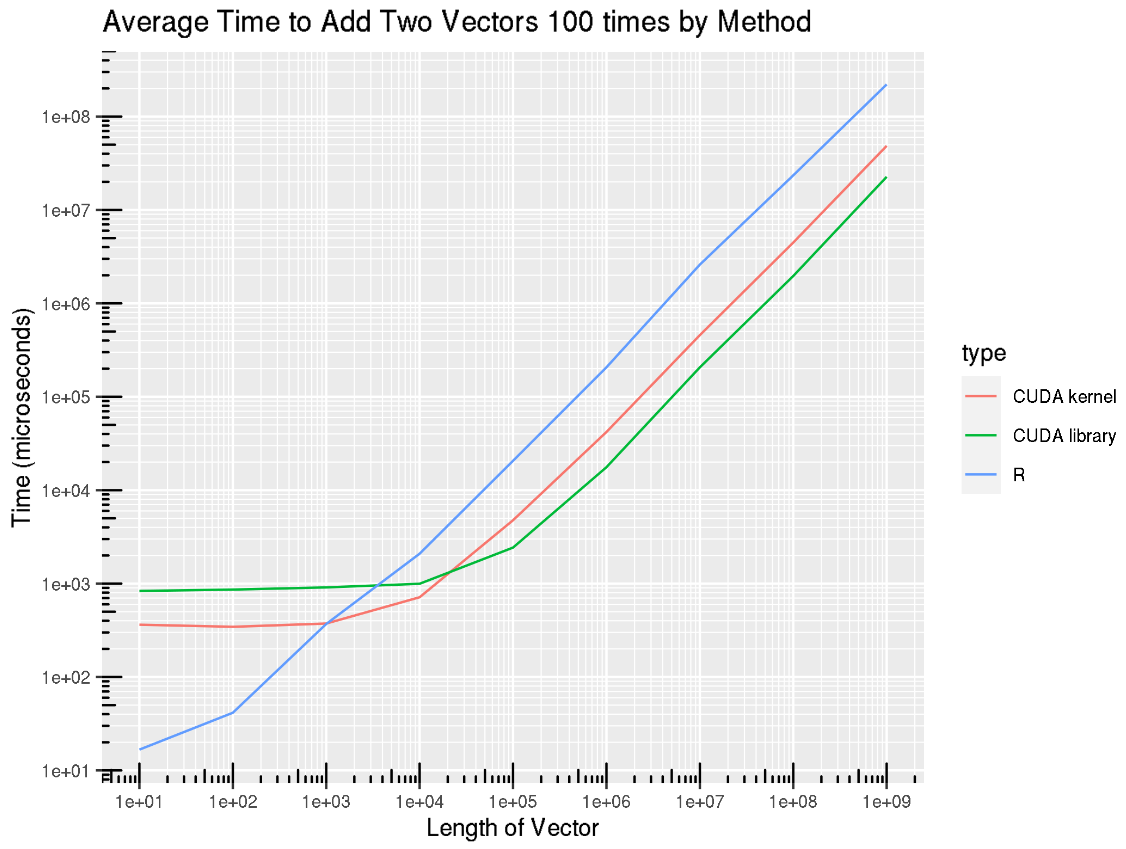


Figure C.2: Average times for 100 evaluations of functions that sequentially add two vectors 100 times in microseconds. The axes scale is log_{10}. The both vectors have dimension 10^i for $i = 1, \ldots, 9$. The three functions where called from R and utilized either R vector addition, the Nvidia cuBLAS library or a custom kernel implementation. They were added on a Palmetto HPC node with a V100 and 64gb of memory.



Appendix D Descent Algorithms

In this section, we review the concepts of stochastic gradient descent and stochastic coordinate descent for linear regression. The objective is to apply the concepts to simple and familiar problems in order to grasp the methodology.

D.1 Linear Regression

Linear regression acts as a baseline for most modeling. It can be written as the following forms:

$$\begin{aligned}&\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2N} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2, \end{aligned} \tag{1}$$

where N is the number of observations, p is the number of variables and y_i is ith observation and x_{ij} is the jth co-variate of the ith observation.

We can find the exact solution by taking partial derivatives with respect to $\boldsymbol{\beta}$.

$$\begin{aligned}&\frac{\partial}{\partial \beta_k} \left(\frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right) \\ &= -\frac{1}{N} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j) x_{ik} \\ &= -\frac{1}{N} \mathbf{X}_k^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \end{aligned} \tag{2}$$

or

$$\nabla(\boldsymbol{\beta}) = -\frac{1}{N} \mathbf{X}^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \tag{3}$$

Thus we have a critical point $\boldsymbol{\beta}_{ols}$ for the solution to $0 = \nabla(\boldsymbol{\beta})$.

$$\begin{aligned} 0 = \nabla(\boldsymbol{\beta}) &= -\frac{1}{N} \mathbf{X}^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \\ \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{Y} \text{ (the normal equation)} \end{aligned}$$

$$\boldsymbol{\beta}_{ols} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \text{ when } \mathbf{X}^T\mathbf{X} \text{ is invertible} \tag{4}$$

D.2 Stochastic Descent

Even though there can be an exact solution to linear regression, we can use other methods to compare. One method that we present in two variations are stochastic methods. It was Robbins and Monro [83] who first presented the idea to solve optimization problems stochastically. Their methodology was to iterate to find the solution by $x_{n+1} - x_n = a_n(\alpha - y_n)$ where a_n is in ℓ_2, y_n is a random variable, α is a constant, and x_n is a sequence heading to the solution. This method essentially is a contraction that finds a solution, similar to other methods such as Newton's method, $x_{n+1} - x_n = \nabla^2 F(x_n)^{-1}\nabla F(x_n)$.

To distinguish between the two methods, we set the notation $F(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{i=1}^N(y_i - \sum_{j=1}^p x_{ij}\boldsymbol{\beta}_j)^2$ and $F_i(\boldsymbol{\beta}) = \frac{1}{2}(y_i - \sum_{j=1}^p x_{ij}\boldsymbol{\beta}_j)^2$. Thus we have $F(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=1}^N F_i(\boldsymbol{\beta})$.

For Stochastic Gradient Descent (SGD) we have the equation

$$\begin{aligned}\boldsymbol{\beta}_{t+1} &= \boldsymbol{\beta}_t - \eta_t \nabla F_i(\boldsymbol{\beta}_t) \\ &= \boldsymbol{\beta}_t - \eta_t(-\mathbf{x}_i(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_t))\end{aligned} \tag{5}$$

where η_t is called the learning rate. The learning rate is usually of the form $\eta_t = Ct^{-c}$ for some constants C and c. Then we have the algorithm:

```
Result: argmin_β $F(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^p$
while not converged do
    Chose $i \in \{1, \dots, N\}$ uniformly at random.
    Let $\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_t(-\mathbf{x}_i(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_t)$.
end
```

Moreover to add stability one can use the implicit SGD

$$\begin{aligned}\boldsymbol{\beta}_{t+1} &= \boldsymbol{\beta}_t - \eta_t \nabla F_i(\boldsymbol{\beta}_{t+1}) \\ &= \boldsymbol{\beta}_t - \eta_t(-x_i(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_{t+1})) \\ &= \boldsymbol{\beta}_t - \frac{\eta_t}{1 + \eta_t \|\mathbf{x}_i\|_2^2}(-x_i(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_t)).\end{aligned} \tag{6}$$

Additionally, an intermediate method uses mini-batch where one takes a subset of $[N]$,

$S_t \subset [N]$, of size $B = |S_t|$ such that

$$\begin{aligned}\boldsymbol{\beta}_{t+1} &= \boldsymbol{\beta}_t - \eta_t \frac{1}{B}\sum_{i\in S_t} \nabla F_i(\boldsymbol{\beta}_t) \\ &= \boldsymbol{\beta}_t - \eta_t \frac{1}{B}\sum_{i\in S_t} (-x_i(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_t)) \end{aligned} \tag{7}$$

where $B = N$ recovers the gradient descent.

For Stochastic Coordinate Descent (SCD) we have the equation

$$\begin{aligned}\beta_{k,t+1} &= \beta_{k,t} - \alpha_t(\nabla F(\boldsymbol{\beta}_t))_k \\ &= \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})) \end{aligned} \tag{8}$$

where α_t is the step size. In the case of linear regression, a step size of 1 is sufficient. Then we have the algorithm:

```
Result: $\text{argmin}_{\boldsymbol{\beta}} F(\boldsymbol{\beta})$
Set $\boldsymbol{\beta} = 0 \in \mathbb{R}^p$
while not converged do
    Chose $k \in \{1, \dots, p\}$ uniformly at random.
    Let $\beta_{k,t+1} \leftarrow \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}))$.
end
```

D.3 LASSO

Popularized by Tibshirani [93] the least absolute shrinkage and selection operator (lasso) helps automate variable selection. This section we will apply the lasso techniques on both SGD and SCD in linear regression. First we have the new problem form:

$$\begin{aligned}&\underset{\boldsymbol{\beta}}{\text{argmin}} \frac{1}{2N}\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\beta\|_1 \\ &= \underset{\boldsymbol{\beta}}{\text{argmin}} \frac{1}{2N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{i=1}^{p}|\beta_j|, \end{aligned} \tag{9}$$

for any λ. Let $F(\boldsymbol{\beta}) = \frac{1}{2N}\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$, $G(\boldsymbol{\beta}) = \lambda\|\beta\|_1$ and $H(\boldsymbol{\beta}) = F(\boldsymbol{\beta}) + G(\boldsymbol{\beta})$.

We can find the solution by taking partial derivatives with respect to $\boldsymbol{\beta}$.

$$\begin{aligned}
&\frac{\partial}{\partial \beta_k}\left(\frac{1}{2N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p}|\beta_j|\right) \\
&= -\frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)x_{ik} + \lambda\,\mathrm{sign}(\beta_k) \\
&= -\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\,\mathrm{sign}(\beta_k) \qquad (10)
\end{aligned}$$

or

$$\nabla H(\boldsymbol{\beta}) = -\frac{1}{N}\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) + \mathrm{sign}(\boldsymbol{\beta}) \qquad (11)$$

where $\mathrm{sign}(\cdot) \in \{-1, 0, 1\}^p$ is a vector version of the sign function for inputs in $\mathbb{R}^p$.

When solving our gradient descent problems β can change sign component wise and we must allow for the possibility that a component is zero. Therefore we want to track the sign changes at each iteration of $\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \gamma_t \nabla H(\boldsymbol{\beta}_t)$.

Solving for critical points,

$$\begin{aligned}
0 &= -\frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)x_{ik} + \lambda\,\mathrm{sign}(\beta_k) \\
\beta_k &= \frac{\frac{1}{N}\sum_{i=1}^{n} x_{ik}(y_i - \sum_{j\neq k}^{p} x_{ij}\beta_j) - \mathrm{sign}(\beta_k)\lambda}{\frac{1}{N}\sum_{i=1}^{N} x_{ik}^2}. \qquad (12)
\end{aligned}$$

However there are issues. Let $c_k = \frac{1}{N}\sum_{i=1}^{n} x_{ik}(y_i - \sum_{j\neq k}^{p} x_{ij}\beta_j)$. Then we have the following conditions:

$$\beta_k > 0 \text{ if and only if } c_k > \lambda, \beta_k < 0 \quad \text{if and only if } c_k < -\lambda, \beta_k = 0 \text{ if and only if } c_k \in [-\lambda, \lambda].$$

Thus we have the operator

Definition 2 (Soft Threshold Operator)**.** *Let* $\omega \in \mathbb{R}^d$ *and* τ *be a vector where each component is the same,* $\mathrm{sign}(\cdot)$ *be the vector sign function, and* $(\cdot)_+ = \max(\cdot, 0)$*. Then* $s_\tau(\omega) = \mathrm{sign}(\omega)(|\omega| - \tau)_+$

is the soft threshold operator with

$$s_\tau(\omega) = \text{sign}(\omega)(|\omega| - \tau)_+ = \begin{cases} \omega - \tau & \omega > \tau \\ \omega + \tau & \omega < -\tau \\ 0 & \omega \in [-\tau, \tau] \end{cases}$$

Then

$$\beta_k = \frac{s_\lambda\left(\frac{1}{N}\sum_{i=1}^n x_{ik}(y_i - \sum_{j\neq k}^p x_{ij}\beta_j)\right)}{\frac{1}{N}\sum_{i=1}^N x_{ik}^2}. \tag{13}$$

D.4 LASSO with Stochastic Descent

Incorporating stochastic descent in the SGD and SCD cases with lasso is a small modification. In the case of SGD, we have the following conditions:

$$\begin{aligned}
\beta_{k,t+1} > 0 &\text{ if and only if } \beta_{k,t} - \eta_t \nabla H_i(\boldsymbol{\beta}_t) > 0 \\
&\text{ if and only if } \beta_{k,t} - \eta_t(-(y_i - \sum_{j=1}^p x_{ij}\beta_{j,t})x_{ik} + \lambda) > 0 \\
&\text{ if and only if } \beta_{k,t} - \eta_t(-(y_i - \sum_{j=1}^p x_{ij}\beta_{j,t})x_{ik}) > \eta_t\lambda, \\
\beta_{k,t+1} < 0 &\text{ if and only if } \beta_{k,t} - \eta_t \nabla H_i(\boldsymbol{\beta}_t) < 0 \\
&\text{ if and only if } \beta_{k,t} - \eta_t(-(y_i - \sum_{j=1}^p x_{ij}\beta_{j,t})x_{ik} - \lambda) < 0 \\
&\text{ if and only if } \beta_{k,t} - \eta_t(-(y_i - \sum_{j=1}^p x_{ij}\beta_{j,t})x_{ik}) < -\eta_t\lambda, \\
\beta_{k,t+1} = 0 &\text{ if and only if } \beta_{k,t} - \eta_t(-(y_i - \sum_{j=1}^p x_{ij}\beta_{j,t})x_{ik} \in [-\eta_t\lambda, \eta_t\lambda].
\end{aligned} \tag{14}$$

The above equations can be rewritten as

$$\boldsymbol{\beta}_{t+1} = s_{\eta_t\lambda}\left(\boldsymbol{\beta}_t - \eta_t(-(y_i - \mathbf{x}_i^T\boldsymbol{\beta}_t)\mathbf{x}_i)\right). \tag{15}$$

Thus we have the modified SGD algorithm:

```
Result: $\operatorname{argmin}_{\boldsymbol\beta} H(\boldsymbol\beta)$
Set $\boldsymbol\beta = 0 \in \mathbb{R}^p$
while not converged do
    Chose $i \in \{1, \ldots, N\}$ uniformly at random.
    Let $\boldsymbol\beta_{t+1} \leftarrow s_{\eta_t\lambda}\left(\boldsymbol\beta_t - \eta_t(-(y_i - \mathbf{x}_i^T\boldsymbol\beta_t)\mathbf{x}_i)\right)$.
end
```

In the case of SCD, we have the following conditions:

$$\begin{aligned}
\beta_{k,t+1} > 0 &\text{ if and only if } \beta_{k,t} - \alpha_t \nabla H(\boldsymbol\beta_t)_j > 0\\
&\text{ if and only if } \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t) + \lambda) > 0\\
&\text{ if and only if } \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t)) > \eta_t\lambda,\\
\beta_{k,t+1} < 0 &\text{ if and only if } \beta_{k,t} - \alpha_t \nabla H_i(\boldsymbol\beta_t) < 0\\
&\text{ if and only if } \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t) - \lambda) < 0\\
&\text{ if and only if } \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t)) < -\eta_t\lambda,\\
\beta_{k,t+1} = 0 &\text{ if and only if } \beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t)) \in [-\eta_t\lambda, \eta_t\lambda]. \qquad (16)
\end{aligned}$$

The above equations can be rewritten as

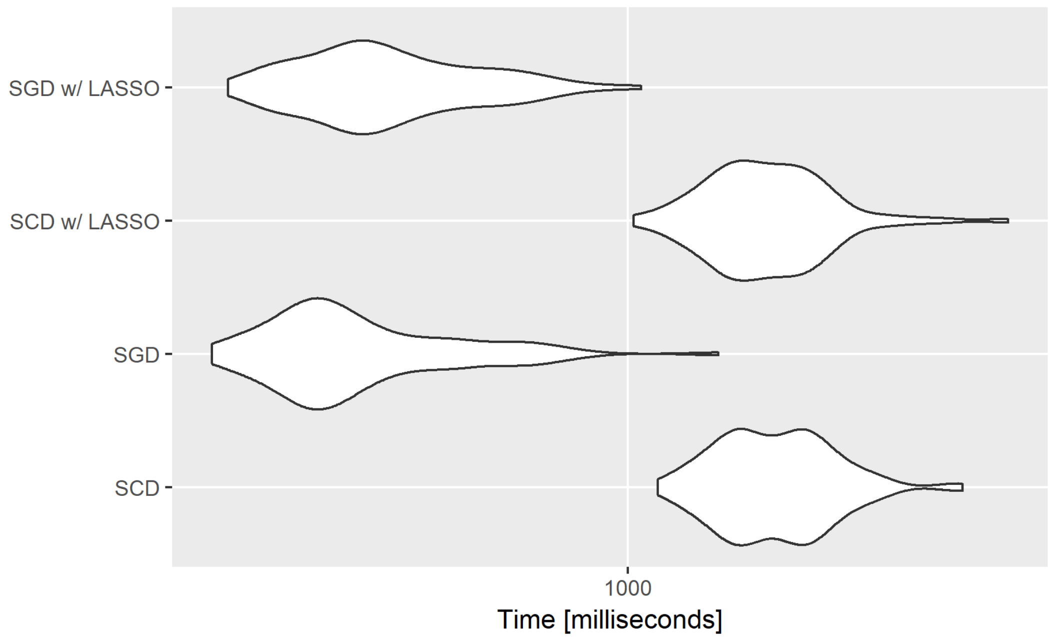
$$\boldsymbol\beta_{k,t+1} = s_{\eta_t\lambda}\left(\boldsymbol\beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t))\right). \qquad (17)$$

Thus we have the modified SCD algorithm

```
Result: $\operatorname{argmin}_{\boldsymbol\beta} H(\boldsymbol\beta)$
Set $\boldsymbol\beta = 0 \in \mathbb{R}^p$
while not converged do
    Chose $k \in \{1, \ldots, p\}$ uniformly at random.
    Let $\beta_{k,t+1} \leftarrow s_{\alpha_t\lambda}\left(\boldsymbol\beta_{k,t} - \alpha_t(-\frac{1}{N}\mathbf{X}_k^T(\mathbf{Y} - \mathbf{X}\boldsymbol\beta_t))\right)$.
end
```

In the R script found at `github.com/pcubre/cuda_examples/blob/master/thesis_linear_regression_sgd_scd_lasso.R`, we have the simple program to compare the methods. The algorithms stop when there is approximate convergence to the solution. For these examples convergence is measured relative L_2 error of 5% to the true solution or the solution from the R package glmnet. The following violin plots exams the speed of evaluating 100 iterations

Figure D.1: Timings for 100 iterations for stochastic descent methods for linear regression.



Appendix E Horseeshoe Probit

In this Appendix, we provide supplemental proofs and lemmas used to in the derivation of the Horseshoe probit model.

Definition 3 (Gamma Density)**.** *Let the shape and rate be $\alpha > 0$ and $\beta > 0$ respectively. Let $x \in (0, \infty)$. Then the density is*

$$G(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

Definition 4 (Inverse Gamma Density)**.** *Let the shape and scale be $\alpha > 0$ and $\beta > 0$ respectively. Let $x \in (0, \infty)$. Then the density is*

$$IG(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} e^{-\frac{\beta}{x}}.$$

Lemma E.1. *If X is generated by a random variable with distribution $F_X(x)$ and density $f_X(x)$, then $1/X$ has distribution $F_{\frac{1}{X}}(x) = 1 - F_X(\frac{1}{x})$ and density $f_{\frac{1}{X}}(x) = f_X(\frac{1}{x})\frac{1}{x^2}$.*

Proof. We have that $F_{\frac{1}{X}}(x) = P(\frac{1}{X} < x) = P(\frac{1}{x} < X) = 1 - F_X(\frac{1}{x})$. Therefore $f_{\frac{1}{X}}(x) = F'_{\frac{1}{X}}(x) = F'_X(\frac{1}{x})\frac{1}{x^2} = f_X(\frac{1}{x})\frac{1}{x^2}$ □

Lemma E.2. *If $X \sim G(\alpha, \beta)$, then $\frac{1}{X}$ is $IG(\alpha, \beta)$.*

Proof. Let X be generated by $G(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$. Then by the previous lemma $1/X$ is generated by $G(\frac{1}{x}|\alpha, \beta)\frac{1}{x^2} = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} e^{-\frac{\beta}{x}} = IG(\alpha, \beta)$. □

Lemma E.3. *$G(\alpha, \beta)$ is $G(\alpha, 1)/\beta$.*

Proof. Let $g(x) = x/\beta$ and $X \sim G(\alpha, 1)$. Then $g^{-1}(x) = x\beta$ and $(g^{-1})'(x) = \beta$. Thus $f_{X/\beta}(y) = f_X(x\beta)\beta = G(\alpha, \beta)$. □

Definition 5 (Exponential Density)**.** *Let the rate be $\lambda > 0$ and $x \in [0, \infty)$. Then the density is*

$$Exp(x|\lambda)\lambda e^{-\lambda x}$$

and the distribution is

$$1 - e^{-\lambda x}.$$

Lemma E.4. *$G(\alpha = 1, \beta = \lambda)$ is $Exp(\lambda)$.*

Proof. We have that $G(x|\alpha = 1, \beta = \lambda) = \frac{\lambda}{1} x^{1-1} e^{-\lambda x} = Exp(x|\lambda)$. □

Definition 6 (Cauchy Density)**.** *Let the location and scale be x_0 and $\gamma > 0$ respectively. Let $x \in (-\infty, \infty)$. Then the density is*

$$f(x|x_0, \gamma) = \frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}.$$

Lemma E.5. *Let $C^+(0,1)$ be the truncated Cauchy distribution on $\mathbb{R}^+$. Then the density is*

$$f(x|0,1) = \frac{2}{\pi(1+x^2)}.$$

Proof. We find the integrating constant for the standard Cauchy distribution over the positive reals is $\frac{\int_0^\infty \frac{dx}{1+x^2} = \tan^{-1}(x)|_0^\infty = \pi}{2}$. □

Lemma E.6. *If X has distribution $F_X(x)$ and density $f_X(x)$, then $\sqrt{X}$ has distribution $F_{\sqrt{X}}(x) = F_X(x^2)$ and density $f_{\sqrt{X}}(x) = f_X(x^2)2x$.*

Proof. The proof is strait forward as $F_{\sqrt{X}}(x) = P(\sqrt{X} < x) = P(X < x^2) = F_X(x^2)$ and $f_{\sqrt{X}}(x) = F'_{\sqrt{X}}(x) = F'_X(x^2)2x = f_X(x^2)2x$. □

Theorem E.7. *Let $\sqrt{x} \sim C^+(0,1)$ be a truncated Cauchy distribution on $\mathbb{R}^+$. Then $x|a \sim IG(\frac{\nu}{2}, \frac{\nu}{a})$, $a \sim IG(\frac{1}{2}, \frac{1}{A^2})$.*

Proof.

$$\begin{aligned}
f(x) &= \int f(x|a) f(a) da \\
&= \int \frac{\left(\frac{\nu}{a}\right)^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)} x^{-\frac{\nu}{2}-1} e^{-\frac{\frac{\nu}{a}}{x}} \frac{\left(\frac{1}{A^2}\right)^{\frac{1}{2}}}{\Gamma\left(\frac{1}{2}\right)} a^{-\frac{1}{2}-1} e^{-\frac{\frac{1}{A^2}}{a}} da \\
&= \frac{\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi}} x^{-\frac{\nu}{2}-1} \left(\frac{1}{A^2}\right)^{\frac{1}{2}} \int a^{-\frac{\nu+1}{2}-1} e^{-\frac{1}{a}\left(\frac{\nu}{x}+\frac{1}{A^2}\right)} da \\
&= \frac{\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi}} x^{-\frac{\nu}{2}-1} \left(\frac{1}{A^2}\right)^{\frac{1}{2}} \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\left(\frac{\nu}{x}+\frac{1}{A^2}\right)^{\frac{\nu+1}{2}}} \int \frac{\left(\frac{\nu}{x}+\frac{1}{A^2}\right)^{\frac{\nu+1}{2}}}{\Gamma\left(\frac{\nu+1}{2}\right)} a^{-\frac{\nu+1}{2}-1} e^{-\frac{1}{a}\left(\frac{\nu}{x}+\frac{1}{A^2}\right)} da \\
&= \frac{\Gamma\left(\frac{\nu+1}{2}\right)\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi}A} \frac{1}{x^{-\frac{\nu}{2}-1}} \frac{1}{\left(\frac{\nu}{x}+\frac{1}{A^2}\right)^{\frac{\nu+1}{2}}}
\end{aligned}$$

$$= \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi\nu}A}\left(\frac{1}{1+\frac{x}{A^2\nu}}\right)^{\frac{\nu+1}{2}}\frac{1}{x^{\frac{1}{2}}}$$

Thus we let $\nu = 1$ and $A = 1$. Then we have the density is $f(x) = \frac{1}{\pi(1+x)^2}\frac{1}{x^{\frac{1}{2}}}$. With the lemma we have that $\sqrt{X} \sim C^+(0,1)$. □

The below is Horseshoe Probit algorithm where we indication a parallel operation.

```
Result: $\boldsymbol{\beta}^{(t)}$
for $t$ iterations do
    do in parallel
        $A \leftarrow \mathbf{X}^T\mathbf{X}$.
    end
    do in parallel
        $A \leftarrow A + \text{diag}\left(1/\lambda_1^2, \dots, 1/\lambda_p^2\right)/\tau^2$.
    end
    do in parallel
        $\Sigma^{-1} \leftarrow DenseLinearSolver(Ax = I)$.
    end
    do in parallel
        $s \sim N(0,1)$ of size $n$.
    end
    do in parallel
        $\mathbf{R}s \leftarrow TriangleSolver(\Sigma^{-1}x = s)$
    end
    do in parallel
        $B \leftarrow \mathbf{X}^T\mathbf{z}^{(t-1)}$
    end
    do in parallel
        $\boldsymbol{\mu}^{(t)} \leftarrow DenseLinearSolver(\Sigma^{-1}\boldsymbol{\mu} = B)$
    end
    do in parallel
        $\boldsymbol{\beta}^{(t)} \leftarrow \mathbf{R}s + \boldsymbol{\mu}$
    end
    do in parallel
        $Y_i^* \leftarrow (2Y_i - 1)$.
        $T_i \sim TN(0, 1, \mu_i^{(t)}Y_i^*, \infty)$ by Li and Ghosh sampler.
        $z_i^{(t)} \leftarrow Y_i^*(T_i + \mu_i Y_i^*)$
    end
    do in parallel
        $U \sim Uniform(0,1)$ of size $2p+1$.
        $\lambda_i^{-2} \leftarrow -\log(U_i)/(\nu_i^{-1} + \tau^{-2}\boldsymbol{\beta}_i^{(t)}/2)$.
        $\nu_{-1} \leftarrow -\log(U_{i+p})/(1 + \lambda_i^{-2})$.
        $\epsilon \leftarrow -\log(U_{2p+1})/(1 + \tau^{-2})$.
    end
    do in parallel
        $C \leftarrow \sum(\boldsymbol{\beta}_i^{(t)})^2/\lambda_i^2$.
        $\tau^2 \sim Gamma((p+1)/2, 1)$ by Cheng sampler.
        $\tau^{-2} \leftarrow (\epsilon^{-1} + C/2)/\tau^2$.
    end
end
```
Algorithm E.1: Parallel Horseshoe Probit.

Appendix F Horseshoe Probit Code Repositories

In this Appendix, we provide the implementation of the Horseshoe Probit in two forms: R and R/Cuda. The code for the Horseshoe Probit in R at `https://github.com/pcubre/cuda_examples/blob/master/cuHo.R`.

We then have the code for the Horseshoe Probit in Cuda and R First, we have the R code to launch the CUDA at `https://github.com/pcubre/cuda_examples/blob/master/cuH.R`. The CUDA code that launches the main process at `https://github.com/pcubre/cuda_examples/blob/master/cuHorse1.cu`

Bibliography

[1] Hologic. Aptima combo 2 assay for ct/ng, 2017.

[2] R. C. H. Cheng. The generation of gamma variables with non-integral shape parameter. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 26(1):71–75, 1977.

[3] Robert Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, 14(4):436–440, 12 1943.

[4] D. Du, F.K. Hwang, and F. Hwang. *Combinatorial Group Testing and Its Applications.* Series on applied mathematics. World Scientific Publishing Company Pte Limited, 2000.

[5] Christopher R. Bilder. *Group Testing for Identification*, pages 1–11. American Cancer Society, 2019.

[6] American Red Cross. Infectious disease, hla and abo donor qualification testing, 2021.

[7] Sheila F O'Brien, Q-L Yi, W Fan, V Scalia, MA Fearon, and J-P Allain. Current incidence and residual risk of hiv, hbv and hcv at canadian blood services. *Vox sanguinis*, 103(1):83–86, 2012.

[8] Hideko Mine, Hiroyuki Emura, Masaki Miyamoto, Tsugikazu Tomono, Kiyoshi Minegishi, Hiroyuki Murokawa, Retsuji Yamanaka, Akira Yoshikawa, Kusuya Nishioka, Japanese Red Cross NAT Research Group, et al. High throughput screening of 16 million serologically negative blood donors for hepatitis b virus, hepatitis c virus and human immunodeficiency virus type-1 by nucleic acid amplification testing with specific and sensitive multiplex reagent in japan. *Journal of virological methods*, 112(1-2):145–151, 2003.

[9] Michael Schmidt, L Pichl, C Jork, MK Hourfar, V Schottstedt, FF Wagner, E Seifried, TH Müller, J Bux, and J Saldanha. Blood donor screening with cobas s 201/cobas taqscreen mpx under routine conditions at german red cross institutes. *Vox sanguinis*, 98(1):37–46, 2010.

[10] John R Papp, Julius Schachter, Charlotte A Gaydos, and Barbara Van Der Pol. Recommendations for the laboratory-based detection of chlamydia trachomatis and neisseria gonorrhoeae—2014. *MMWR. Recommendations and reports: Morbidity and mortality weekly report. Recommendations and reports/Centers for Disease Control*, 63:1, 2014.

[11] Daniel J Westreich, Michael G Hudgens, Susan A Fiscus, and Christopher D Pilcher. Optimizing screening for acute human immunodeficiency virus infection with pooled nucleic acid amplification tests. *Journal of Clinical Microbiology*, 46(5):1785–1792, 2008.

[12] Tam T Van, Joseph Miller, David M Warshauer, Erik Reisdorf, Daniel Jernigan, Rosemary Humes, and Peter A Shult. Pooling nasopharyngeal/throat swab specimens to increase testing capacity for influenza viruses by pcr. *Journal of clinical microbiology*, 50(3):891–896, 2012.

[13] Christopher R Bilder, Peter C Iwen, Baha Abdalhamid, Joshua M Tebbs, and Christopher S McMahan. Tests in short supply? try group testing. *Significance (Oxford, England)*, 17(3):15, 2020.

[14] EW Cupp, KJ Tennessen, WK Oldland, HK Hassan, GE Hill, CR Katholi, and TR Unnasch. Mosquito and arbovirus activity during 1997-2002 in a wetland in northeastern mississippi. *Journal of medical entomology*, 41(3):495–501, 2004.

[15] Paula Saá, Melanie Proctor, Gregory Foster, David Krysztof, Colleen Winton, Jeffrey M Linnen, Kui Gao, Jaye P Brodsky, Ronald J Limberger, Roger Y Dodd, et al. Investigational testing for zika virus among us blood donors. *New England Journal of Medicine*, 378(19):1778–1788, 2018.

[16] Joseph L Gastwirth. The efficiency of pooling in the detection of rare mutations. *American Journal of Human Genetics*, 67(4):1036, 2000.

[17] Charles P Woodbury, John F Fitzloff, and Stacy S Vincent. Sample multiplexing for greater throughput in hplc and related methods. *Analytical chemistry*, 67(5):885–890, 1995.

[18] Keith H. Thompson. Estimation of the proportion of vectors in a natural population of insects. *Biometrics*, 18(4):568–578, 1962.

[19] Emmanuel Barillot, Bruno Lacroix, and Daniel Cohen. Theoretical analysis of library screening using a n-dimensional pooling strategy. *Nucleic acids research*, 19(22):6241–6247, 1991.

[20] Steven F Arnold et al. Generalized group testing procedures. *The Annals of Statistics*, 5(6):1170–1182, 1977.

[21] Navneet K Dhand, Wesley O Johnson, and Jenny-Ann LML Toribio. A bayesian approach to estimate ojd prevalence from pooled fecal samples of variable pool size. *Journal of Agricultural, Biological and Environmental Statistics*, 15(4):452–473, 2010.

[22] CP Farrington. Estimating prevalence by group testing using generalized linear models. *Statistics in medicine*, 11(12):1591–1597, 1992.

[23] Christopher R. Bilder. *Group Testing for Estimation*, pages 1–11. American Cancer Society, 2019.

[24] RM Phatarfod and Aidan Sudbury. The use of a square array scheme in blood testing. *Statistics in Medicine*, 13(22):2337–2343, 1994.

[25] Hae-Young Kim, Michael G Hudgens, Jonathan M Dreyfuss, Daniel J Westreich, and Christopher D Pilcher. Comparison of group testing algorithms for case identification in the presence of test error. *Biometrics*, 63(4):1152–1163, 2007.

[26] Andrew Sterrett. On the detection of defective members of large populations. *Ann. Math. Statist.*, 28(4):1033–1036, 12 1957.

[27] M. Sobel and P. A. Groll. Group testing to eliminate efficiently all defectives in a binomial sample. *The Bell System Technical Journal*, 38(5):1179–1252, Sept 1959.

[28] FK Hwang. A generalized binomial group testing problem. *Journal of the American Statistical Association*, 70(352):923–926, 1975.

[29] Joanna Lynn Lewis, Vivian Marie Lockary, and Sadika Kobic. Cost savings and increased efficiency using a stratified specimen pooling strategy for chlamydia trachomatis and neisseria gonorrhoeae. *Sexually Transmitted Diseases*, 39(1):46–48, 2012.

[30] Christopher R Bilder, Joshua M Tebbs, and Peng Chen. Informative retesting. *Journal of the American Statistical Association*, 105(491):942–955, 2010.

[31] Christopher S McMahan, Joshua M Tebbs, and Christopher R Bilder. Informative dorfman screening. *Biometrics*, 68(1):287–296, 2012.

[32] Jacqueline M Hughes-Oliver and William H Swallow. A two-stage adaptive group-testing procedure for estimating small proportions. *Journal of the American Statistical Association*, 89(427):982–993, 1994.

[33] Milton Sobel and RM Elashoff. Group testing with a new goal, estimation. *Biometrika*, 62(1):181–193, 1975.

[34] Chao L Chen and William H Swallow. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics*, pages 1035–1046, 1990.

[35] Christopher R Bilder and Joshua M Tebbs. Empirical bayesian estimation of the disease transmission probability in multiple-vector-transfer designs. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 47(4):502–516, 2005.

[36] Stijn Vansteelandt, Els Goetghebeur, and Thomas Verstraeten. Regression models for disease prevalence with diagnostic tests on pools of serum samples. *Biometrics*, 56(4):1126–1133, 2000.

[37] Aurore Delaigle and Alexander Meister. Nonparametric regression analysis for group testing data. *Journal of the American Statistical Association*, 106(494):640–650, 2011.

[38] Aurore Delaigle, Peter Hall, and JR Wishart. New approaches to nonparametric and semiparametric regression for univariate and multivariate group testing data. *Biometrika*, 101(3):567–585, 2014.

[39] Minge Xie. Regression analysis of group testing samples. *Statistics in medicine*, 20(13):1957–1969, 2001.

[40] Boan Zhang, Christopher R Bilder, and Joshua M Tebbs. Group testing regression model estimation when case identification is a goal. *Biometrical Journal*, 55(2):173–189, 2013.

[41] Christopher S McMahan, Joshua M Tebbs, Timothy E Hanson, and Christopher R Bilder. Bayesian regression for group testing data. *Biometrics*, 73(4):1443–1452, 2017.

[42] Yan Liu, Christopher S McMahan, Joshua M Tebbs, Colin M Gallagher, and Christopher R Bilder. Generalized additive regression for group testing data. *Biostatistics*, 2020.

[43] Chase N Joyner, Christopher S McMahan, Joshua M Tebbs, and Christopher R Bilder. From mixed effects modeling to spike and slab variable selection: A bayesian regression model for group testing data. *Biometrics*, 76(3):913–923, 2020.

[44] Boan Zhang, Christopher R Bilder, and Joshua M Tebbs. Regression analysis for multiple-disease group testing data. *Statistics in medicine*, 32(28):4954–4966, 2013.

[45] Roche. cobas® taqscreen mpx test, 2009.

[46] Roche. cobas® taqscreen mpx test, version 2.0, 2019.

[47] Joshua M Tebbs, Christopher S McMahan, and Christopher R Bilder. Two-stage hierarchical group testing for multiple infections with application to the infertility prevention project. *Biometrics*, 69(4):1064–1073, 2013.

[48] Peijie Hou, Joshua M Tebbs, Christopher R Bilder, and Christopher S McMahan. Hierarchical group testing for multiple infections. *Biometrics*, 73(2):656–665, 2017.

[49] Peijie Hou, Joshua M Tebbs, Dewei Wang, Christopher S McMahan, and Christopher R Bilder. Array testing for multiplex assays. *Biostatistics*, 21(3):417–431, 2020.

[50] Jacqueline M Hughes-Oliver and William F Rosenberger. Efficient estimation of the prevalence of multiple rare traits. *Biometrika*, 87(2):315–327, 2000.

[51] T Verstraeten, B Farah, Luc Duchateau, and R Matu. Pooling sera to reduce the cost of hiv surveillance: a feasibility study in a rural kenyan district. *Tropical Medicine & International Health*, 3(9):747–750, 1998.

[52] Juexin Lin, Dewei Wang, and Qi Zheng. Regression analysis and variable selection for two-stage multiple-infection group testing data. *Statistics in medicine*, 38(23):4519–4533, 2019.

[53] Christopher R Bilder, Joshua M Tebbs, and Christopher S McMahan. Informative group testing for multiplex assays. *Biometrics*, 75(1):278–288, 2019.

[54] Christopher R Bilder, Joshua M Tebbs, and Christopher S McMahan. Informative array testing with multiplex assays. *Statistics in medicine*, 2021.

[55] James H. Albert and Siddhartha Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679, 1993.

[56] Sean M O'Brien and David B Dunson. Bayesian multivariate logistic regression. *Biometrics*, 60(3):739–746, 2004.

[57] Centers for Disease Control and Prevention. Sexually transmitted disease surveillance 2019, 2021.

[58] E Scott Larsen and David McAllister. Fast matrix multiplies using graphics hardware. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, pages 55–55, 2001.

[59] Martin Rumpf and Robert Strzodka. Level set segmentation in graphics hardware. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, volume 3, pages 1103–1106. IEEE, 2001.

[60] Mark J Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Physically-based visual simulation on graphics hardware. In *Graphics Hardware*, volume 2002, pages 1–10, 2002.

[61] Theodore Kim and Ming C Lin. Visual simulation of ice crystal growth. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 86–97. Citeseer, 2003.

[62] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[63] Adám Moravánszky. Dense matrix algebra on the gpu. *ShaderX2: Shader Programming Tips and Tricks with DirectX*, 9:352–380, 2003.

[64] Nico Galoppo, Naga K Govindaraju, Michael Henson, and Dinesh Manocha. Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 3–3. IEEE, 2005.

[65] Adam L Beberg, Daniel L Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S Pande. Folding@ home: Lessons from eight years of volunteer distributed computing. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[66] NVIDIA. Nvidia cuda compute unified device architecture programming guide, 2007.

[67] Alexander Terenin, Shawfeng Dong, and David Draper. Gpu-accelerated gibbs sampling. *arXiv preprint arXiv:1608.04329*, 2016.

[68] Peng Chen, Joshua M Tebbs, and Christopher R Bilder. Group testing regression models with fixed and random effects. *Biometrics*, 65(4):1270–1278, 2009.

[69] Christopher S McMahan, Joshua M Tebbs, and Christopher R Bilder. Regression models for group testing data with pool dilution effects. *Biostatistics*, 14(2):284–298, 2013.

[70] Hae-Young Kim and Michael G Hudgens. Three-dimensional array-based group testing algorithms. *Biometrics*, 65(3):903–910, 2009.

[71] Joseph L Gastwirth and Wesley O Johnson. Screening with cost-effective quality control: potential applications to hiv and drug testing. *Journal of the American Statistical Association*, 89(427):972–981, 1994.

[72] Md S Warasi, Joshua M Tebbs, Christopher S McMahan, and Christopher R Bilder. Estimating the prevalence of multiple diseases from two-stage hierarchical pooling. *Statistics in medicine*, 35(21):3851–3864, 2016.

[73] Eugene Litvak, Xin M Tu, and Marcello Pagano. Screening for the presence of a disease by pooling sera samples. *Journal of the American Statistical Association*, 89(426):424–434, 1994.

[74] Lawrence M Wein and Stefanos A Zenios. Pooled testing for hiv screening: capturing the dilution effect. *Operations Research*, 44(4):543–569, 1996.

[75] NVIDIA. Nvidia's next generation cuda compute architecture: Fermi, 2009.

[76] NVIDIA. Nvidia tesla p100, 2016.

[77] NVIDIA. Nvidia turing gpu architecture, 2018.

[78] NVIDIA. CUDA toolkit documentation v8.0, 2020.

[79] NVIDIA. CUDA toolkit documentation v8.0, 2020.

[80] NVIDIA. NVCC gpu compilation, 2020.

[81] NVIDIA. cuBLAS API, 2020.

[82] Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11(Nov):3183–3234, 2010.

[83] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[84] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[85] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for l1-regularized loss minimization. *Journal of Machine Learning Research*, 12(Jun):1865–1892, 2011.

[86] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[87] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[88] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[89] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[90] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[91] Carlos M Carvalho, Nicholas G Polson, and James G Scott. The horseshoe estimator for sparse signals. *Biometrika*, 97(2):465–480, 2010.

[92] Yifang Li and Sujit K. Ghosh. Efficient sampling methods for truncated multivariate normal and student-t distributions subject to linear inequality constraints. *Journal of Statistical Theory and Practice*, 9(4):712–732, 2015.

[93] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.