UNIVERSITY OF CALIFORNIA
Los Angeles

# Scalable Methods for Survival Analysis using Massive Observational Data

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computational Medicine

by

Jianxiao Yang

2023

ABSTRACT OF THE DISSERTATION

Scalable Methods for Survival Analysis using Massive Observational Data

by

Jianxiao Yang
Doctor of Philosophy in Computational Medicine
University of California, Los Angeles, 2023
Professor Marc A. Suchard, Chair

The emerging observational health data, such as electronic health records and administrative claims, provide a rich resource for learning about treatment effects and risks. However, computational challenges arise when fitting statistical models to such large-scale and high-dimensional data. In this dissertation, I employ parallel computing techniques to address the computational bottlenecks associated widely used statistical models in observational studies. First, I present a novel parallel scan algorithm to scale up the Cox proportional hazards model and the Fine-Gray model. This advancement significantly accelerates the execution of large-scale comparative effectiveness and safety studies involving millions of patients and thousands of patient characteristics by an order of magnitude. Second, I apply an efficient parallel segmented-scan algorithm to accelerate the computational intensive parts shared by the stratified Cox model, the Cox model with time-varying covariates, and the Cox model with time-varying coefficients. This innovation enables efficient large-scale and high-dimensional Cox modeling with stratification or time-varying effect, delivering an order of magnitude speedup over traditional central processing unit-based methods. Third, I introduce a memory-efficient approach for fitting pooled logistic regression models with massive sample-size data. This approach offers a valuable tool, allowing for pooled logistic regression analysis on massive sample sizes, even when computational resources are limited. I have implemented all of the above work in the open-source `R` package `Cyclops`.

The dissertation of Jianxiao Yang is approved.

Andrew J. Holbrook

Eric M. Sobel

Hua Zhou

Marc A. Suchard, Committee Chair

University of California, Los Angeles

2023

*To my parents*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I would like to thank my adviser, Marc Suchard, who introduced me to the fascinating world of statistical computing when I was still a medical student. His outstanding guidance and unwavering patience meant a lot to me through this tough but rewarding journey. I also want to thank the other members of my dissertation committee, Hua Zhou, Eric Sobel, and Andrew Holbrook, for their valuable perspectives and support. Hua taught me the basics of scientific computing and allowed me to do my preceptorship in his class, for which I am very thankful. Eric was always kind and supportive during my PhD studies, which gave me strength. I want to thank Andrew for his help when I was struggling with GPU parallel computing at the beginning and for always being encouraging. Lastly, I want to thank Janet Sinsheimer, who was on my committee, for her warm introduction and kind encouragement, beginning even before I started my PhD. I am deeply saddened that she could not see my dissertation completed.

I would like to express my gratitude for the friendships I have built at UCLA. My cohort in the Biomath Department has been great companions on this academic journey. I am also extremely appreciative of everyone in Marc's lab for all their help and support, both emotionally and technically.

I am also grateful for the financial support I received during my PhD. I was supported by the Burroughs Wellcome Fund Inter-school Training Program in Chronic Diseases Training Grant for my first year and graduate student researcher funds from Marc Suchard for the rest of the time.

Finally, I would like to express my profound gratitude for the unconditional love and wholehearted support from my family. I would never have accomplished this without them.

# VITA

2013–2018 B.M. Basic Medical Sciences, Peking University, Beijing, China.

2018–2019 Burroughs Wellcome Fund Inter-school Training Program in Chronic Diseases Training Grant, University of California, Los Angeles

2019–present Graduate Student Researcher, Department of Human Genetics, University of California, Los Angeles

# PUBLICATIONS

**Yang, J.**, Schuemie, M. J., Ji, X., and Suchard, M. A. (2023). Massive parallelization of massive sample-size survival analysis. *Journal of Computational and Graphical Statistics*, (just-accepted), 1-23.

**Yang, J.**, Schuemie, M. J., and Suchard, M. A. (2023). Efficient GPU-accelerated fitting of observational health-scaled stratified and time-varying Cox models. *arXiv preprint arXiv:2310.16238*.

Kim, Y., Tian, Y., **Yang, J.**, Huser, V., Jin, P., Lambert, C. G., ... , and Suchard, M. A. (2020). Comparative safety and effectiveness of alendronate versus raloxifene in women with osteoporosis. *Scientific Reports*, 10(1), 11115.

Schuemie, M. J., Cepeda, M. S., Suchard, M. A., **Yang, J.**, Tian, Y., Schuler, A., ... , and Hripcsak, G. (2020). How confident are we about observational findings in health care: a benchmark study. *Harv Data Sci Rev*, 2(1), 10.

# CHAPTER 1

# Introduction

Observational health studies have thrived in the last few decades as digital health technology advances. Although observational health data are collected for non-research purpose, they provide rich information for understanding the healthcare process. For instance, electronic health records (EHRs) are collected for patient care during routine medical practice and contain rich information about patient conditions necessary for medical practice. Administrative claims, such as those filled through public and private health insurers, capture diverse patient populations from communities historically excluded from clinical trials (Schneeweiss and Avorn, 2005).

Despite the obvious non-random collection of observational healthcare data, analytics using them often possess many advantages compared to randomized controlled trials (RCTs). Observational data lower barriers for conducting health data analytic since there is no need to spend time on cohort recruitment and data collection. Observational health data more realistically depict the clinical landscape owing to their massive scale. Thus they are more suitable for detecting rare clinical events and measuring real-world treatment effectiveness (Flay et al., 2005).

Computational challenges, however, arise when using observational data. The massive scales of the databases that hold EHRs and claims data offer more power for statistical analyses but are more computationally demanding. Typical EHRs and claims contain millions of individuals with thousands of demographic and clinical covariates. Statistical models used in health analytic become computationally expensive and often stall with such large sample sizes and high dimension. Specifically, typical log-likelihood computation for semi-parametric models in survival analysis (such as the Cox proportional hazards model (Cox, 1972) for right-censored data and the Fine-Gray proportional

subdistribution hazards model (Fine and Gray, 1999) for competing risk data) traditionally scale in terms of $\mathcal{O}(N^2)$ operations, which quickly becomes computationally infeasible as the sample size grows. Incorporating time-varying effects further necessitates a larger dataset since we introduce multiple time points for each individual. This, in turn, implies the need to store and process a significantly larger amount of data. Such models, including the Cox model with time-varying covariates, Cox model with time-varying coefficients, and pooled logistic models, introduce additional computational complexity in terms of data manipulation, likelihood estimation, and survival function calculations.

In this dissertation, I present three efficient statistical computing approaches, including state-of-the-art parallel computing techniques on graphics processing unit (GPUs), to tackle several computational challenges in observational heath data analysis. I focus on widely used statistical models for exploring time-to-event data and longitudinal data. I harness cutting-edge computing devices and novel parallel algorithms to accelerate these statistical models when applied to large-scale data. Additionally, I develop appropriate data manipulation techniques to optimize memory usage for these models.

Chapter 2 reviews the basics of counter-factual based causal inference in the context of observational study. Chapters 3, 4 and 5 each represent independent projects and can be read as standalone studies.

In Chapter 3, I present a novel parallel algorithm to address the computational bottlenecks of massive sample-size survival analysis. This chapter introduces time- and memory- efficient single-pass parallel scan algorithms for Cox proportional hazards models and forward-backward parallel scan algorithms for Fine-Gray models for analysis with and without a competing risk. Simulation and real-word experiments in this chapter demonstrate that GPUs accelerate the computation of fitting these complex models in large databases by orders of magnitude as compared to traditional multi-core CPU parallelism. This implementation opens up the possibility of conducting efficient large-scale observational studies involving millions of patients and thousands of patient characteristics.

In Chapter 4, I propose an efficient parallel segmented-scan algorithm to enhance the scalability of three important variants of the Cox model: the stratified Cox model, the Cox model with time-varying covariates, and the Cox model with time-varying coefficients. This chapter introduces how to convert the computationally demanding parts of the stratified Cox model into segmented operations. Furthermore, I apply an efficient parallel segmented-scan algorithm to alleviate the data communication challenges inherent in naive segmented computation. This chapter also details the transformation of the Cox model with time-varying covariates into a stratified version and illustrates how to convert a time-varying coefficient into a set of time-varying covariates. These transformations allow these models to benefit from the aforementioned algorithmic enhancements. I demonstrate that this GPU implementation significantly accelerates the computation of fitting these complex models on large-scale and high-dimensional simulated and real-world data by an order-of magnitude compared to a similarly optimized CPU implementation.

In Chapter 5, I present a memory-efficient approach for fitting pooled logistic regression models with massive sample-size data. This chapter outlines the necessary data manipulation and likelihood derivation required to fit a pooled logistic regression without redundantly including the baseline data for every time point. This implementation enables the execution of large-scale observation studies using pooled logistic regression model even with limited computing resources.

Finally, Chapter 6 explores future directions and potential applications related to the models discussed in the previous chapters. I outline a framework to evaluate whether including a competing risk under the Fine-Gray model for comparative effectiveness estimation reduces systematic error as compared to the usual Cox model in the large-scale setting.

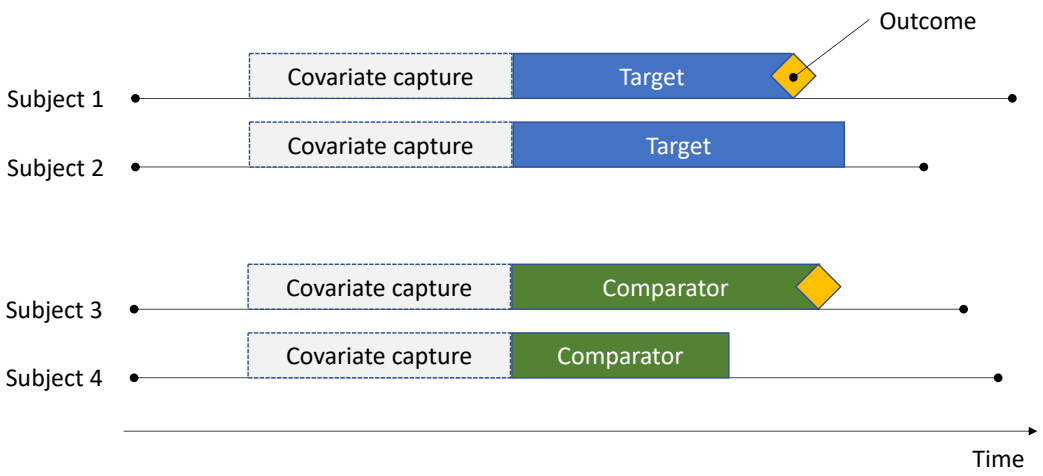
# CHAPTER 2

# Background: Comparative Effectiveness and Safety Study using Large-scale Observational Data

## 2.1 Causal inference based on counter-factuals

In clinical and epidemiological studies, we often seek to make inferences about the efficacy or risk of one treatment compared to a placebo or another treatment regarding a specific outcome. In practice, we can only observe the patient's outcome given the received treatment, but not the potential outcome given the counter-factual treatment (Little and Rubin, 2000). Randomized control trials (RCTs) can provide this type of inference within the potential outcome framework, as randomization ensures that patients in different treatment groups are interchangeable (Hernán and Robins, 2006). Hence, RCTs serve as the gold standard for effectiveness studies while also providing information about the risk of adverse events during the initial exposure period. However, even large RCTs contain a limited number of patients compared to the prescribed population after the product is approved, and they may be underpowered for rare events and long-term effect estimation (Ryan et al., 2013). While the availability of electronic health records and claims offers new opportunities to study medication efficacy and risk in large populations, these observational data lack the randomization required for making inferences under the counterfactual framework. Therefore, in this dissertation, I employ the new-user cohort design and propensity score-based cohort balancing to achieve a form of randomization when working with observational data.

Figure 2.1: The new-user cohort design.




## 2.2 New-user cohort design

The new-user cohort design is a vital method employed in observational health studies to assess the effectiveness and safety of medication. It attempts to emulate a RCT (Hernán and Robins, 2016) with two groups that are prospectively followed from the initiation of treatment using observational data. Figure 2.1 illustrates this cohort design. The target cohort comprises new users of the target treatment over a specified time period, while the comparator cohort contains the new-users of another treatment. Both cohorts contain only those patients who have at least one diagnosis record for the target treatment indication. The endpoints are either the time when the outcome occurs or the end of the observation time in the database. We typically restrict the inclusion criteria to patients with a washout period before the initial exposure to mitigate the influence of previous usage (Ray, 2003). The above definition closely resembles the way in which data are collected in RCTs, except for the treatment is not assigned by randomization. To address the issue of non-randomization when using observational data, we also capture baseline covariates, including demographics, medical conditions, drug usage, etc., before the treatment initiation, for propensity score estimation and adjustment.

## 2.3 Propensity score estimation and adjustment

The propensity score (PS) is a fundamental statistical tool used to address the issue of non-randomization in observational studies. It is defined as the probability that a patient is assigned to one treatment over the other based on observed baseline characteristics. Therefore, propensity scores are often estimated through logistic regression, where the binary outcome is the treatment assignment, and the covariates include baseline patient characteristics captured before treatment initiation. In the context of high-dimensional observational data, we usually construct an $\ell_1$-regularized logistic regression model using all available baseline covariates to achieve cohorts balance (Tian et al., 2018).

After estimating the propensity scores for each patient in both cohorts, we can use various PS-based methods to reduce bias due to non-randomization by balancing covariates between treatment groups. One such method is propensity score matching (Rosenbaum and Rubin, 1983), which forms matched sets of patients in the target and comparator cohorts with similar propensity score values. Subsequently, we only compare patients within the same matched set in the outcome model, resulting in a stratified outcome model. Stratification based on propensity scores (Rosenbaum and Rubin, 1984) involves grouping patients into different strata, often around five, based on their propensity scores. This method, again, requires a stratified outcome model to compare patients within the same strata. Inverse probability of treatment weighting (IPTW) (Rosenbaum and Rubin, 1983) assigns a sample weight to each patient based on their propensity score. This weight represents the proportion of patients with the same baseline characteristics in the population. Consequently, the outcome model becomes weighted.

# CHAPTER 3

# Massive Parallelization of Massive Sample-Size Survival Analysis

## 3.1 Introduction

Increasing accessibility of large-scale observational health data provides rich opportunities to study comparative effectiveness and safety of medical products, but also poses unprecedented challenges. Typical administrative claims and electronic health record (EHR) databases now follow tens to hundreds of millions of individuals (Hripcsak et al., 2016) with tens of thousands of possible health conditions, drugs and procedures occurring over decades of patient lives (Suchard et al., 2013). The massive scales of these databases offer more power for statistical analyses to learn about the effects of these products on health outcomes but also bring taxing computational burden.

The increasing complexity of common statistical models further exacerbated this big-data problem. For instance, the Cox proportional hazards model and the Fine-Gray model are widely applied in comparative effectiveness and safety studies. The computational complexity of likelihood evaluation for the Cox model and the Fine-Gray model naively grows quadratically with sample size. In addition, some form of regularization (Madigan et al., 2010) is often needed to achieve parsimonious model selection when entertaining hundreds to thousands of patient characteristics. This regularization typically requires computationally intensive cross-validation to select the optimal regularization parameter(s), further straining limited computational resources.

One can distribute computationally intensive work to the cloud or dedicated clusters that house multiple central processing units (CPUs) across separate compute nodes that are linked together

loosely through an Ethernet or InfiniBand network (Holbrook et al., 2021). However, for problems that require communication between nodes, communication latency may become a severe bottleneck. Since fitting survival models generally requires iterative algorithms, the communication latency costs often overpower the gains from the parallelized work within each iteration. To minimized communications between CPUs, we often follow coarse-scale parallelism, where programs are split into small number of large tasks (Barney et al., 2010). However, this may result in load imbalance and only achieve "embarrassingly parallel" benefits (Suchard et al., 2010). Furthermore, CPU clusters and the cloud can be costly and arcane for many clinical researchers.

Multi-core CPU parallelization is another choice for distributing intensive computational works, as modern CPU chip usually consists of 2 to 18 or more cores that can run independently. One limitation of this architecture is that all cores share a single "memory bandwidth", the amount of data that can be written to or read from memory in a given period of time (Holbrook et al., 2021). This approach also suffers from the limited number of cores. Thus, multi-core CPU parallelization is often only useful for modest-scale problems.

In contrast, graphics processing units (GPUs) offer a relatively inexpensive and efficient approach for speeding up fine-scale parallel computation. In fine-scale parallelism, we decompose programs into a large number of small tasks to facilitate load balancing and achieve much higher level of parallelism than coarse grain approaches (Barney et al., 2010). The GPU is an ideal device for fine-scale parallelism because (1) it consists of hundreds to thousands of compute cores and (2) the shared memory architecture of a GPU's coupled thread block allows for threads to communicate and share data among each other at a very high speed. Finally, GPUs are often conveniently available on standard laptops and desktop computers and can be externally connected to a personal computer.

Accelerating statistical computing via GPUs is an emerging discipline. As examples, Zhou et al. (2010) attain 100-fold speedups with GPUs in high-dimensional optimization. Suchard et al. (2013) demonstrate that GPU parallelization achieves one to two orders of magnitude improvement over CPUs for a Bayesian self-controlled case series model. Beam et al. (2016) accelerate Hamiltonian Monte Carlo using GPUs by efficient evaluation of their probability kernel and its gradient. Terenin

et al. (2019) demonstrate that Gibbs sampling can run orders of magnitude faster than on a CPU. Holbrook et al. (2021) apply GPU computing to a Bayesian multidimensional scaling model and deliver more than 100-fold speedups over serial calculations. Ko et al. (2022) explore the GPU parallelization for proximal gradient descent on modest-sized $\ell_1$ regularized dense Cox regression using PyTorch. In this paper, we leverage GPU parallelization to the Cox model and the Fine-Gray model through innovative algorithmic mapping that play to the GPU's strengths for accelerating observational studies utilizing massive healthcare data. Specifically, we identify the computational bottleneck of the Cox model and the Fine-Gray model and take advantage of the cutting-edge GPU-accelerated library CUB (Merrill and Garland, 2016) that navigate this bottleneck. We further implement our GPU advances in the easy-to-use `R` package `Cyclops` (Suchard et al., 2013). Our implementation supports a sparse data format, considering that observational healthcare data are generally sparse; the vast majority of patient characteristics are encoded as the presence or absence of some clinical condition, drug exposure, medical procedure or laboratory measurement above or below a cutoff point within given time-frames. We finally demonstrate that our GPU implementation accelerates the computation of fitting these complex models by order-of-magnitude compared to a similar CPU implementation on multiple GPUs and CPUs with different technical specifications.

## 3.2 Methods

### 3.2.1 Cox proportional hazards model

We first establish notation under a typical survival analysis setting. Suppose there are $N$ observed individuals available in a study. For individual $i = 1, \ldots, N$, let $Y_i = \min(T_i, C_i)$ represent their survival time, where $T_i$ and $C_i$ are the time-to-event time and right-censoring time, respectively. Let $\delta_i$ be the indicator variable such that $\delta_i = 1$ if we observe the event occurrence of individual $i$ and $\delta_i = 0$ if the individual $i$ is censored. Let $\mathbf{x}_i$ be a $P$-dimensional vector of time-independent covariates for individual $i$. The survival data then consist of triplets $\{Y_i, \delta_i, \mathbf{x}_i\}_{i=1}^n$.

The cumulative distribution function of survival times gives the probability that the event of

interest has occurred by time $t$, i.e. $F(t|\mathbf{x}) = \Pr(T \leq t|\mathbf{x})$. The survival function gives the probability that the event has not occurred by time $t$, i.e. $S(t) = \Pr(T > t)$. Then we define the hazard function of time-to-event time as:

$$h(t) = \lim_{\Delta t \to \infty} \frac{\Pr(t \leq T < t + \Delta t | T \geq t)}{\Delta t} = \frac{f(t)}{S(t)}, \tag{3.1}$$

where $f(t) = \frac{d}{dt}F(t)$ is the density function of random variable $T$.

Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_P)^\top$ be a $P$-dimensional vector of unknown, underlying model parameters. Assuming survival times $y_1, y_2, \ldots, y_N$ are independent and identically distributed from density $f(y|\boldsymbol{\beta})$ and $\boldsymbol{\beta}$ parameterized the survival function $S(y|\boldsymbol{\beta})$, Cox (1972) proposes a semi-parametric hazard function as the product of an unspecified baseline hazard function $h_0(y_i|\boldsymbol{\beta})$ and an exponential link function of covariates:

$$h(y_i|\boldsymbol{\beta}) = h_0(y_i) \exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right). \tag{3.2}$$

Parameter estimation of the Cox proportional hazards model follows from the log-partial likelihood

$$l_{\text{partial}}(\boldsymbol{\beta}) = \sum_{i=1}^{N} \delta_i \left\{ \mathbf{x}_i^\top \boldsymbol{\beta} - \log \left[ \sum_{r \in R_1(Y_i)} \exp\left(\mathbf{x}_r^\top \boldsymbol{\beta}\right) \right] \right\}, \tag{3.3}$$

where $R_1(Y_i) = \{r : y_r \geq Y_i\}$ denotes the set of subjects who are "at risk" for event at time $Y_i$. Then one often estimates $\boldsymbol{\beta}$ by its maximum log-partial likelihood estimator $\hat{\boldsymbol{\beta}}_{\text{mple}} = \arg\max_\beta \{l_{\text{partial}}(\boldsymbol{\beta})\}$.

This log-partial likelihood has a complicated form due to the repeated calculation of the risk sets, and thus brings a high computation burden. In practice, we need to keep track of the sum of many terms for each subject that usually requires $\mathcal{O}(N^2)$ number of operations and will explode quickly as $N$ increases (Kawaguchi et al., 2021). This computational burden prevents traditional model fitting as there can be millions of observations available in observational health databases.

### 3.2.2 Fine-Gray model

The Fine-Gray model (Fine and Gray, 1999) generalizes the Cox proportional hazards model for competing risks time-to-event data that consist of more than one type of event. Unlike the standard survival analysis setting such as under the Cox model where individuals are only susceptible to one type of event during follow-up, competing risks arise when individuals can experience more than one type of event and the occurrence of one type of event will either prevent the occurrence or change the underlying risk of the others (Noordzij et al., 2013). For individual $i$, competing risks data inherit the definition of event time $T_i$, possible right censoring time $C_i$, event indicator $\delta_i$, and covariates $\mathbf{x}_i$ from the standard survival data setting, and additionally include an event type variable $\varepsilon_i$. Without loss of generality, we assume there are two types of events, where $\varepsilon_i = 1$ indicates that $T_i$ refers to the time of primary event and $\varepsilon_i = 2$ indicates the competing risk event.

The cumulative incidence function (CIF) for competing risks data describes the probability of failing from the event of interest before the other possible (competing) event. Under the above setting when $\varepsilon = 1$ indicating the event of interest, the CIF and hazards function are defined as:

$$F_1(t|\mathbf{x}) = \Pr(T \leq t, \varepsilon = 1|\mathbf{x}), \text{ and} \tag{3.4}$$

$$h_1(t|\mathbf{x}) = \lim_{\Delta t \to \infty} \frac{\Pr\{t \leq T < t + \Delta t, \varepsilon = 1 | \{T \geq t\} \cup (\{T < t\} \cap \{\varepsilon \neq 1\}), \mathbf{x}\}}{\Delta t} = -\frac{d}{dt} \log\{1 - F_1(t|\mathbf{x})\}. \tag{3.5}$$

To model the covariate effects on $F_1(t|\mathbf{x})$, Fine and Gray (1999) propose the proportional subdistribution hazards function:

$$h_1(y_i|\boldsymbol{\beta}) = h_{10}(y_i) \exp(\mathbf{x}_i^\top \boldsymbol{\beta}), \tag{3.6}$$

where $h_{10}(y_i|\boldsymbol{\beta})$ is an unspecified baseline subdistribution hazard, and $\boldsymbol{\beta}$ is a P-dimensional vector of model parameters.

Parameter estimation of the Fine-Gray subdistribution proportional hazards model follows from

the log-pseudo likelihood

$$l_{\text{pseudo}}(\boldsymbol{\beta}) = \sum_{i=1}^{N} I(\delta_i \varepsilon_i = 1) \left\{ \mathbf{x}_i^\top \boldsymbol{\beta} - \log \left[ \sum_{r \in R(Y_i)} \hat{w}_r(Y_i) \exp\left( \mathbf{x}_r^\top \boldsymbol{\beta} \right) \right] \right\}, \tag{3.7}$$

where $R(Y_i) = \{r : (y_r \geq Y_i) \text{ or } (y_r < Y_i \text{ and } \varepsilon_r \neq 1)\}$ denotes the risk set at time $Y_i$, and $\hat{w}_r(t)$ is a time-dependent weight based on an inverse probability of censoring weighting (IPCW) technique (Robins and Rotnitzky, 1992). Assuming two event types exist, the risk set $R(Y_i)$ contains two disjoint set: $R_1(Y_i) = \{r : y_r \geq Y_i\}$ and $R_2(Y_i) = \{r : y_r < Y_i \cap \varepsilon_r = 2\}$, where $R_1(Y_i)$ is the regular risk set that includes the observations that have an observed time equalling or after $Y_i$ and $R_2(Y_i)$ includes the observations that have observed the competing event before time $Y_i$. Here we follow the design of weighted score functions for incomplete data with right censoring in Fine and Gray (1999) for unbiased estimation from the complete-data partial likelihood. The time-dependent weights are defined as $\hat{w}_r(t) = I(C_r \geq \min(T_r, t))\hat{G}(t)/\hat{G}(\min(Y_r, t))$, where $\hat{G}(t)$ is the Kaplan-Meier estimate of $G(t)$ and $G(t) = \Pr(C > t)$ is the survival function of censoring variable $C$. Combined, one can estimate $\boldsymbol{\beta}$ by its maximum log-partial likelihood estimator $\hat{\boldsymbol{\beta}}_{\text{mple}} = \arg\max_{\beta} \{l_{\text{pseudo}}(\boldsymbol{\beta})\}$.

### 3.2.3 Statistical regularization

Observational healthcare datasets often include a large number of patient characteristics. For example, administrative claims usually contain information on all drug prescriptions, medical procedures and diagnosis codes for patients, and EHRs generally further contain demographics, medical history notes, laboratory results, and other health status indications (Madigan et al., 2014). A statistical regularization approach is typical in such high-dimensional data analysis to avoid overfitting. We can conveniently add a penalty $\pi(\boldsymbol{\beta})$ for $\boldsymbol{\beta}$ to the log-partial likelihood of Cox model or the log-pseudo likelihood of Fine-Gray model and estimate $\boldsymbol{\beta}$ through these joint penalized likelihoods to achieve regularization.

For $\ell_1$ regularization that shrinks many components of $\boldsymbol{\beta}$ to be zero, we define a separable penalty

for each dimension $\beta_j$ in $\boldsymbol{\beta}$ through

$$\pi(\boldsymbol{\beta}) = \sum_j \pi(\beta_j|\gamma_j) = -\sum_j \gamma_j|\beta_j|, \tag{3.8}$$

where the tuning parameters $\gamma_j$ control the degree of regularization for each dimension. Similarly, one may employ an $\ell_2$ penalty on the dimensions of $\boldsymbol{\beta}$, such that:

$$\pi(\boldsymbol{\beta}) = \sum_j \pi(\beta_j|\tau_j) = -\sum_j \frac{\beta_j^2}{2\tau_j}. \tag{3.9}$$

Usually one assumes $\gamma_j = \gamma \; \forall j$ and $\tau_j = \tau \; \forall j$, and we choose $\gamma$ or $\tau$ through cross-validation.

Note that statistical inference in the context of regularization remains a challenge. Various standard errors estimators based on the non-parametric bootstrap have been proposed (Casella et al., 2010; Chatterjee and Lahiri, 2011), but an approach that is both computationally efficient and statistically valid still remains out of reach. Since we are focusing on computational bottleneck in this paper, we decide to follow the standard practice of regularization in large-scale and high-dimensional observational health studies (Mueller-Using et al., 2016; Shortreed and Ertefaie, 2017; Bramante et al., 2021) despite the limitations of regularization.

### 3.2.4 Maximum likelihood estimation using cyclic coordinate descent

Following Genkin et al. (2007) and Mittal et al. (2014), we exploit a cyclic coordinate descent (CCD) algorithm to reduce the high-dimensional penalized likelihood optimization down to a large set of simple one-dimensional optimizations (Wu et al., 2008). This method cycles through each covariate and updates it using a Newton step while holding all other covariates as constants. The advantage of CCD is it only requires the calculation of scalar gradients and Hessians and avoids the inversion of large Hessian matrices in high-dimensional regression.

Specifically, for each one-dimensional optimization problem, we pick the $\beta_j^{(new)}$ by maximizing $g(\beta_j) = l(\beta_j) + \pi(\beta_j)$ while holding all other $\beta_j$'s unchanged. The second-order Taylor approxima-

tion of the penalized log likelihood at current $\beta_j$ is:

$$g(z) \approx g(\beta_j) + g'(\beta_j)(z-\beta_j) + \frac{1}{2}g''(\beta_j)(z-\beta_j)^2. \tag{3.10}$$

Then the new estimate $\beta_j^{(\text{new})}$ falls out

$$\beta_j^{(\text{new})} = \beta_j + \Delta\beta_j = \beta_j - \frac{g'(\beta_j)}{g''(\beta_j)}. \tag{3.11}$$

We employ a trust region approach similar to Genkin et al. (2007) to restrict the the step size so that the quadratic remains a reasonable approximation to the objective and improve convergence. In particular, we update $\beta_j$ during iteration $k$ by

$$\Delta\beta_j^{(k)} = -\frac{g'(\beta_j^{(k)})}{g''(\beta_j^{(k)})}, \text{ and} \tag{3.12}$$

$$\beta_j^{(k+1)} = \beta_j^{(k)} + \text{sgn}(\Delta\beta_j^{(k)}) \min\{|\Delta\beta_j^{(k)}|, \Delta_j^{(k)}\}, \tag{3.13}$$

where we update the trust region half-width as $\Delta_j^{(k+1)} = \max\{2|\Delta\beta_j^{(k)}|, \Delta_j^{(k)}/2\}$, starting with $\Delta_j^{(0)} = 1$.

Note that both the negated log likelihood of the Cox model and the Fine-Gray model are convex in $\boldsymbol{\beta}$, as well as the $\ell_1$ and $\ell_2$ penalty terms. Although the $\ell_1$-norm is nondifferentiable at origin, we can follow the approach of Wu et al. (2008) to compute the directional derivatives along each forward and backward coordinate direction for our objective function. In particular, we compute the directional derivatives in both directions by plugging in $\text{sgn}(\beta_j) = +1$ and $\text{sgn}(\beta_j) = -1$ when $\beta_j^{(k)} = 0$, and only update $\beta_j$ in a direction when the directional derivative is negative, otherwise we keep $\beta_j^{(k+1)} = 0$. Since the objective function is convex, it is impossible for both directional derivatives to be negative, but either direction with a negative directional derivative will result in a successful update. Although we lack of rigorous proof of convergence when employing a trust region, as the induced step sizes fail to meet the strict convergence conditions for this optimization

problem (Xu and Yin, 2017), we have not observed this issue in our work.

### 3.2.5 Massive parallelization on GPUs

Parallelization through clusters and multi-core CPUs exhibits a number of drawbacks that makes these devices ill-suited for massive survival analysis, as discussed in the Introduction. As such, this paper exploits massive parallelization on GPUs through new fine-scale algorithm decomposition for speeding up large-scale computations. Here we begin with an overview on GPU computing and summarize its main strengths and weaknesses.

The modern GPU contains an array of multithreaded streaming multiprocessors (SMs), where hundreds to thousands of work threads execute simultaneously (Nickolls et al., 2008). Many threads group together as a thread block, in which threads communicate through shared memory and co-operate through barrier synchronization. Thread blocks are further grouped into kernel grids. The programmer specifies the number of threads per block and number of blocks forming the grid. In our code, we program this ensemble via CUDA, a parallel computing platform that allows general-purpose computing on GPUs (GPGPU) using a familiar C-like syntax.

Understanding the memory hierarchy of GPUs is important for achieving optimal performance for parallel programs. Each thread has its own set of processor registers and local memory for thread-private variables, which provide the fastest memory access. Each thread block has a limited shared memory pool that is only visible to the threads within this block. All threads also have access to a large high-bandwidth, but off-chip (global) memory embedded on the GPU card. Shared memory provide high-speed access, while accessing global memory is hundreds of times slower (Micikevicius, 2009).

In our implementation, GPUs handle only the most computationally intensive tasks. When such a task is scheduled, relevant data are first copied from host CPU memory to the global memory on the GPU device. Then the GPU kernel is launched, which loads data to on-chip memory for defined operations and writes results to global memory. Finally, results are copied from the device back to

the host.

This parallel programming model has several limitations that we should keep in mind. First, we should minimize data transfer between the host and device because the transfer is extremely slow. Second, accessing global memory on the GPU is also relatively slow, so we want to minimize the reads from global memory and the writes to global memory. When we do read or write from global memory, we want sequential threads to access sequential addresses in memory. In this manner, the GPU "coalesces" multiple memory requests into a smaller number of 128-byte transactions, and we want to over-subscribe each GPU core with multiple threads, so that the cores remain active through thread-context-switching and the latency in memory access can be hidden. Third, launching a kernel also has overhead on the order of microseconds, so it is preferred to combine a series of kernels into a larger "fused" one. Finally, contemporary GPUs issue single instructions to a "warp" of 32 threads simultaneously, such that all threads within a warp must execute the same instruction each clock cycle. When threads within a warp follow different data-dependent conditional branches, their execution becomes temporally serialized; this can cause a performance penalty. To avoid this issue, one attempts to minimize the number of diverging branches within a warp.

### 3.2.6 Tree-based parallel algorithms: reduction and scan

Here we review two useful building blocks for massively parallel algorithms: *reduction* and *scan*. Reductions convert an array of elements into a single result. For example, if the reduction operator is addition, then the reduction takes an array $[a_0, a_1, \ldots, a_{n-1}]$ and returns a single value $\sum_{i=0}^{n-1} a_i$. Reductions are useful for implementing log-likelihood calculations, since independent samples contribute additively to the model log-likelihood. Taking an array $[a_0, a_1, \ldots, a_{n-1}]$, the *scan* operation returns the array $[a_0, a_0 + a_1, \ldots, \sum_{i=0}^{n-1} a_i]$. If we start from the beginning of the input array as above, the resulting array is called a *prefix sum*. While the resulting array is called a *suffix sum* if we start from the end and proceed towards the beginning. Scans are useful in accumulating statistics about individuals in the risk set in survival analysis. Implementing a sequential version of a reduction or scan both require $\sim n$ additions on an array of length $n$.

The parallel versions of reduction and scan use a tree-based approach shown in Figure 3.1. Note that effective parallelization of these types of binary tree traversals requires low latency sharing of partial sums across threads with appropriate synchronization, both aspects in which the large number of threads concurrently executing on the GPU greatly outperform multi-core CPU threads.

To obtain a parallel, work-efficient, and communication-avoiding prefix scan, we invoke the CUB library (Merrill and Garland, 2016). The efficiency of their prefix-scan approaches a simple copy operation, as their prefix-scan requires the optimal ~$2n$ data movements: $n$ reads and $n$ writes to the global memory. The scan is constructed on two levels of organization: (1) a global device-wide scan and (2) a set of local block-wide scans within each thread block. The local block-wide scan utilize a *reduce-then-scan* strategy that can be visually resembled as an "hourglass" shape comprising an *up-sweep* and a *down-sweep* as shown in Figure 3.1c. In the up-sweep phase, we traverse the tree from leaves to root for computing the partial sums. Then the running prefixes are aggregated in the block-wide down-sweep traversing back up the tree from root using the partial sums computed in the up-sweep phase. The global scan implementation within CUB applies a single-pass chained-scan approach to achieve just ~$2n$ global data movements. The global scan further propose a decoupled look-back strategy by assigning each thread block a status flag indicating one of the three status: (1) aggregate (partial sum) of the block is available; (2) prefix of the block is available; and (3) no information about the block is available for other processor. Then each block will perform the computation conditional on its predecessor's status flag. We refer readers to Merrill and Garland (2016) for more details on this algorithm. In this paper, we extend these parallel algorithms for Cox models and Fine-Gray models based on the implementation of reduction and scan from the CUB library.

### 3.2.7 GPU massive parallelization for parameter estimation

CCD is an inherently serial algorithm in which each iteration is based on the result of the last iteration. As we mentioned earlier, even within an iteration $t$, CCD cycles through each covariate $j$ for $j = 1, 2, \ldots, P$ one by one and the computation work for the next covariate cannot begin until the

current update finishes. This serial algorithm however can still benefit greatly from parallelization by exploiting fine-grain problem decomposition within each iteration. Within each iterate's covariate update, careful benchmarks reveal that over 95% of the runtime lies in computing the log-likelihood gradient $g'(\beta_j)$ and Hessian $g''(\beta_j)$.

To understand the computational work, let $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_N)^\top$ be an N-dimensional column vector and $\mathbf{M}$ be an $N \times N$ loading matrix with entries

$$M_{ij} = \begin{cases} 1 & \text{for } j \in R_1(Y_i) \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

Recall that the risk set $R_1(Y_i)$ contains all observations that have an observed event time equalling or after $Y_i$. Thus if we arrange the observations by their observed time $Y_i$ in decreasing order, matrix $\mathbf{M}$ is clearly a lower triangular, binary matrix, and matrix-vector multiplication $\mathbf{M}[\exp(\mathbf{X}\boldsymbol{\beta})]$ becomes a prefix sum over the elements in $\exp(\mathbf{X}\boldsymbol{\beta})$, where we define exponentiation (exp) as element-wise operation. For example, given $Y_i > Y_{i'}$, the set $R(Y_i)$ consists of all the observations from $R(Y_{i'})$ and the set $\{t : Y_t \in [Y_{i'}, Y_i]\}$, then $\sum_{r \in R(Y_i)} \exp\left(\mathbf{x}_r^\top \boldsymbol{\beta}\right) = \sum_{r \in R(Y_{i'})} \exp\left(\mathbf{x}_r^\top \boldsymbol{\beta}\right) + \sum_{r \in \{t : Y_t \in [Y_{i'}, Y_i]\}} \exp\left(\mathbf{x}_r^\top \boldsymbol{\beta}\right)$. Making these substitutions in Equation 3.3, we arrive at

$$L_{\text{pseudo}}(\boldsymbol{\beta}) = \boldsymbol{\delta}^\top \mathbf{X}\boldsymbol{\beta} - \boldsymbol{\delta}^\top \log\{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})]\}, \tag{3.15}$$

where we define forming the logarithm (log) as element-wise operation, and $S_{\text{pre}}[\boldsymbol{\nu}]$ as the prefix sum of arbitrary vector $\boldsymbol{\nu}$. Then the unidimensional gradient and Hessians under the Cox proportional hazards model falls out as

$$g'(\beta_j) = \boldsymbol{\delta}^\top \mathbf{X}_j - \boldsymbol{\delta}^\top \mathbf{G} \text{ and} \tag{3.16}$$

$$g''(\beta_j) = -\boldsymbol{\delta}^\top (\mathbf{H} - \mathbf{G} \times \mathbf{G}), \tag{3.17}$$

where

$$\mathbf{G} = \frac{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j]}{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})]} \text{ and} \tag{3.18}$$

$$\mathbf{H} = \frac{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j]}{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})]} \tag{3.19}$$

and vector $\mathbf{X}_j$ is the $j$-th column of $\mathbf{X}$. Note that we define here multiplication ($\times$) and division ($/$) as element-wise operations as well.

While matrix-vector multiplication involving $\mathbf{M}$ takes $\mathcal{O}(N^2)$ operations, identifying the cumulative structure reduces the time complexity to linear by calculating prefix sum $S_{\text{pre}}[\boldsymbol{\nu}]$ in series, and parallelization further reduces the time complexity to $\mathcal{O}(\log_2 N)$ due to parallel scan's tree-based structure (Harris et al., 2007). Finally, the vector-vector multiplication involving $\delta^\top$ can be calculated as an element-wise multiplication in parallel in constant time and a reduction through a binary-tree in $\mathcal{O}(\log_2 N)$.

Under the Fine-Gray model, let $\mathbf{W} = (w_1, \ldots, w_N)^\top$ be an N-dimensional column vector of precomputed censoring weights described in previous section, and $\mathbf{N}$ as an $N \times N$ loading matrix with entries

$$N_{ij} = \begin{cases} 1 & \text{for } j \in R_2(Y_i) \\ 0 & \text{otherwise.} \end{cases} \tag{3.20}$$

Recall that $R_2(Y_i) = \{r : Y_r < Y_i \cap \varepsilon_r = 2\}$ and $R_1(Y_i)$ and $R_2(Y_i)$ are disjointed, so $\mathbf{N}$ is an upper triangular, binary matrix and $\mathbf{N}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{W}]$ is a suffix sum if we arrange the observations by their observed time $Y_i$ in decreasing order. Then making the following substitutions in Equation 3.16 and 3.17 yields the unidimensional gradient and Hessians under the Fine-Gray model

$$\mathbf{G} = \frac{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j] + S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{W}]}{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})] + S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{W}]} \text{ and} \tag{3.21}$$

$$\mathbf{H} = \frac{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j] + S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j \times \mathbf{W}]}{S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})] + S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{W}]}, \tag{3.22}$$

where we define $S_{\text{suf}}[\boldsymbol{\nu}]$ as suffix sum of vector $\boldsymbol{\nu}$.

It is worth noting that the risk set under the competing risk setting consists of two disjoint sets due to multiple types of event, thus a single pass scan furnishes neither the numerator nor denominator. Instead, the numerator and denominator of Equation 3.21 and 3.22 can be computed through a forward (prefix) scan $S_{\text{pre}}[\boldsymbol{\nu}]$ plus a backward (suffix) scan $S_{\text{suf}}[\boldsymbol{\nu}]$ together (Kawaguchi et al., 2021).

In summary, we can decompose the calculation of the gradient and Hessian for $\beta_j$ in the following four sequential steps:

(1) Read in non-zero $x_{ij}$ for $i = 1, 2, \ldots, N$ and update $[\mathbf{X}\boldsymbol{\beta}]_i$ as

$$[\mathbf{X}\boldsymbol{\beta}]_i^{(\text{new})} = [\mathbf{X}\boldsymbol{\beta}]_i + x_{ij}\Delta\beta_j.$$

Then perform three element-wise embarrassingly parallel transformations that read from the new estimate of $[\mathbf{X}\boldsymbol{\beta}]$, and output $[\exp(\mathbf{X}\boldsymbol{\beta})]$, $[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j]$ and $[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j]$. Here we should keep in mind that $\mathbf{X}$ is generally sparse such that many elements $x_{ij}$ are zeros, and exponentiation is an expensive operation in floating-point.

(2) Scans:

(a) Under the Cox model, we perform three forward scans that take the three output vectors of (1) as input, and return $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})]$, $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j]$ and $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j]$.

(b) Under the Fine-Gray model, we perform three forward scans and three backward scans that take the three output vectors of (1) as input, and return $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta})]$, $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j]$, $S_{\text{pre}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j]$, $S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{W}]$, $S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{W}]$ and $S_{\text{suf}}[\exp(\mathbf{X}\boldsymbol{\beta}) \times \mathbf{X}_j \times \mathbf{X}_j \times \mathbf{W}]$.

(3) An element-wise transformation that takes the output vectors of (2) as well as the indicator vector $\boldsymbol{\delta}$ as input, and outputs two new vectors $\boldsymbol{\delta} \times \mathbf{G}$ and $\boldsymbol{\delta} \times (\mathbf{H} - \mathbf{G} \times \mathbf{G})$.

(4) Two reductions that take the output vectors of (3) as input, and output two double summations

$\boldsymbol{\delta}^\top \mathbf{G}$ and $g''(\beta_j) = \boldsymbol{\delta}^\top (\mathbf{H} - \mathbf{G} \times \mathbf{G})$. Note that the first term $\boldsymbol{\delta}^\top \mathbf{X}_j$ in gradient $g'(\beta_j)$ can be precomputed and the value does not change during CCD.

Although parallel computing of the above numerical operations is much faster than serial evaluation, one crucial limitation is that a memory transaction involving reading or writing from global memory may take up to two orders of magnitude more time than a regular numerical operation applied to the value sitting in the limited number of on-chip registers within a GPU (or CPU for that matter) (Holbrook et al., 2021). In order to minimize memory transactions, we fuse several of our operations together for both our serial and parallel implementations. First, we fuse the multiple scans in step (2) together as a tuple-scan, which takes a tuple of three vectors as input, and outputs a tuple of three (or six) vectors. Note that in Fine-Gray model, we perform the forward tuple-scan and the backward tuple-scan on the same tuple of three vectors, but read the input in two opposite directions simultaneously. Similarly, we can fuse the multiple reductions in step (4) as a tuple-reduction. Since the element-wise transformations in step (3) take $\mathcal{O}(1)$ time with GPU parallelization, step (3) and step (4) can be regarded as a transformation-reduction. Finally, since both scan and reduction parallelization share the same binary-tree structure, we further fuse steps (2) - (4) together into a single kernel. Through fusion, for example, the output tuple from the scans never need to be written to global memory, nor read back for the later transformation-reductions. The fused kernel saves $2/3$ of the reads and writes than executing three separated kernels. It is also worth noting that the computational work of the gradient and Hessian evaluation share a similar structure and even some component such as $\mathbf{G}$, such that we have fused the computation of gradient and Hessian together to circumvent unnecessary kernel overhead and facilitate data reuse of these intermediate terms.

To exploit the sparsity of the design matrix $\mathbf{X}$ , we parallelize the transformation in step (1) using a sparse CUDA kernel, which only reads in and processes the non-zero entries while keeping other entries as zeros all the time during CCD updates. This sparse kernel saves data movement as well as reduces memory bandwidth requirements significantly when $\mathbf{X}$ is sparse, which is common in real-world scenarios.

Figure 3.2 illustrates our fused kernel for evaluating the log-likelihood gradient and Hessian on

the GPU. In this kernel, we spawn $S = B \times IPT \times G$ threads, where $B$ is the number of concurrent threads forming a thread block, $IPT$ is the number of work-items (elements of input) evaluated per thread, and $G = \lceil \frac{N}{B \times IPT} \rceil$ denotes the number of thread blocks. Both of the block size $B$ and the thread grain size $IPT$ are constrained by the hardware and are tunable constants. In practice, we follow the parameter settings in CUB with $B = 128$ and $IPT = 15$ for the fused kernel. For the sparse kernel we wrote, we choose $B = 256$ and $IPT = 1$. In the figure, each box in dashed line represent a thread block and recall that all threads within a block can access a shared memory that is a low-latency and on-chip (Nickolls et al., 2008) and is useful for performing the efficient scan in step (2) and reduction in step (4). The threads in parallel first read the values of tuple $\{\exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right), x_{ij} \exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right), x_{ij}^2 \exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right)\}$ for the current covariate column $j$ from global memory and then conduct the single-pass adaptive look-back tuple-scan (Merrill and Garland, 2016) utilizing a *reduce-then-scan* strategy. Next, the threads read in the values of $\delta$ and perform the transformation with the on-chip cumulative sums and $\delta$. The threads then perform a binary-tree tuple-reduction using shared memory, and one thread from each block writes its partial aggregates to global memory. Finally, a single-block reduction kernel sums over the $G$ partial aggregates and writes the gradient $g'(\beta_j)$ and Hessian $g''(\beta_j)$ back to global memory.

When CCD processes the $j$-th covariate, we only need to update $\beta_j$ and corresponding vector $\mathbf{X}\boldsymbol{\beta}$ if $\Delta\beta_j \neq 0$. Recall that the computation of $\Delta\beta_j$ requires $g'(\beta_j)$ and $g''(\beta_j)$ when regularization applies. Since both gradient $g'(\beta_j)$ and Hessian $g''(\beta_j)$ are computed on the GPU, we avoid $P$ data transfers between GPU and CPU in one CCD cycle by using a GPU kernel to check if $\Delta\beta_j \neq 0$ and then update $\beta_j$ and $\mathbf{X}\boldsymbol{\beta}$ if needed. Figure 3.3 details the workflow to implement CCD using GPU parallelization.

It is worth noting that thread-divergent branches in a CUDA kernel can substantially impact performance as execution gets temporally serialized. However, this is not an issue in our implementation because such branch divergence penalties only occur when threads within the same warp, but not across warps, need to execute alternative instructions. The branches in scan and reduction kernel in CUB are mainly due to slightly different instructions for the first processing tile and the last pro-

cessing tile. Thus the divergence occurs across warps and does not have an impact on performance penalty.

### 3.2.8 Multi-stream cross-validation

We use $k$-fold cross-validation to search for the optimum tuning parameters $\gamma$ or $\tau$ that controls the strength of regularization. We search for different values for the tuning parameter. For each value, the procedures are as follows:

(1) Randomly split data into $k$ partitions.

(2) For each of the $k$ partitions, we fit survival models via CCD on the remaining $k-1$ partitions and compute the predictive log-likelihood of this partition using the estimated $\hat{\boldsymbol{\beta}}$.

(3) Average out-of-sample likelihood across all $k$ folds.

(4) Repeat Step (1) - Step (3) $r$ times to reduce spurious effects from random data partitioning.

Finally, we select the tuning parameter with the smallest average out-of-sample likelihood as the desired optimal value.

We further improve the efficiency of cross-validation using multi-stream GPU and multi-threaded CPU approaches. Here, a stream denotes a sequence of operations (kernels) that execute in issue-order on a GPU (Rennich, 2011). Instead of the fitting $k$ partitioned models serially in a single (default) stream, we fit the $k$ models across $s$ streams in parallel, where $1 < s \leq k$ and each GPU stream is scheduled by an independent CPU thread. Likewise, for the multi-threaded CPU approach, $s$ CPU threads evaluate the $k$ models in parallel.

### 3.2.9 Comparison with an alternative massive parallelization of Cox models

In Ko et al. (2022), the authors presented sample PyTorch code for parallelizing proximal gradient descent on modest-sized $\ell_1$ regularized dense Cox regression. Here we provide a qualitative comparison of our method with this alternative approach, focusing on the per-cycle cost of cyclic coordinate

descent and proximal gradient descent.

A single iteration of proximal gradient descent on $\ell_1-$regularized Cox regression requires the following steps:

(1) A matrix-vector multiplication $\mathbf{X}\boldsymbol{\beta}$.

(2) An element-wise transformation $\exp(\cdot)$ on the output vector of (1).

(3) A scan on the output vector of (2).

(4) Two matrix-vector multiplications of $\mathbf{P}\boldsymbol{\delta}$ and $\mathbf{X}^{\mathbf{T}}[\boldsymbol{\delta}-\mathbf{P}\boldsymbol{\delta}]$, where the matrix $\mathbf{P}=(\pi_{ij})$ is defined as $\pi_{ij}=I(y_i\geq y_j)\exp(x_i^T\beta)/\sum_{i:y_i\geq y_j}\exp(x_i^T\beta)$.

Without considering the specific parallel library, the above steps contain $\mathcal{O}(NP)+\mathcal{O}(N)+\mathcal{O}(N)+\mathcal{O}(N^2)$ operations, respectively. Meanwhile, one sweep of coordinate descent implemented in Cyclops requires no more than $\mathcal{O}(NP)$ operations (the worst case is that the data is dense), as each of the four steps outlined in Section 3.2.7 only at most requires $\mathcal{O}(N)$ operations. Additionally, step (4) in the proximal gradient descent approach described earlier can also be reduced to $\mathcal{O}(N)$ by utilizing the same trick discussed in Section 3.2.7.

It is important to note that the number of iterations required to achieve an equivalent termination criterion is highly dependent on the data and is beyond the scope of our manuscript.

We would also like to emphasize that one of the main feature of our implementation is that it supports and benefits from a sparse data format when available, as discussed in Section 3.2.7.

## 3.3 Results

We examine the performance of GPU vs CPU computing for fitting our massive sample-size survival models. To accomplish this, we conduct a series of synthetic experiments to investigate the relative compute-time of our parallelization across different sample sizes. We then reproduce a real-world study using our GPU implementation under a Cox model and extend the study to the competing risks setting. If not specified, we use a system equipped with a 10 core 3.3 GHz Intel(R) Xeon(R) W-2155

CPU and an NVIDIA Quadro GV100 with 5120 CUDA cores and 32GB RAM that can achieve up to 7.4 Tflops double-precision point performance.

### 3.3.1 Kernel fusion

Kernel fusion is one of the main strategy we apply for achieving optimal performance. Taking the Cox model as an example, we compare three different implementations for steps (2) - (4) in Section 3.2.7:

(a) Separated kernels

  (i) Three separated scans using `cub::DeviceScan::InclusiveSum`,

  (ii) One transformation operation using `thrust::transform`, and

  (iii) Two separated reductions using `cub::DeviceReduce::Sum`.

(b) Partially-fused kernels

  (i) A tuple-scan using `cub::DeviceScan::InclusiveScan`, and

  (ii) A transformed tuple-reduction using `cub::TransformInputIterator` and `cub::DeviceReduce::Reduce`.

(c) Single fused kernel

We simulate the input vectors with $N = 100,000$ to $1,000,000$ from Uniform$(1,2)$. Figure 3.4 shows the speedup of the fused kernel and the partially-fused kernels over the separated kernels. The partially-fused kernels are $8-31$ times faster than the separated kernels. The single fused kernel further generates up to a 42-fold speedup compared with the separated kernels. Table 3.1 shows the runtimes (in milliseconds) in details when $N = 100,000$. Interestingly, we find performing a tuple-scan on three vectors is almost three times faster than performing three separated scans (0.0036 ms v.s. 0.0098 ms), demonstrating the values of fusion.

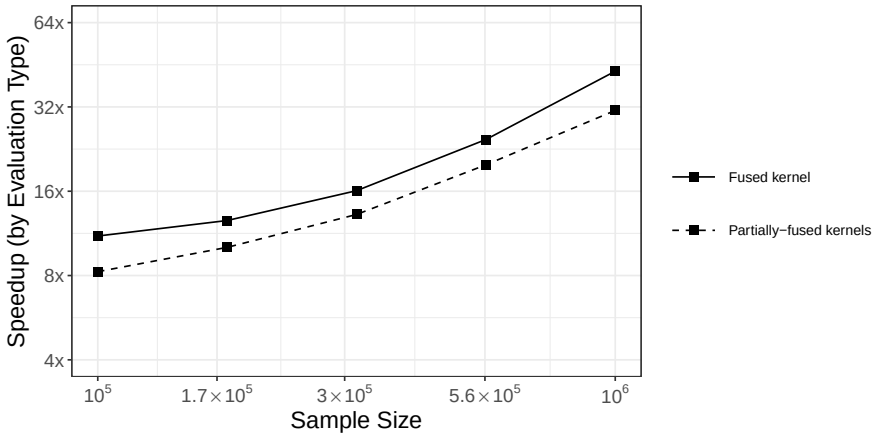



Figure 3.4: Speedup of the fused kernel and the partially-fused kernels over separated kernels for steps (2) - (4) in Section 3.2.7. The sample sizes range between $N = 10^5$ to $10^6$. Solid line shows the speedup of the fused kernel over separated kernels, and dashed line shows the speedup of the partially-fused kernel over separated kernels.

| Task | Separated kernels | Partially-fused kernels | Fused kernel |
|---|---|---|---|
| Three scans | 0.0098 | 0.0036 | 0.0052 |
| Transformation | 0.0415 | 0.0035 | |
| Two reductions | 0.0066 | | |
| **Total time** | 0.0579 | 0.0071 | 0.0052 |

Table 3.1: Runtimes (in milliseconds) of the separated kernels, the partially fused kernels, and the fused kernel for steps (2) - (4) in Section 3.2.7 when $N = 100,000$.

### 3.3.2 Synthetic experiments

We simulate indicator data $\mathbf{X}$ with $N = 100,000$ to $1,000,000$ samples and $P = 1000$ covariates with sparsity of 95%, where we randomly choose 5% of the entries uniformly to be 1s. We then draw

$$\beta_j \sim N(0,1) \times \text{Bernoulli}(0.80) \;\forall\; j \text{ and}$$
$$T_i \sim \text{Exponential}\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right) \;\forall\; i,$$

where the Bernoulli$(0.80)$ specifies that, on average, 80% of the $\beta_j$ are set to 0 to induce sparsity, and $T_i$ represent the time-to-event time for individual $i$. For generating competing risk times, we first draw:

$$\beta_{1j} \sim N(0,1) \times \text{Bernoulli}(0.80) \text{ and set } \beta_{2j} = -\beta_{1j} \; \forall \; j,$$

where $\boldsymbol{\beta}_1$ is the regression parameter of primary event and $\boldsymbol{\beta}_2$ is the regression parameter of competing event (Kawaguchi et al., 2021). Then we follow the design of Fine and Gray (Fine and Gray, 1999), where the cumulative incidence function (CIF) of primary event is a unit exponential mixture with mass $1-p$ at $\infty$ when $\mathbf{x}_i = \mathbf{0}$:

$$\Pr(T_{1i} \leq t_1 | \mathbf{x}_i) = 1 - [1 - p\{1 - \exp(-t_1)\}]^{\exp(\mathbf{x}_i^\top \boldsymbol{\beta}_1)},$$

and draw survival time of competing event $T_{2i}$ using an exponential distribution with rate $\exp(\mathbf{x}_i^\top \boldsymbol{\beta}_2)$. We set $p = 0.5$ in practice.

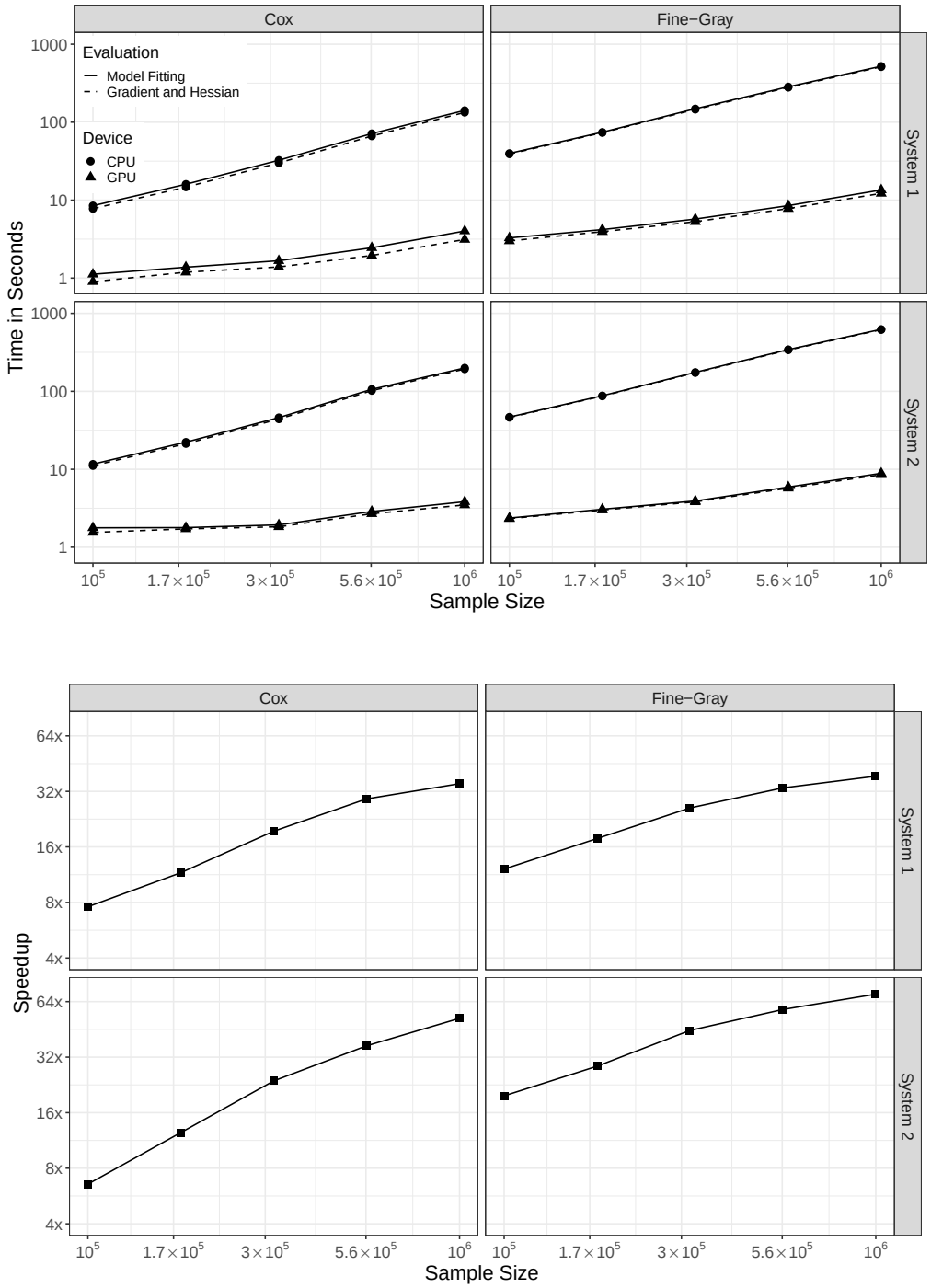



Figure 3.5: Runtimes (in seconds) and speedup of the GPU implementation relative to the CPU implementation for Cox and Fine-Gray models with a fixed $\ell_1$ penalty using a single GPU stream or CPU thread. The sample sizes range between $N = 10^5$ to $10^6$ with a sparsity of 95%. Solid lines show the total model fitting time, and dashed lines show the time for computing gradients and Hessians. The gap between the GPU's solid and dashed lines in the figures identify mostly computation that we did not port to the GPU, as data transfer between host and device during CCD updates accounts for less than 10% of this overhead and $\sim 1\%$ of the total model fitting time.

We fit these simulants under a fixed $\ell_1$ penalty with $\gamma = \sqrt{2}$ on a single CPU core and the default CUDA stream. To provide a comprehensive comparison for the performance gains, we conduct the experiments on two systems with different technical specifications. System 1 is equipped with a 3.3 GHz Intel(R) Xeon(R) W-2155 CPU (launch date 2017) and an NVIDIA Quadro GV100 (2018) with 5120 CUDA cores and 32GB RAM that can achieve up to 7.4 Tflops double-precision point performance. System 2 is equipped with a 2.2 GHz Intel(R) Xeon(R) Silver 4214 CPU (2019) and an NVIDIA A100 (2020) with 6912 CUDA cores and 80GB RAM that can achieve up to 9.7 Tflops double-precision point performance. Figure 3.5 presents runtimes comparisons across computing devices with a fixed number of covariates ($P = 1000$) with 95% sparsity and varying sample size $N$. We report both the total model fitting time and the time for computing gradients and Hessians, where the latter is our target of parallelization. On system 1 which is equipped with a powerful CPU, we see that the GPU parallelization delivers up to a 42-fold speedup for both Cox and Fine-Gray models in terms of computing gradient and Hessians. Despite additional data transfer and device initialization, GPU parallelization is still 35 $\times$ faster for this Cox model and 39 $\times$ faster for this Fine-Gray model relatively to our CPU implementation with one million samples. On system 2 which is equipped with a more powerful GPU, we see that the GPU parallelization achieves up to a 52-fold speedup for this Cox model and a 70-fold speedup for this Fine-Gray model. The data transfer between host and device during CCD updates only accounts for $\sim 1\%$ of the total model fitting time. We can see a rapid increase of runtimes on the CPU with increasing sample size, while the GPU approach continues to yield relatively shorter runtimes across varying sample sizes, as the devices is still not fully occupied.

### 3.3.3 Multi-stream cross-validated experiments

We further use these synthetic experiments to explore the performance of our approach using multi-threaded CPU and multi-stream GPU computing by simultaneously searching for an optimal strength of regularization. Here, we use 10-fold cross-validation with 10 repetitions per fold, resulting in 100 cross-validation replicates to estimate an optimal $\gamma$ under $\ell_1$ regularization. On the GPU, we

distribute the 100 replicates to $s$ CUDA streams driven by $s$ CPU threads. We also allow each of the CPU threads to process the 100 replicates directly on the CPU to demonstrate the performance of corresponding multi-threaded CPU parallelization. Figure 3.6 shows the runtimes of $\ell_1$ regularized Cox regression on varying number $s$ of threads or streams. Generally, our parallelization achieves more speedup through multi-stream GPU computing than through multi-core CPU threads alone. For example, the runtime of fitting a Cox model on $1,000,000$ samples reduces from nearly 9 hours across 8 CPU threads to 14 minutes across 8 CUDA streams. We also observe that the curves flatten out as number of CUDA streams and CPU threads increases, and this pattern is particularly obvious in the experiments on smaller sample sizes and on the CPU. This pattern suggests that there is both less computation to parallelize with relatively small sample sizes and that multi-core CPU parallelization remains limited by the smaller memory bandwidth available to the CPU.

### 3.3.4 Cardiovascular effectiveness of antihypertensive drug classes

The large-scale evidence generation and evaluation across a network of databases for hypertension (LEGEND-HTN) study (Suchard et al., 2019) provided real-world evidence on the comparative effectiveness and safety of five first-line antihypertensive drug classes using a retrospective, comparative new-user cohort design. Specifically, LEGEND-HTN studied the relative risk of 55 health outcomes of interest , including three major cardiovascular events (acute myocardial infarction, hospitalization for heart failure, and stroke), six secondary effectiveness outcomes, and 46 safety outcomes. Within each of nine observational health data sources, LEGEND-HTN employed propensity score matching or stratification for confounding adjustment and Cox proportional hazards modeling for hazard ratio (HR) estimation between new-users of each of the different drug classes. Interestingly, LEGEND-HTN identified that new-users of thiazide or thiazide-like diuretics (THZs) have a lower risk as compared to new-users of angiotensin-converting enzyme inhibitors (ACEIs) in terms of several effectiveness and safety outcomes, even though ACEIs are the most commonly initiated monotherapy across databases.

Here we examine patients initiating ACEIs and THZs, where the outcome is hospitalization for

| | Before matching | | | After matching | | |
|---|---|---|---|---|---|---|
| | THZ | ACEi | Standardised difference | THZ | ACEi | Standardised difference |
| **Age group (years)** | | | | | | |
| 10-14 | 0.1% | 0.1% | −0.02 | 0.1% | 0.1% | −0.01 |
| 15-19 | 0.4% | 0.6% | −0.02 | 0.5% | 0.5% | 0 |
| 20-24 | 1.5% | 1.0% | 0.04 | 1.4% | 1.2% | 0.02 |
| 25-29 | 3.3% | 2.1% | 0.07 | 3.0% | 2.8% | 0.01 |
| 30-34 | 5.5% | 3.8% | 0.08 | 5.1% | 4.9% | 0.01 |
| 35-39 | 7.7% | 6.0% | 0.07 | 7.3% | 7.2% | 0 |
| 40-44 | 10.1% | 8.5% | 0.05 | 9.8% | 9.9% | 0 |
| 45-49 | 12.2% | 11.4% | 0.03 | 12.1% | 12.1% | 0 |
| 50-54 | 13.8% | 14.0% | −0.01 | 13.9% | 13.8% | 0 |
| 55-59 | 13.0% | 14.5% | −0.05 | 13.3% | 13.5% | −0.01 |
| 60-64 | 10.6% | 12.5% | −0.06 | 11.0% | 10.9% | 0 |
| 65-69 | 8.1% | 9.6% | −0.05 | 8.4% | 8.5% | 0 |
| 70-74 | 5.5% | 6.6% | −0.05 | 5.7% | 5.9% | −0.01 |
| 75-79 | 4.5% | 4.9% | −0.02 | 4.6% | 4.7% | 0 |
| 80-84 | 3.1% | 3.5% | −0.02 | 3.2% | 3.4% | −0.01 |
| 85-89 | 0.7% | 0.8% | 0.02 | 0.7% | 0.6% | 0.01 |
| **Gender** | | | | | | |
| Female | 63.3% | 44.2% | 0.39 | 60.9% | 61.6% | −0.02 |
| **Medical history (general)** | | | | | | |
| Chronic obstructive lung disease | 3.1% | 3.6% | −0.03 | 3.1% | 3.3% | −0.02 |
| Diabetes | 7.3% | 24.2% | −0.48 | 7.8% | 8.2% | −0.02 |
| Hyperlipidemia | 30.8% | 42.4% | −0.24 | 32.3% | 32.0% | 0.01 |
| Obesity | 16.3% | 13.8% | 0.07 | 15.4% | 15.4% | 0 |
| Osteoarthritis | 11.5% | 11.2% | 0.01 | 11.5% | 11.9% | −0.01 |
| **Medical history (cardiovascular disease)** | | | | | | |
| Cerebrovascular disease | 1.5% | 2.2% | −0.06 | 1.5% | 1.6% | −0.01 |
| Coronary arteriosclerosis | 2.0% | 3.8% | −0.11 | 2.0% | 2.1% | 0 |
| Heart disease | 8.0% | 10.7% | −0.09 | 8.0% | 8.2% | −0.01 |
| **Medication use** | | | | | | |
| Anti-inflammatory and antirheumatic products | 47.8% | 45.6% | 0.04 | 46.9% | 47.6% | −0.01 |
| Antithrombotic agents | 17.0% | 24.4% | −0.18 | 17.4% | 17.7% | −0.01 |
| Beta blocker | 14.1% | 20.5% | −0.17 | 14.5% | 14.5% | 0 |

Table 3.2: Baseline hypertensive patient characteristics for new-user of THZ and ACEi in the Optum EHR database. We conduct a propensity score matching with the matching ratio of $1:1$. Less standardised difference of population proportions after matching indicate improved balance between two new-user cohorts in terms of confounders. THZ = thiazide or thiazide-like diuretics. ACEi = angiotensin-converting enzyme inhibitors.

heart failure from the Optum® de-identified Electronic Health Record dataset (Optum EHR). Optum EHR represents data from 85 million individuals that are commercially or Medicare insured in the United States. The data contain medical claims, pharmacy claims, laboratory tests, vital signs, body measurements, and information derived from clinical notes. A total of $1,014,618$ patients are included in our study, 75.8% of whom initiated an ACEI and 24.2% of whom initiated a THZ. We consider the main treatment covariate (ACEI or THZ exposure), in addition to $9,811$ baseline patient characteristic covariates. Table 3.2 presents a small selection of major patient baseline characteristics. From all baseline characteristics, we build a propensity score model and match ACEI and THZ new-users in a $1:1$ ratio. A total of $434,866$ patients were kept for further analysis after propensity score matching. We find ACEI new-users are more likely to be male, have diabetes, hyperlipidemia or heart disease comparing with patients initiating a THZ before propensity score matching. The THZ and ACEi cohorts are well-balanced on all $9,811$ baseline patient characteristics after matching, identified through low standardized difference of population proportions. The average sparsity of patient characteristic covariates is 97.11%, which means 2.89% of the entries are non-zero, occupying about 2.96GB RAM in sparse matrix format. We first fit a Cox proportional hazards model to estimate the HR between THZ and ACEi initiation for the risk of hospitalization for heart failure. We also fit a Fine-Gray model in which we consider acute myocardial infarction as a competing risk of hospitalization for heart failure, given myocardial infarction substantially elevates the future risk of heart failure. We include all patient characteristics as additional covariates in the Cox and Fine-Gray models with $\ell_1$ regularization on all variables except the treatment variable to achieve a limited form of adjustment for possible residual confounding, and again employ a 10-fold 10-replicate cross-validation to search for optimal tuning parameters. The analyses require almost two days to fit the regularized Cox model and more than three days to fit the regularized Fine-Gray model using eight CPU cores, while only taking 3.87 hours and 8.57 hours for the regularized Cox model and the regularized Fine-Gray model with our GPU implementation. Figure 3.7 reports the runtimes for regularized Cox and Fine-Gray models on varying numbers of CUDA streams versus the corresponding multi-core CPU parallelization.

Through massive parallelization, we find that initializing with a THZ shows better effectiveness than initializing with an ACEI in terms of hospitalization for heart failure risk under both the Cox model (HR 0.83, 95% bootstrapped percentile interval [BPI] $0.72-0.94$) and the Fine-Gray model (HR 0.83, 95% BPI $0.71-0.95$), where we provide the BPIs as crude measures of sampling variability given the challenges in constructing nominal confidence intervals for regularized models (Casella et al., 2010; Chatterjee and Lahiri, 2011). There are 215 and 134 out of 9,811 baseline covariates with non-zero effect sizes in the Cox and Fine-Gray models, respectively; these covariates include age, gender, hyperlipidemia, diabetes, osteoarthritis, and heart disease. While our estimates remain in line with the LEGEND-HTN study, they are reassuring in two ways. First, massive parallelization has enabled us to provide additional adjustment for possible residual confounding due to unobserved imbalance after propensity score matching without overfitting through cross-validation. Second, a consistent estimate after controlling for an obvious competing risk reduces concern over bias from informative censoring. Neither including thousands of baseline covariates nor handling a competing risk at this scale were possible in the original LEGEND-HTN study due to their erroneous time requirements; both are now feasible.

## 3.4 Discussion

This paper presents a time- and memory-efficient GPU implementation of regularized Cox and Fine-Gray regression models for analyzing large-scale time-to-event data with competing risks. We efficiently implement it in the open-source `R` package `Cyclops` (Suchard et al., 2013). In simulation studies, our GPU implementation is $35-70$ times faster than the equivalent CPU implementation with up to 1 million samples. In our real-world example with $\sim$ 400,000 hypertension patients and $\sim$ 9,000 covariates, massive parallelization reduces the total runtimes of both regularized Cox regression and regularized Fine-Gray regression with cross-validation from a few days on multi-core CPUs to few hours on a GPU.

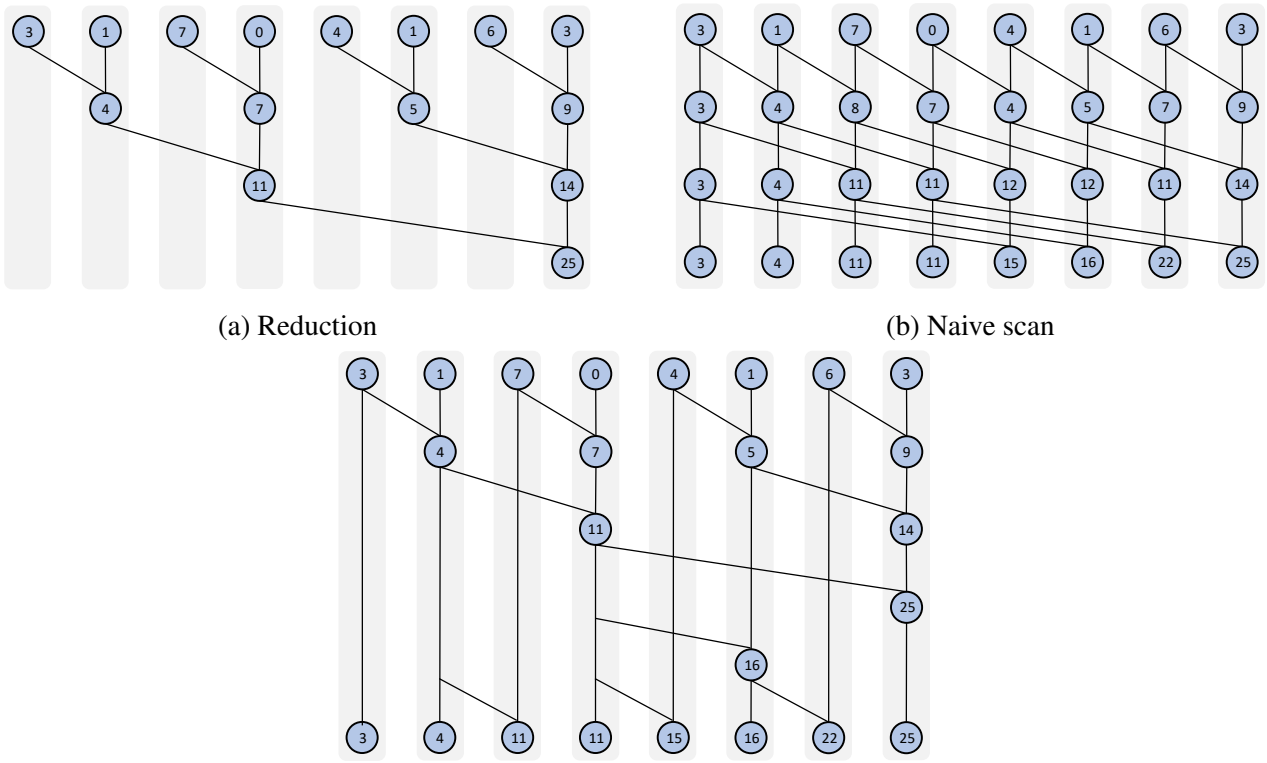
The observed speed-up is a combination of algorithmic advances as well as hardware optimiza-

tion. First, we observe that the cumulative structure of the risk set can be computed by a single-pass parallel scan algorithm, which can be very efficiently computed in parallel using a tree-based approach. This enables us to develop a work efficient and communication-avoiding tuple-scan algorithm in accumulating statistics about individuals in the risk set, dramatically speeding up likelihood, gradient, and Hessian evaluations. In particular, this tree-based scan algorithm takes great advantage of the high-speed shared memory of thread block on GPUs. Furthermore, as CCD is an inherently serial algorithm, we fuse multiple serial steps (1) transformations, (2) scans and (3) reductions together into a single vectorizable kernel. This fusion is possible because transformation is relatively lightweight operation, and scan and reduction algorithm share the same binary-tree structure. This kernel fusion reduces expensive memory transactions and kernel overheads which usually prevent the serial algorithm from benefiting by parallel computing. We also exploit the sparsity of the design matrix $\mathbf{X}$ through a sparse CUDA kernel to further save data movements. Finally, we only off-load the most computationally intensive routines to the GPU without rewriting the whole codebase.

There are numerous opportunities for improvement. For instance, our multi-stream implementation for $k$-fold cross validation has not achieved full concurrency for $> 1$ replicates, especially for relatively small sample size ($\sim$ 100,000) due to the low GPU utilization. The other potential problem is that we have to replicate the design matrix on each CUDA stream, which increases the memory bandwidth requirement. To overcome these limitations, one may use a single larger kernel to perform many repetitions of $k$-fold cross-validation on a single stream. By increasing the computational work in a single kernel, we can over-subscribe GPU threads to achieve a higher utilization and improve data reuse, but of course this creates the danger of exhausting register memory. In addition, we plan to add inference on parameter estimates via bootstrapping by synchronizing the computation of sub-samples. Both bootstrapping and cross-validation require independent computations on different sub-samples, which may benefit from parallel computing.

Finally, GPU hardware and libraries are rapidly maturing, and statisticians stand to benefit from GPU parallelization. We conclude this work with a few tips for educational purposes. First, simple building blocks such as scans and reductions are ubiquitous in statistical inference. One can

easily apply cutting-edge parallel implementations of these building blocks to gain instant speed-up. Second, kernel fusions should be employed whenever possible for serial algorithms to reduce expensive memory transactions and kernel overheads. Finally, when optimize an existing program, one should consider off-loading the most computationally intensive routines to the GPU in stages, before rewriting the whole codebase.




(a) Reduction

(b) Naive scan

(c) *Reduce-then-scan*

Figure 3.1: Tree-based parallel implementation of reduction and scan. Each grey box (column) represents a thread. Each thread runs a series of steps, and some steps must wait for results from other threads. Subfigures (a) and (b) show the naive binary tree-based approach for reduction and scan, respectively. Subfigure (c) presents a work-efficient scan algorithm using *reduce-then-scan* strategy, which includes an up-sweep phase for accumulating the partial sums and a down-sweep phase for aggregating the prefix sums. The tree-based approach for reduction and scan generally requires much data communication between threads but this remains low latency in shared memory, and thus is suitable for GPU parallelization.

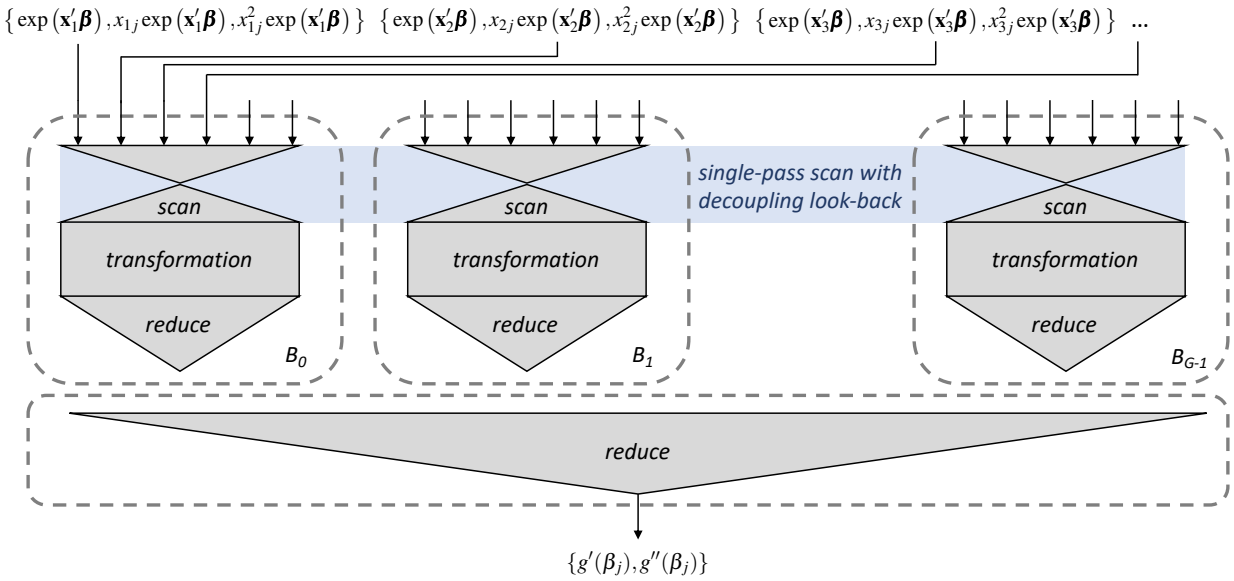



Figure 3.2: Fused kernel for evaluating the gradient and Hessian for $\beta_j$. We fused the single-pass scan with decoupling look-back (represented by the "hourglass"), the element-wise transformation (represented by the rectangle), and the reduction (represented by the upside down triangle) together in a fused kernel. Specifically, each of $G$ blocks (showed as a box in dashed line) reads a batch of triplets from global memory, performs a single-pass adaptive tuple-scan and a transformation-reduction to compute local partial sum of gradients and Hessians in shared memory, then efficiently adds the pair of partial sums in a binary-tree tuple-reduction within a single block and writes the resulting pair of gradient and Hessian to global memory.

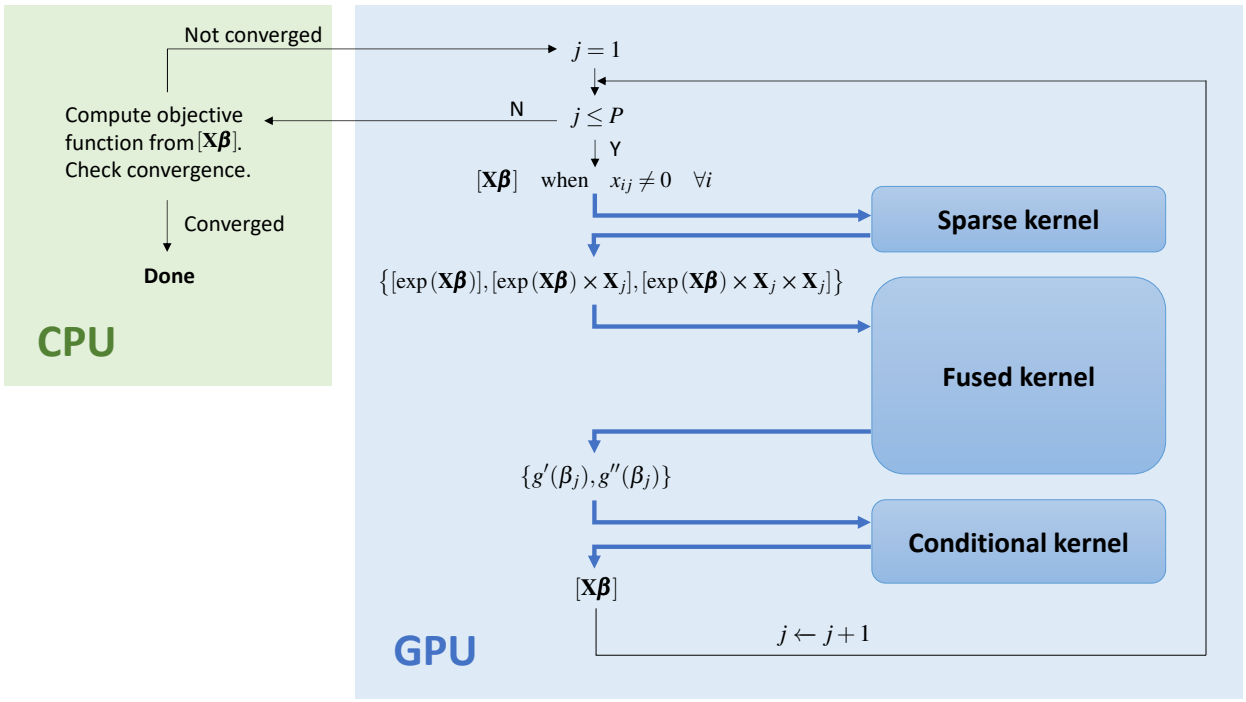



Figure 3.3: The workflow of implementing CCD using GPU parallelization: for each $j = 1, \ldots, P$, a sparse kernel reads in the entries of $[\mathbf{X}\boldsymbol{\beta}]$ for which $x_{ij} \neq 0$ and writes the updated tuple of vectors to the global memory of the GPU, then a fused kernel performs a tuple-scan and a transformation-reduction to compute the gradient and Hessian of the log-likelihood with respect to $\beta_j$. Finally a conditional kernel computes $\Delta\beta_j$ and updates $\beta_j$ and $\mathbf{X}\boldsymbol{\beta}$ if $\Delta\beta_j \neq 0$. Blue arrows represent data transactions to or from global memory. No data transfer between the GPU and CPU is needed until CCD finishes a complete cycle.

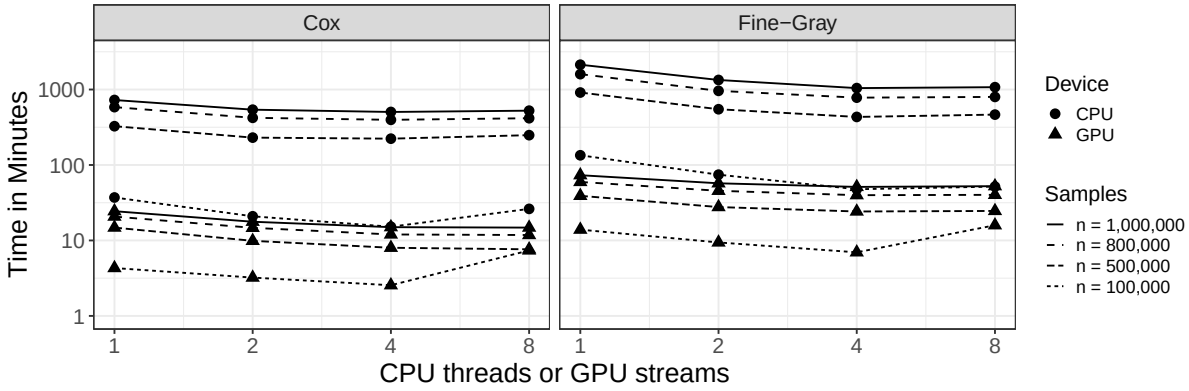



Figure 3.6: Runtimes for $\ell_1$ regularized Cox and Fine-Gray models with 10-fold 10-replicate cross-validation using multi-core CPU and multi-stream GPU computing. The sample sizes range between $N = 10^5$ to $10^6$ with a sparsity of 95%.

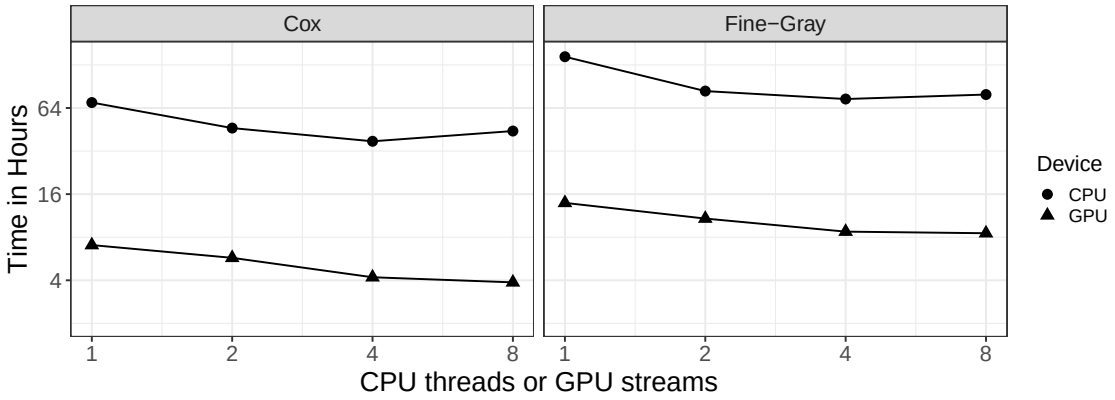



Figure 3.7: Runtimes (in hours) for $\ell_1$ regularized Cox and Fine-Gray models using multi-stream GPU and multi-core CPU computing. The data contain $434,866$ new-users of THZs and ACEIs, each with $9,811$ baseline patient characteristics covariates. We add a $\ell_1$ penalty on all baseline covariates and use a 10-fold 10-replicate cross-validation to search for optimum tuning parameters.

# CHAPTER 4

# Efficient GPU-Accelerated Fitting of Stratified and Time-Varying Extended Cox Models

## 4.1 Introduction

The Cox proportional hazards model (Cox, 1972) reigns as the most popular semi-parametric approach in survival analysis, providing valuable insights into the relationships between covariates and the hazard function. Numerous extensions of the Cox model, such as stratification, time-varying covariates, and time-varying coefficients, have been developed to accommodate the dynamic nature of real-world research problems. The stratified Cox model (Therneau et al., 2000) can handle covariates that do not satisfy the proportional hazards (PH) assumption by stratifying the data on them. Additionally, Crowley and Hu (1977) introduces a method to handle time-varying covariates in the Cox model, enabling the analysis of covariate changes over time. Similarly, Zucker and Karr (1990) extends the model to incorporate time-varying coefficients, facilitating the examination of how covariates' effects evolve over time.

While these contributions have significantly improved the versatility of the Cox model, they often encounter challenges in handling the ever-expanding size of modern observational datasets, particularly electronic health record (EHR) and administrative claims sources. EHR and claims datasets now encompass up to hundreds of millions of individuals, involving hundreds of thousands of patient characteristics, diseases, medications, and procedures occurring over decades of patient lives (Hripcsak et al., 2016, 2021). The Cox model itself exhibits quadratic growth with sample size in its naïve implementation, further exacerbating the computational burden. Moreover, the

various extensions of the Cox model introduce additional complexities, making the model fitting process even more challenging. Therefore, statistical computing challenges arise both from the large-scale data and the intricacies of the extended Cox models. Addressing these challenges calls for the utilization of advanced computing techniques to scale up survival analysis using these semi-parametric models.

Research studies on these extensions of the Cox model mainly focus on proposing the novel estimation approaches and remain limited to small to moderate-sized data (Tian et al., 2005; Thackham and Ma, 2020; He et al., 2022). The utilization of graphics processing units (GPUs) and fine-grained parallelism to accelerate statistical computations is a relatively new and emerging area within the field of medical statistics. For instance, Suchard et al. (2013) showcase that massive parallelization through GPUs yields one to two orders of magnitude improvement over traditional central processing unit (CPU) parallelization when applied to a computationally demanding self-controlled case series models. Ko et al. (2022) investigate GPU parallelization for proximal gradient descent on modest sized $\ell_1$ regularized dense Cox regression using off-the-self software, such as PyTorch. These studies highlight the significant performance gains that leveraging GPUs achieves in complex statistical computations. Recently, we have proposed a time- and memory-efficient GPU implementation of regularized Cox and Fine-Gray regression models for analyzing large-scale, time-to-event data with and without competing risks (Yang et al., 2023).

In this manuscript, we leverage massive parallelization to enhance the scalability of the seemingly less parallelizable stratified Cox model, the Cox model with time-varying covariates, and the Cox model with time-varying coefficients. Specifically, we demonstrate that the Cox model with time-varying coefficients can be transformed into the Cox model with time-varying covariates when utilizing discrete time-to-event data. To accomplish this, we reveal that the Cox model with time-varying covariates shares a similar partial likelihood structure as the stratified Cox model. Consequently, all three extensions of the Cox model we investigate encounter the same computational bottleneck due to segmented scan, particularly in cases with high stratification or frequent changes in time-varying effects. Recognizing that segmented operations are not immediately obviously par-

allelizable, we address this issue by transforming the computational bottleneck into un-segmented operations (Sengupta et al., 2008). While even un-segmented scans, with their apparent serial output dependence, may not intuitively appear readily parallelizable, we leverage a single-pass parallel scan algorithm implemented in the cutting-edge GPU accelerated library `CUB` (Merrill, 2015). We implement our work in the easy-to-use `R` package `Cyclops`. Our GPU implementation significantly accelerates the computation of fitting these complex models on large-scale and high-dimensional simulated and real-world data by an order of magnitude compared to a similarly optimized CPU implementation, reducing the fitting time for the analyses containing one million patients from nearly one day to just one to two hours.

## 4.2 Methods

### 4.2.1 The stratified Cox proportional hazards model

The stratified Cox model provides a straightforward approach to handle a covariate that does not satisfy the proportional hazards (PH) assumption. For example, we can stratify the observations into different strata based on their disease stage when the disease stage does not meet the PH assumption, so that only the observations within each stratum share the same baseline hazard function. Let $T_{ki}$ denote the time-to-event time and $C_{ki}$ be the right-censoring time for individual $i$ in stratum $k$, $i = 1, ..., n_k$, and $k = 1, ..., K$. Here $n_k$ is the sample size of stratum $k$, and $K$ is the number of strata in the stratified Cox model. Then the total sample size is $N = \sum_{k=1}^{K} n_k$. For an individual, the observed time is given by $Y_{ki} = \min(T_{ki}, C_{ki})$, and $\delta_{ki}$ indicates whether the individual fails or is censored at $Y_{ki}$ by the value 1 versus 0. Let $\mathbf{x}_{ki}$ be a $P$-dimensional covariate vector for this individual. For this stratified Cox model with $K$ strata, the hazard for an individual from stratum $k$ is

$$h_k(t|\mathbf{x}) = h_{0k}(t) \exp\left(\mathbf{x}^\top \boldsymbol{\beta}\right), \tag{4.1}$$

where $h_{0k}(t)$ is an unspecified baseline hazard function for stratum $k$, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_P)^\top$ is a set of unknown, underlying model parameters that we wish to estimate. Note that unlike the classic Cox model that assumes the same baseline hazard function $h_0(t)$ for all individuals, a stratified Cox model allows a distinct baseline hazard function for each stratum but a common or shared set of model parameters $\boldsymbol{\beta}$.

The partial likelihood of the stratified Cox model falls out as the product of the partial likelihood contributions from all strata:

$$l_{\text{partial}}(\boldsymbol{\beta}) = \prod_{k=1}^{K} \prod_{i=1}^{n_k} \left[ \frac{\exp\left(\mathbf{x}_{ki}^\top \boldsymbol{\beta}\right)}{\sum_{r \in R_k(Y_{ki})} \exp\left(\mathbf{x}_{kr}^\top \boldsymbol{\beta}\right)} \right]^{\delta_{ki}}, \tag{4.2}$$

where $R_k(Y_{ki}) = \{r : Y_{kr} \geq Y_{ki}\}$ consists of the set of subjects who remain "at risk" for an event at time $Y_{ki}$ in stratum $k$.

When dealing with high-dimensional data, researchers may add an $\ell_1$-penalty into all or a large subset of the model parameters $\pi(\boldsymbol{\beta}) = \sum_j \pi(\beta_j|\gamma_j) = -\sum_j \gamma_j|\beta_j|$ to the log-partial likelihood and achieve regularization through estimating the joint penalized likelihood (Genkin et al., 2007; Mittal et al., 2014). In practice, one generally assumes $\gamma_j = \gamma \; \forall j$ and chooses $\gamma$ through cross-validation (Mittal et al., 2014).

### 4.2.2 The Cox model with time-varying covariates

In traditional Cox regression analysis, we usually only measure the covariates at baseline once. However, certain covariates may change during the follow-up period, such as repeated measurements in medical research. The Cox model is able to encompass time-varying covariates using a hazard function

$$h(t|\mathbf{x}(t)) = h_0(t) \exp\left(\mathbf{x}(t)^\top \boldsymbol{\beta}\right). \tag{4.3}$$

The partial likelihood is similar in form to the classic Cox model:

$$L_{\text{partial}}(\boldsymbol{\beta}) = \prod_{i=1}^{N} \left[ \frac{\exp\left(\mathbf{x}_i(Y_i)^\top \boldsymbol{\beta}\right)}{\sum_{r \in R(Y_i)} \exp\left(\mathbf{x}_r(Y_i)^\top \boldsymbol{\beta}\right)} \right]^{\delta_i}. \tag{4.4}$$

In practice, measurements of time are often discrete (Tutz et al., 2016), thus we consider the common case where $\mathbf{x}_i(t)$ can be seen as a piecewise-constant function on $K$ time intervals for individual $i = 1, 2, \ldots, N$ (Ngwa et al., 2016) such that

$$\mathbf{x}_i(t) = \begin{cases} \mathbf{x}_{1i} & \text{for } t \in [t_0, t_1) \\ \mathbf{x}_{2i} & \text{for } t \in [t_1, t_2) \\ \cdots & \\ \mathbf{x}_{Ki} & \text{for } t \in [t_{K-1}, t_K) \end{cases} \tag{4.5}$$

for some constants $0 = t_0 < t_1 < t_2 < \cdots < t_K = \max(Y_1, Y_2, \ldots, Y_N)$. Correspondingly, we can transform the time-fixed data, including the survival time and event indicator, into the discrete time intervals mentioned above. This type of data is also referred to as discrete time-to-event data (Tutz et al., 2016). Let set $S_k = \{i : Y_i \in [t_{k-1}, t_k)\}$ for $k = 1, 2, \ldots, K$. Then the partial likelihood becomes

$$L_{\text{partial}}(\boldsymbol{\beta}) = \prod_{k=1}^{K} \prod_{i \in S_k} \left[ \frac{\exp\left(\mathbf{x}_{ki}^\top \boldsymbol{\beta}\right)}{\sum_{r \in R(Y_i)} \exp\left(\mathbf{x}_{kr}^\top \boldsymbol{\beta}\right)} \right]^{\delta_i} \tag{4.6}$$

$$= \prod_{k=1}^{K} \prod_{i=1}^{N} \left[ \frac{\exp\left(\mathbf{x}_{ki}^\top \boldsymbol{\beta}\right)}{\sum_{r \in R(Y_{ki}^{(\text{aug})})} \exp\left(\mathbf{x}_{kr}^\top \boldsymbol{\beta}\right)} \right]^{\delta_{ki}^{(\text{aug})}}, \tag{4.7}$$

where augmented variables $Y_{ki}^{(\text{aug})} = \min(Y_i, t_k)$ and $\delta_{ki}^{(\text{aug})}$ indicates whether the individual $i$ fails or is censored at time $Y_{ki}^{(\text{aug})}$.

Although observations in different time intervals still share the same baseline hazard function, the partial likelihood follows the same structure as the stratified Cox model. Now we are able to fit a Cox model with time-varying covariates as a Cox model stratified on $K$ time intervals with

the augmented design matrix $\mathbf{X}^{(\text{aug})} = \begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \cdots & \mathbf{x}_{1N} & \mathbf{x}_{21} & \cdots & \mathbf{x}_{KN} \end{bmatrix} \in \mathbb{R}^{(K\times N)\times P}$, augmented observed time $\mathbf{Y}^{(\text{aug})} \in \mathbb{R}^{(K\times N)}$, and augmented event indicator $\boldsymbol{\delta}^{(\text{aug})} \in \mathbb{R}^{(K\times N)}$.

### 4.2.3 The Cox model with time-varying coefficients

Time-varying coefficient arises in survival analysis when a covariate's effect on the outcome is not constant over the follow-up time. For instance, we usually assume that the COVID vaccine efficacy varies before and after 14 days of vaccination. The proportional hazards assumption of the Cox model does not hold in this situation, as the hazard ratio comparing two specifications of a time-varying coefficient is no longer independent of time.

The extension of a Cox model with time-varying coefficients has a hazard function

$$h(t|\mathbf{x}) = h_0(t)\exp\left(\mathbf{x}^\top \boldsymbol{\beta}(t)\right), \tag{4.8}$$

where $\boldsymbol{\beta}(t) = (\beta_1(t), \beta_2(t), \ldots, \beta_P(t))^\top$. The partial likelihood is as follows:

$$L_{\text{partial}}(\boldsymbol{\beta}) = \prod_{i=1}^{N} \left[ \frac{\exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}(Y_i)\right)}{\sum_{r\in R(Y_i)} \exp\left(\mathbf{x}_r^\top \boldsymbol{\beta}(Y_i)\right)} \right]^{\delta_i}. \tag{4.9}$$

Often one can specify $\beta_j(t)$ as a simple step function (Zhang et al., 2018). Without loss of generality, suppose $x_1$ has a time-varying effect on the outcome before and after time $t_s$, then $\boldsymbol{\beta}(t)$ contains entries

$$\beta_j(t) = \begin{cases} \beta_{11} & \text{for } j = 1, t < t_s \\ \beta_{12} & \text{for } j = 1, t \geq t_s \\ \beta_j & \text{for } j = 2, 3, \ldots, P. \end{cases} \tag{4.10}$$

Then some simple calculation shows that

$$\mathbf{x}_i^\top \boldsymbol{\beta}(t) = \begin{cases} x_{i1}\beta_{11} + \sum_{j=2}^{P} x_{ij}\beta_j & \text{for } t < t_s \\ x_{i1}\beta_{12} + \sum_{j=2}^{P} x_{ij}\beta_j & \text{for } t \geq t_s \end{cases} = \mathbf{x}_i^\top(t)\boldsymbol{\beta}^{(\text{aug})}, \tag{4.11}$$

where

$$\mathbf{x}_i(t) = \begin{cases} (x_{i1}, 0, x_{i2}, \ldots, x_{iP})^\top & \text{for } t < t_s \\ (0, x_{i1}, x_{i2}, \ldots, x_{iP})^\top & \text{for } t \geq t_s \end{cases} \quad \text{and} \quad \boldsymbol{\beta}^{(\text{aug})} = (\beta_{11}, \beta_{12}, \beta_2, \ldots, \beta_P)^\top. \tag{4.12}$$

In this way, we can model a time-varying coefficient as a set of time-varying covariates and further turn a Cox model with time-varying covariates to a Cox model that stratified on time intervals.

### 4.2.4 Maximum partial likelihood estimation using cyclic coordinate descent

To maximize $L_{\text{partial}}(\boldsymbol{\beta})$ with or without a regularization penalty with respect to $\boldsymbol{\beta}$, we consider a cyclic coordinate descent (CCD) algorithm which cycles through each covariate $\beta_j$ and updates it by a Newton approach while holding all other covariates as contants (Genkin et al., 2007; Mittal et al., 2014). Specifically, when we cycle through the $j$-th covariate at $l$-th iteration, we can rewrite the partial log likelihood using a second-order Taylor approximation:

$$g(\beta_j) \approx g(\beta_j^{(l-1)}) + g'(\beta_j^{(l-1)})(\beta_j - \beta_j^{(l-1)}) + \frac{1}{2} g''(\beta_j^{(l-1)})(\beta_j - \beta_j^{(l-1)})^2, \tag{4.13}$$

where $g'(\beta_j^{(l-1)})$ and $g''(\beta_j^{(l-1)})$ represent the one-dimensional objective gradient and Hessian with respect to $\beta_j$ evaluated at previous iteration, respectively. To minimize the objective, the new estimate at $l$-th iteration falls out as:

$$\beta_j^{(l)} = \beta_j^{(l-1)} + \Delta\beta_j^{(l)} = \beta_j^{(l-1)} - \frac{g'(\beta_j^{(l-1)})}{g''(\beta_j^{(l-1)})}. \tag{4.14}$$

This CCD approach avoids the inversion of large Hassian matrices in the high-dimensional setting and only requires the scalar gradients and Hessians. This opens up opportunities for fine-grain parallelization (discussed in the next section). When the objective function is simply the negative log partial likelihood of stratified Cox model, the gradient and Hessian fall out as

$$g'(\beta_j) = -\sum_{k=1}^{K}\sum_{i=1}^{n_k} x_{kij}\delta_{ki} + \sum_{k=1}^{K}\sum_{i=1}^{n_k} \delta_{ki}\frac{\sum_{r\in R_k(Y_{ki})} x_{krj}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)}{\sum_{r\in R_k(Y_{ki})}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)} \tag{4.15}$$

and

$$g''(\beta_j) = \sum_{k=1}^{K}\sum_{i=1}^{n_k} \delta_{ki}\frac{\sum_{r\in R_k(Y_{ki})} x^2_{krj}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)}{\sum_{r\in R_k(Y_{ki})}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)} - \sum_{k=1}^{K}\sum_{i=1}^{n_k} \delta_{ki}\left(\frac{\sum_{r\in R_k(Y_{ki})} x_{krj}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)}{\sum_{r\in R_k(Y_{ki})}\exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right)}\right)^2. \tag{4.16}$$

Note that the repeated evaluations in the numerator and denominator of Equations (4.15) and (4.16) constitute the computational bottleneck. We can conveniently add the penalty $\pi(\boldsymbol{\beta})$ for $\boldsymbol{\beta}$ into the objective function when the regularization is needed.

With the step size $\Delta\beta_j^{(l)} = -\frac{g'(\beta_j^{(l-1)})}{g''(\beta_j^{(l-1)})}$ derived from Newton's method, we further improve the convergence by restricting the step size through a trust region approach (Genkin et al., 2007). Specifically, we initialize a trust region half-width $\Delta_j^{(0)} = 1$ and update it as $\Delta_j^{(l)} = \max\{2|\Delta\beta_j^{(l-1)}|, \Delta_j^{(l-1)}/2\}$. Subsequently, we constrain the step size when updating the parameter at iteration $l$ as follows:

$$\beta_j^{(l+1)} = \beta_j^{(l)} + \text{sgn}\left(\Delta\beta_j^{(l)}\right)\min\{|\Delta\beta_j^{(l)}|, \Delta_j^{(l)}\}. \tag{4.17}$$

When not considering the penalty $\pi(\boldsymbol{\beta})$, the objective function is clearly convex and differentiable everywhere in $\boldsymbol{\beta}$. However, the $\ell_1$ penalty is convex but nondifferentiable at origin. Therefore, we compute the directional derivatives (Wu et al., 2008) in both directions by settting $\text{sgn}\left(\Delta\beta_j^{(l)}\right) = +1$ and $\text{sgn}\left(\Delta\beta_j^{(l)}\right) = -1$ at origin. If both directional derivatives are non-negative, we skip the update for this iteration. If either directional derivative is negative, we update $\beta_j$ in that direction to minimize the objective. Since the objective is convex, it is impossible for both directional derivatives to be negative. While we do not provide a rigorous proof of convergence, the trust region method

was demonstrated effectiveness in our simulations and real-world experiments.

### 4.2.5 GPU-accelerated statistical computing strategies

There are two distinct strategies in parallel computing: coarse-grained parallelism and fine-grained parallelism. The former divides the problem into a small number of large tasks due to limitations in data communication between cores. This strategy is typically used in parallel computing on clusters and multi-core CPUs (Suchard et al., 2010). In contrast, fine-grained parallelism breaks down computational workloads into a large number of tiny tasks that run in almost lockstep. This strategy requires significant data communication between cores and is well-suited for GPUs since they have shared memory (Holbrook et al., 2021).

While conventionally used for graphic rendering, GPUs are growing in popularity in recent years for their potential to accelerate various scientific and engineering applications. In this section, we briefly review the basics of parallel computing on GPUs and discuss strategies for accelerating statistical computing using fine-grained parallelism.

Understanding the hierarchical structure of threads and memory of GPUs is crucial for achieving high performance in GPU programming. Each thread can access its own set of processor registers and local memory for thread-private variables. Collections of up to 512 threads on current hardware group together as a thread block that has a limited shared memory only accessible to the threads within this block, enabling efficient data communication within the same block. A grid is a collection of blocks that execute the same kernel function. GPUs also sport large, but off-chip global memory accessible by all executing threads, regardless of if they live in the same or different blocks. Accessing consecutive addresses in this global memory by threads in the same block leads to coalesced transactions, delivering much higher memory high-bandwidth than for most CPUs. It is important to note that the register memory, local memory of a thread, and shared memory of a block are high-speed on-chip memory, while accessing global memory is relatively slower. Therefore, minimizing the number of global memory transactions can greatly improve the performance of GPU

programs.

GPU parallel computing works by executing kernels that are functions that run in parallel across a set of parallel threads, following a single instruction, multiple thread (SIMT) architecture. To maximize the utilization of hardware resources, contemporary GPUs employ warp and lockstep execution (NVIDIA, 2023). A warp is a group of 32 parallel threads that execute the same instruction simultaneously. If threads within a warp diverge due to data-dependent branches, the warp executes each branch path serially, and the threads converge back to the same path after all branch paths complete. Thus, minimizing the number of diverging branches within a warp is crucial for achieving high performance. Although a branch penalty exists when the branch divergence occurs within a warp, modern GPUs are significantly more efficient at branching code than prior parallel processors with the single-instruction, multiple-data (SIMD) architecture.

In this paper, we adopt a widely used heterogeneous computing model (NVIDIA, 2023) between the CPU and GPU for accelerating computation. Specifically, we begin by offloading the most computationally demanding parts of the program to the GPU, while allowing the rest of the program to run on the CPU. Note that this requires moving data between the host (CPU) and device (GPU) memory, which can be slow due to limited bandwidth between devices, thus we aim to minimize these data movements. Once we partition the program across the host and device appropriately, the powerful computing capabilities of the GPU can make up for the expensive data movement by performing intensive calculations much faster than the CPU.

The computationally demanding portions in our case can benefit significantly more from fine-grained parallelism on GPUs than from traditional coarse-grained parallelism on CPUs. In traditional coarse-grained parallelism on CPUs, we divide the computational work into a limited number of batches (depending on the number of available CPU cores), with each core handling the computations for a batch of samples. In contrast, we can achieve much higher degree of parallelization on GPUs by breaking down the computation into numerous small, parallelizable tasks that require extensive inter-task communication.

### 4.2.6 Efficient parallel segmented-scan on GPUs

Here we introduce the crucial building blocks for massive sample size Cox regression analysis with and without stratification on GPUs: the *scan* and *segmented scan*.

A *scan* procedure takes an input array $a = [a_0, a_1, \ldots, a_{n-1}]$ and an associated binary operator $\oplus$, and produces an output array $b$ where $b_i = a_0 \oplus \cdots \oplus a_i$. Implementing a scan serially is trivial and requires $\mathcal{O}(n)$ operations. While the output array displays obvious serial dependence in its values, this procedure remains parallelizable when communication costs between threads is low. The naïve parallel scan algorithm is based on a balanced, binary tree of operations. This algorithm reduces the computational complexity to the height of the tree $\mathcal{O}(\log_2 n)$, assuming parallel operations execute in $\mathcal{O}(1)$ at the same level on the tree. This naïve algorithm, however, performs $\sum_{d=1}^{\log_2 n}(n-2^{d-1}) = \mathcal{O}(n\log_2 n)$ binary operations in total, even more than the sequential scan, though in parallel. Therefore, the naïve algorithm is not work-efficient, as it may require additional computational resources.

A work-efficient scan algorithm arises from a *reduce-then-scan* strategy that who operations visually resemble an "hourglass" shape consisting of an *up-sweep phase* and a *down-sweep phase*. In the *up-sweep phase*, we traverse the tree from leaves to root for computing a set of partial sums. In the *down-sweep phase*, we traverse back up the tree from the root to aggregate the scan output using the partial sums computed in the up-sweep phase. Overall the two phases perform $3(n-1) = \mathcal{O}(n)$ operations and is work-efficient for large arrays.

A *segmented scan* procedure takes an additional *segment descriptor* with the same dimension of the input array that encodes how the input array is divided into segments. For example, a segmented scan of the $+$ operator over an array of integers $a = [3,1,7,0,4,1,6,3]$ with the segment head flags $f = [1,0,1,0,0,1,0,0]$, which divide the input array into three subarrays, produces an output array $b = [3,4,7,7,11,1,7,10]$. Suppose the input array is divided into $K$ segments according to the given segment descriptor, a naïve parallel implementation can easily perform $K$ separate *scan* procedures. This implementation, however, can be very inefficient when $K$ is relatively large due to the overhead

of launching and monitoring for the completion of $K$ separate kernels, each of which may contain only a small amount of work. More importantly, the shorter data access patterns depending on how the input array is divided can interfere with coalescing global memory transactions.

Alternatively, we can avoid the additional overhead and suboptimal data access patterns by transforming a segmented scan into a single regular scan (Schwartz, 1980; Blelloch, 1990; Sengupta et al., 2008). The idea is that given the input array $a$, segment head flags $f$, and associated binary operator $\oplus$, one combines $f$ and $a$ together as a new input array of flag-value pairs $(f_i, a_i)$ and constructs a new binary operator $\oplus^s$ based on $\oplus$ and $f$, such that

$$(f_i, a_i) \oplus^s (f_j, a_j) \coloneqq (f_i \mid f_j, \text{ if } f_j = 1 \text{ then } a_j \text{ else } a_i \oplus a_j). \tag{4.18}$$

Hence, the algorithm's efficiency is independent of how the input array is segmented or the total number of segments. Figure 4.1 illustrates this efficient *segmented scan* algorithm, employing the *reduce-then-scan* strategy.

### 4.2.7 GPU massive parallelization for parameter estimation

In this section, we aim to show how to accelerate the computational work in 4.2.4 using the parallel building block presented in 4.2.6.

We first analyze the time complexity of our CCD algorithm presented in 4.2.4. In one cycle of CCD iteration, we update $\beta_j$ for $j = 1, \ldots, P$ using Newton's updates (4.14), each of which in turn requires evaluating the univariate gradient (4.15) and Hessian (4.16). The first term in gradient (4.15) requires $\mathcal{O}(K) + \mathcal{O}(\sum_{k=1}^{K} n_k = N)$ operations. Naïve computation of the second term in gradient (4.15) and both terms of Hessian (4.16) require $\mathcal{O}(K) + \mathcal{O}(\sum_{k=1}^{K} n_k^2)$ due to the additional inner sums in both numerator and denominator.

We can tackle fortunately all inner sums involving $R_k(Y_{ki})$ as *scan*, if we arrange the observations within a stratum $k$ by their observed time $Y_{ki}$ in decreasing order. Recall that the risk set $R_k(Y_{ki}) = \{r : Y_{kr} \geq Y_{ki}\}$ contains the individuals in stratum $k$ who have an observed time equaling or after $Y_{ki}$,

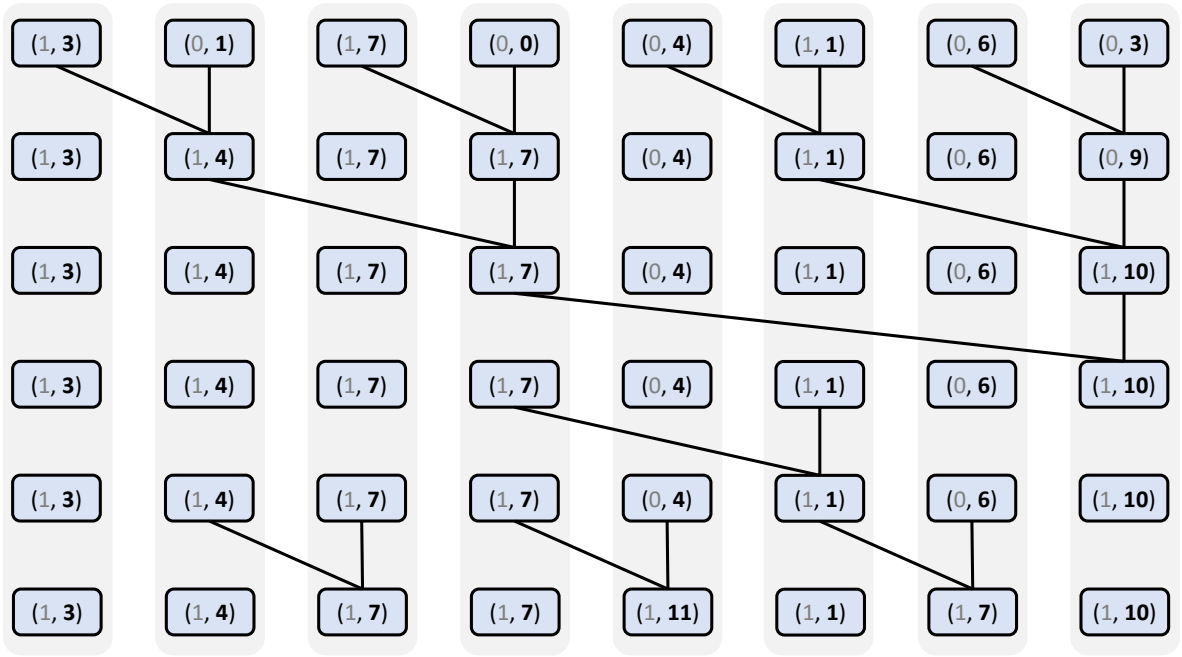



Figure 4.1: An efficient parallel *segmented scan* algorithm based on binary tree. The efficiency of this algorithm is independent of how the input array is partitioned into segments or the number of segments. Each data point consists of a flag-value pair $(f_i, a_i)$, where $f_i$ is a binary indicator indicating whether the $i$-th element serves as the head of a segment, and $a_i$ represents the value intended for scan. The binary operator applied to these pairs is defined as $(f_i, a_i) \oplus^s (f_j, a_j) \coloneqq (f_i \mid f_j, \text{ if } f_j = 1 \text{ then } a_j \text{ else } a_i \oplus a_j)$, where $\oplus \coloneqq +$ in this example. Each vertical grey box represents an individual thread. Every thread executes a sequence of steps, some of which necessitate awaiting outcomes from other threads. The algorithm utilizes a *reduce-then-scan* strategy, which can be vitalized as an "hourglass" shape comprising an up-sweep phase and a down-sweep phase. The binary tree-based parallel algorithm requires a considerable amount of inter-thread data communication, but this latency remains low in shared memory, making it suitable for GPU parallelization.

i.e. $R_k(Y_{ki}) \in R_k(Y_{ki'}) \forall Y_{ki} > Y_{ki'}$. Define $S[\boldsymbol{\nu}]$ as *scan* on a arbitrary vector $\boldsymbol{\nu}$. Taking the denominator of the second term of gradient as an example, we can write the terms within stratum $k$ as

$$\left\{ \sum_{r \in R_k(Y_{ki})} \exp\left(\mathbf{x}'_{kr}\boldsymbol{\beta}\right) \right\}_{i=1}^{n_k} = S\left[ \left\{ \exp\left(\mathbf{x}'_{ki}\boldsymbol{\beta}\right) \right\}_{i=1}^{n_k} \right], \tag{4.19}$$

each with only cost $\mathcal{O}(n_k)$ operations. In this way, we can reformulate the second term of gradient as $2K$ scans and reduce the time complexity from $\mathcal{O}(K) + \mathcal{O}(\sum_{k=1}^{K} n_k^2)$ to $\mathcal{O}(K) + \mathcal{O}(\sum_{k=1}^{K} n_k)$. Similarly, the time complexity of Hessian can be reduced to $\mathcal{O}(K) + \mathcal{O}(\sum_{k=1}^{K} n_k)$. However, this can still

be inefficient when encountering highly-stratified data, due to the factor $K$ in the time complexity.

To this end, we further combine the data across strata $\mathbf{x}_k \in \mathbb{R}^{(n_k \times P)}$ together as $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_K \end{bmatrix} \in \mathbb{R}^{N \times P}$, and turn $K$ scans to a single $K$-segmented scan. Define head flag vectors $\mathbf{f}_k = \{f_{ki}\}_{i=1}^{n_k}$ such that

$$f_{ki} = \begin{cases} 1 & \text{for } i = 1 \text{ and} \\ 0 & \text{for } i = 2, \ldots, n_k, \end{cases} \tag{4.20}$$

for all $k$ and $S_{\text{seg}}[\boldsymbol{\nu}]$ as *segmented scan* on the vector $\boldsymbol{\nu}$ with the head flag $\mathbf{f} = (f_{11}, \ldots, f_{1n_1}, f_{21}, \ldots, f_{Kn_K})$. Then we can write the denominator of the second term in the gradient across all $K$ strata simply as

$$S_{\text{seg}}\left[ \left\{ \exp\left(\mathbf{x}_s' \boldsymbol{\beta}\right) \right\}_{s=1}^{N} \right], \tag{4.21}$$

where $s = \sum_{k'=1}^{k-1} n_{k'} + i$ for $i = 1, \ldots, n_k$ and $k = 1, \ldots, K$. Further, define exponentiation (exp), multiplication ($\times$) and division ($/$) as element-wise operations on vectors. The univariate gradient (4.15) and Hessian (4.16) falls out as:

$$g'(\beta_j) = -\boldsymbol{\delta}^\top \mathbf{X}_j + \boldsymbol{\delta}^\top \frac{S_{\text{seg}}[\mathbf{N_1}]}{S_{\text{seg}}[\mathbf{D}]} \text{ and} \tag{4.22}$$

$$g''(\beta_j) = \boldsymbol{\delta}^\top \frac{S_{\text{seg}}[\mathbf{N_2}]}{S_{\text{seg}}[\mathbf{D}]} - \boldsymbol{\delta}^\top \left( \frac{S_{\text{seg}}[\mathbf{N_1}]}{S_{\text{seg}}[\mathbf{D}]} \times \frac{S_{\text{seg}}[\mathbf{N_1}]}{S_{\text{seg}}[\mathbf{D}]} \right), \tag{4.23}$$

where

$$\mathbf{D} = \exp(\mathbf{X}\boldsymbol{\beta}), \tag{4.24}$$

$$\mathbf{N_1} = \mathbf{X}_j \times \exp(\mathbf{X}\boldsymbol{\beta}) \text{ and} \tag{4.25}$$

$$\mathbf{N_2} = \mathbf{X}_j \times \mathbf{X}_j \times \exp(\mathbf{X}\boldsymbol{\beta}). \tag{4.26}$$

Note that the vectors $\mathbf{X}_j$, $\mathbf{X}\boldsymbol{\beta}$, $\mathbf{f}$, $\mathbf{D}$, $\mathbf{N_1}$, and $\mathbf{N_2}$ in the above equations are all of the same dimension ($N \times 1$). Furthermore, we can avoid the costly matrix-vector multiplication $\mathbf{X}\boldsymbol{\beta}$ in CCD by updating

$[\mathbf{X}\boldsymbol{\beta}]_s$ as shown below:

$$[\mathbf{X}\boldsymbol{\beta}]_s^{(\text{new})} = [\mathbf{X}\boldsymbol{\beta}]_s^{(\text{old})} + x_{sj}\Delta\beta_j, \tag{4.27}$$

for $s = 1, 2, \ldots, N$. Regarding the vector-vector inner-products, such as $\boldsymbol{\delta}^\top \mathbf{X}_j$, we efficiently compute these through a parallelized reduction (i.e., sum) as $\sum_{s=1}^{N} \delta_s x_{sj}$. Note that $\mathbf{X}$ is generally sparse where most of $x_{sj}$ are zeros, resulting in relatively small computations for the above processes. Now we have successfully reduced the time complexity of the univariate gradient (4.15) and Hessian (4.16) to $\mathcal{O}(N)$, and the time complexity of one cycle of our CCD algorithm to $\mathcal{O}(NP)$ under the stratified Cox model, regardless of the number of strata or the data distribution among them.

To parallelize the evaluation of the gradient (4.15) and Hessian (4.16) on a GPU, we generate $S = B \times IPT \times G$ threads. Here, $B$ represents the number of concurrent threads forming a thread block, $IPT$ is the number of input items per thread, and $G = \lceil \frac{N}{B \times IPT} \rceil$ indicates the number of thread blocks. The block size $B$ and thread grain size $IPT$ are constrained by hardware and are tunable constants. In our implementation, we choose $B = 128$ and $IPT = 15$ for the binary-tree based kernel following the parameter settings in CUB, and $B = 256$ and $IPT = 1$ for other kernels we have developed to compute Equations (4.24) through (4.27). Within each thread block, $B$ threads can communicate through shared memory and execute computations in parallel. For instance, the threads first read the nonzero entries of $\mathbf{X}_j$ and $\mathbf{X}\boldsymbol{\beta}^{(\text{old})}$, and then update $\mathbf{X}\boldsymbol{\beta}^{(\text{new})}$, $\mathbf{D}$, $\mathbf{N_1}$, and $\mathbf{N_2}$ concurrently using Equations (4.27), (4.24), (4.25) and (4.26), respectively. Subsequently, the threads read the values of $\mathbf{D}$, $\mathbf{N_1}$, $\mathbf{N_2}$, and the head flag $\mathbf{f}$, and perform an efficient *segmented scan* operation (as detailed in 4.2.6) using the resources of shared memory. Finally, the threads execute the element-wise transformations and binary reductions as shown in Equations (4.22) and (4.23), completing the evaluation of the gradient and Hessian for $\beta_j$ within a single iteration of the CCD process.

## 4.3 Results

We assess the computational efficiency of our GPU implementation versus a similar CPU implementation in fitting the extensions of Cox model to large-scale sample sizes comparable to those we often see in EHR and claims data sources. To accomplish this, we conduct a series of synthetic experiments fitting stratified Cox models across various numbers of strata and sample sizes. We then replicate a real-world study to evaluate the efficacy of antihypertensive drug classes using a stratified Cox model based on matching subjects through their propensity scores. Finally, we investigate the time-varying effect of a safety outcome associated with the above drug classes employing a Cox model with time-varying coefficients. Our computational setup comprises a system equipped with a 10-core, 3.3 GHz Intel(R) Xeon(R) W-2155 CPU, and an NVIDIA Quadro GV100 boasting 5120 CUDA cores and 32GB RAM, capable of achieving up to 7.4 Tflops double-precision floating-point performance.

### 4.3.1 Synthetic experiments

In this section, we illustrate the computational performance of our GPU implementation compared to the corresponding CPU implementation on stratified Cox model in the highly optimized `R` package `Cyclops` (Suchard et al., 2013; Mittal et al., 2014). We simulate a binary design matrix $\mathbf{X}$ under two sample sizes and two dimensions. We randomly set 5% of the entries uniformly to be 1s to mimic the sparse pattern in the observational healthcare data. For each sample size, we stratify the data into various number of strata. We generate the $P$-dimensional $\boldsymbol{\beta}$ from a standard normal distribution with mean zero and unit variance, where we set 80% of the entries to 0 to induce model parameter sparsity as well, that is

$$\beta_j \sim N(0,1) \times \text{Bernoulli}(0.80) \; \forall \; j.$$

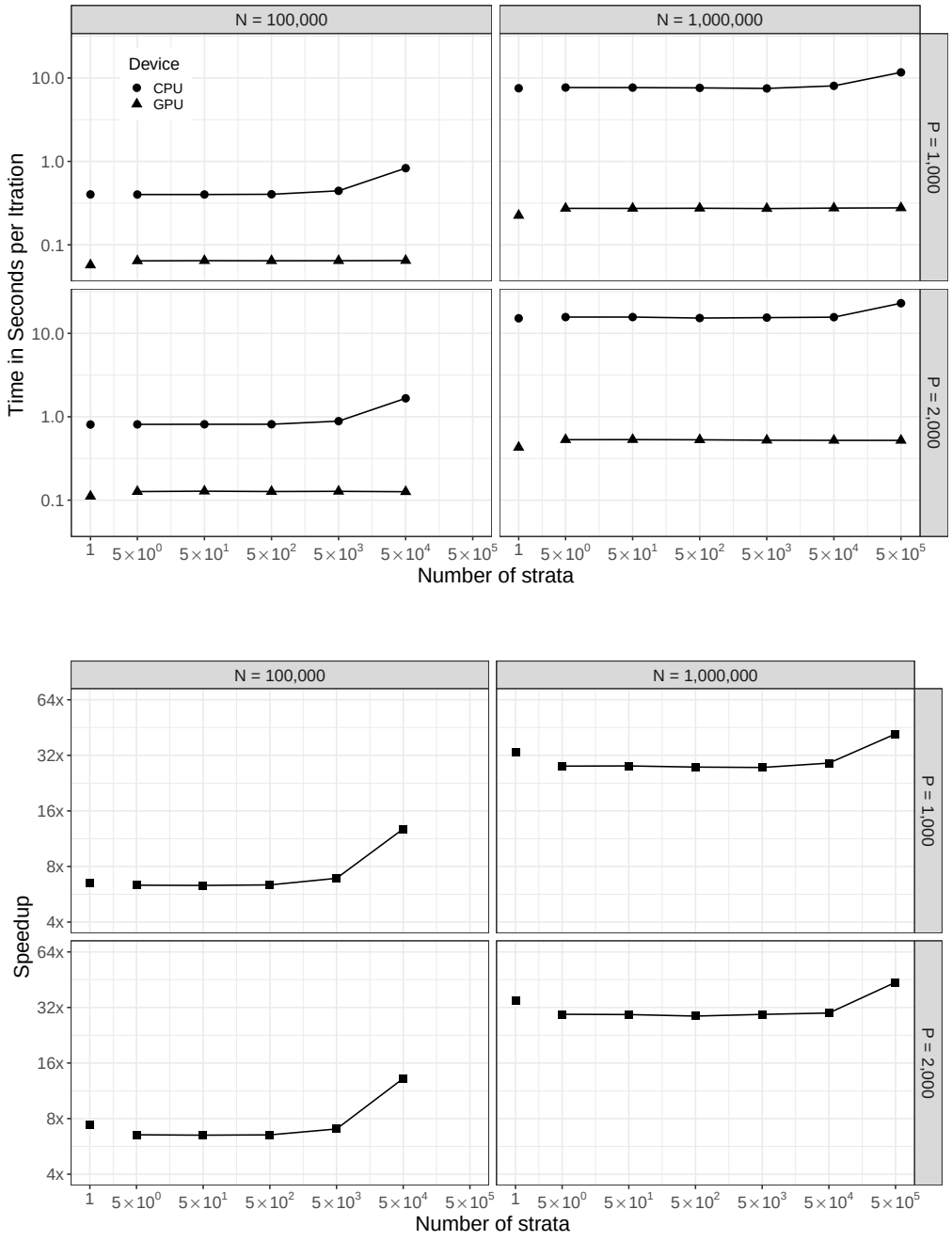



Figure 4.2: Runtimes per iteration (in seconds) and speedup of the GPU implementation relative to the CPU implementation for stratified Cox models with a fixed $\ell_1$ penalty. We conduct experiments using two sample sizes: $N = 10^5$ and $N = 10^6$, and two dimensions: $P = 1{,}000$ and $P = 2{,}000$, both with a sparsity of 95%. For each sample size, we fit stratified Cox models with various number of strata. In addition, we include our previous work on the unstratified Cox model for comparison purposes.

We fit these simulants using a stratified Cox model under a fixed $\ell_1$ penalty with $\gamma = \sqrt{2}$, as further executing the cross-validation does alter the relative performance of the different implementations here. Figure 4.2 presents the runtime per iteration and the speedup of GPU paralllelization relative to a similar CPU implementation. We also include the performance of GPU parallelization of an unstratified $\ell_1$ Cox model (Yang et al., 2023) for comparison purposes, as shown in the left-most point in all figures. We observe that the GPU paralllelization delivers up to a 13-fold speedup and a 43-fold speedup with $100,000$ samples and one million samples, respectively. To put it into an absolute scale, we reduce the total fitting time of a stratified Cox model with one million samples and $50,000$ strata from 17 minutes to 23 seconds. We also observe a rapid increase of runtimes on the CPU with the most highly stratified data, while the GPU approach consistent performs well across slightly and highly stratified data. At $K = N/2$, speculative instruction execution, namely poor branch prediction of where strata start, takes it toll on CPU performance. The relevant performance improvements align with the computational complexity analysis discussed in 4.2.7.

### 4.3.2 Cardiovascular effectiveness of antihypertensive drug classes

In this section, we explore the relative effectiveness of two major hypertension drug classes to demonstrate the advantages of massive parallelization within a real-world example. While most treatment recommendations derive from randomized clinical trials that offer limited comparisons between a few agents, large-scale observational studies can provide valuable insights for estimating the relative risk of important cardiovascular and safety outcomes associated with different drug classes. We primarily focus on two major hypertension drug classes, angiotensin-converting enzyme inhibitors (ACEIs) and thiazide or thiazide-like diuretics (THZs), and one important cardiovascular outcome, hospitalization for heart failure. We follow a comparative new-user cohort design, as outlined in the Large-scale Evidence Generation and Evaluation across a Network of Databases for Hypertension (LEGEND-HTN) study (Suchard et al., 2019).

For our experiments here, we use patient health records on antihypertensive drug classes from the Optum® de-identified Electronic Health Record dataset (Optum EHR). This dataset encompasses

information from 85 million individuals in the United States who are commercially or Medicare insured. We extract a subsample of $946,911$ patients diagnosed with hypertension. Among these individuals, 77% initiate treatment with an ACEI, while the remaining 23% initiate treatment with a THZ. We consider the relative effectiveness of THZ and ACEI in preventing hospitalization for heart failure as main treatment covariate. Additionally, we include $9,976$ baseline patient characteristic covariates, encompassing clinical condition, drug exposure, and medical procedure. The patient characteristic covariates exhibit an average sparsity of 97%, indicating that only 3% of the entries contain non-zero values. We construct a propensity score model incorporating all baseline covariates (Tian et al., 2018) and stratify the individuals into varying number of equally-sized strata based on their propensity score estimates. Specifically, we consider three commonly used strata configurations: $S = 5, 10$, and 20. Finally, we apply the stratified Cox proportional hazards model to estimate the hazard ratio (HR) between THZ and ACEI initiation with respect to the risk of hospitalization for heart failure. We include all patient characteristics and treatment covariates in the stratified Cox model with $\ell_1$ regularization on all covariates except the treatment covariate, and employ a 10-fold cross-validation to search for optimal tuning parameters. Considering that statistical inference under $\ell_1$ regularization remains challenging, we calculate 95% bootstrapped percentile intervals (BPIs) from bootstrap samples. Table 4.1 reports the runtimes (in hours) for both our GPU parallelization and a similar CPU implementation and the HRs estimates with their BPIs. Our GPU parallelization delivers an 11-fold speedup across varying numbers of strata.

### 4.3.3 Time-varying effect of safety outcome of antihypertensive drug classes

In this section, we investigate the time-varying effect of a safety outcome of ACEIs: cough (Dicpinigaitis, 2006). We utilize a similar comparative new-user cohort design and the same dataset as in the previous section to conduct this analysis. Rather than stratifying individuals using propensity scores, we employ a 1:1 matching strategy for THZ and ACEI new-users. After propensity score matching, we retain a total of $407,828$ patients who developed cough with $9,666$ baseline covariates for further analysis. The Kaplan-Meier plots in Figure 4.3 show the survival of patients with cough over time.

| | Runtime (hr) | | |
|---|---|---|---|
| Strata | GPU | CPU | HR (95% [BPI]) |
| 5 | 1.41 | 16.2 | 0.83 (0.72, 0.93) |
| 10 | 1.38 | 15.8 | 0.81 (0.73, 0.90) |
| 20 | 1.57 | 18.0 | 0.82 (0.73, 0.93) |

Table 4.1: Computation times (in hours) for fitting the stratified Cox model for the optimum value of $\ell_1$ regularization parameter (found through 10-fold cross-validation) across GPU and CPU implementations and the HR estimates and their 95% BPIs comparing the relative risk of hospitalization for heart failure risk between new-users of thiazide or thiazide-like duretics and angiotensin-converting enzyme inhibitors. The data contains $946,911$ individuals with $9,977$ covariates. We stratify the individuals into varying number of equally-sized strata based on propensity score estimates.

We can see that the relative risks of THZ and ACEI exposure on cough are different within 10 days of initiating treatment and after 10 days. Therefore, we treat the treatment covariate as two time-varying covariates accordingly. To assess the time-varying effects, we transform the Cox model with a time-varying coefficient to a stratified Cox model with two strata using the method explained in 4.2.3 and 4.2.2. The stratified Cox model for cough contains $812,432$ observations and $9,668$ covariates after appropriate data wrangling. We apply an $\ell_1$ penalty on all covariates except the treatment covariates (within 10 days of initiating treatment and after 10 days). We then performed a 10-fold cross-validation to identify the optimal tuning parameters. Our GPU parallelization significantly reduces the analysis time from 21.6 hours to 1.86 hours.

Through massive parallelization, we find that initializing with a THZ has less risk of developing cough than initializing with an ACEI after 10 days (HR 0.67, 95% BPI $0.65 - 0.69$), while both medications exhibit similar risks within 10 days (HR 0.98, 95% BPI $0.87 - 1.13$). Note that the HRs correspond to the exponentiated $\beta$ coefficients. The difference between two coefficients is 0.38 with a 95% BPIs ranging from 0.30 to 0.48, reinforcing that the relative risks of developing cough differ within and after 10 days of initialization. Previously, dealing with time-varying coefficients on this scale posed challenges in the original LEGEND-HTN study due to computational time burdens. However, with the implementation of our GPU parallelization, this complex analysis has now become feasible.

## 4.4 Discussion

This article presents an efficient GPU implementation of stratified Cox and the time-varying Cox models for analyzing large-scale time-to-event data investigating time-varying effects, building upon our prior work on standard Cox models (Yang et al., 2023). We implement this efficient method in the open-source `R` package `Cyclops` (Suchard et al., 2013). In simulation studies, the GPU implementation for the stratified Cox model shows a speedup of 43 times compared to the equivalent CPU implementation, even with up to 1 million samples. In our real-world examples examining the time-varying risk of developing cough, massive parallelization significantly reduces the total runtimes from nearly a day to less than 2 hours.

The main idea of this article is to simplify a complex model into a more concise one and enhance its implementation efficiency through the innovative use of massive parallelization. In particular, we demonstrate that a Cox model with time-varying coefficients can be transformed into a Cox model with time-varying covariates when using discrete time-to-event data. Moreover, we identify that the Cox model with time-varying covariates shares the similar partial likelihood structure as the stratified Cox model. Finally, we apply an efficient segmented scan algorithm to address the same computational bottleneck of the three extended models due to the similar partial likelihood structure. This algorithm significantly accelerates likelihood, gradient, and Hessian evaluations, thereby improving the overall efficiency of our approach.

There is potential for improvement in our approach. First, both transformations detailed in 4.2.3 and 4.2.2 require augmenting the original design matrix due to repeating the time-independent covariates over time or creating additional time-varying covariates to estimate the time-varying effect in multiple time intervals. While data augmentation can be memory-inefficient, it is possible to save memory by developing mappings on the original data, as both data augmentation techniques involve duplications of the original data. Additionally, the cyclic coordinate descent algorithm we use in the case with $\ell_1$ regularization may lack of rigorous theoretical proof of convergence. Although it is possible that the results to be oscillate around the global minimum, we have not observed this issue

in our experiments.

Nonetheless, this work provides valuable tools for massively sized Cox models with stratification and time-varying effects. In recent years, the growing interest in leveraging large-scale observational healthcare data sources has driven the demand for such models. For instance, Shoaibi et al. (2021) explored the comparative effectiveness of famotidine in hospitalized COVID-19 patients using data from the COVID-19 Premier Hospital Database, which encompasses approximately 700 hospitals throughout the United States. Similarly, Kim et al. (2020) studied the comparative safety and effectiveness of two popular anti-osteoporosis medications on $324,049$ patients across three electronic medical records and six claims databases. The ability to rapidly perform analyses using such large models has opened doors to comprehensive sensitivity assessments regarding stratification designs. Moreover, this work initiates large-scale comparative effectiveness and safety studies with time-varying effects, addressing the limitation of tools for the time-varying Cox models.

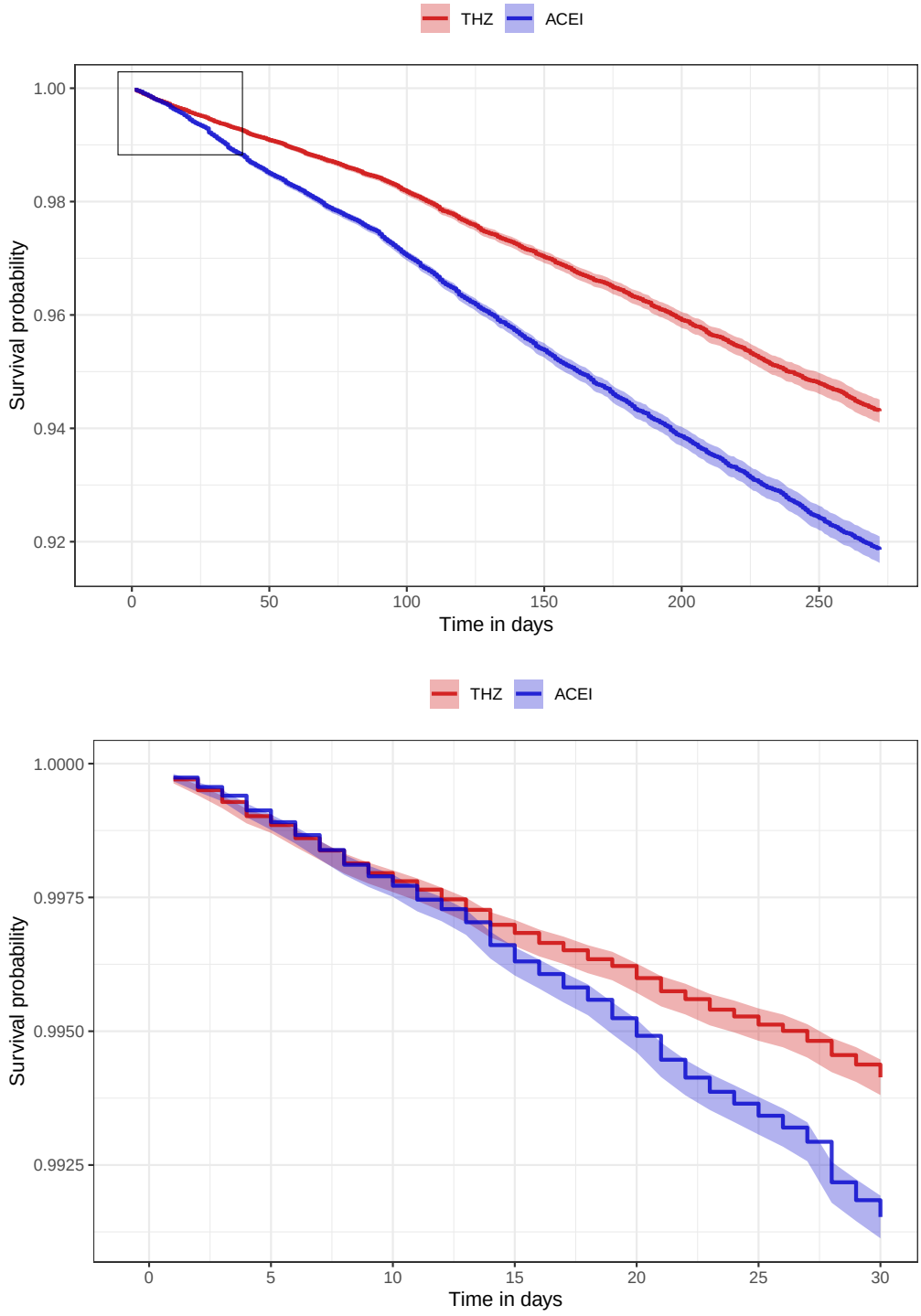



Figure 4.3: Kaplan Meier plots showing survival of patients with cough over time. The red line represents THZ exposure, while the blue line represents ACEI exposure. The upper plot displays all the data included in the analysis, while the lower plot focuses on data within 30 days of exposure. We can see that the risks of developing cough within the first 10 days are similar in both groups, while ACEI carries a higher risk compared to THZ after 10 days.

# CHAPTER 5

# Memory-Efficient Massive Sample-Size Pooled Logistic Regression

## 5.1 Introduction

Time-to-event data and longitudinal data are commonly used in epidemiology and medical studies. The former follows subjects for a period and records the time until an event of interest happens, such as the occurrence of a specific disease or a particular disease stage. The latter involves tracking the same measurements repeatedly over time, such as laboratory results. Many statistical methods have been proposed to analysis time-to-event data and longitudinal data. The Kaplan-Meier survival estimator is the most commonly used non-parametric method (Kaplan and Meier, 1958). While it is straightforward to calculate and easy to understand, it cannot estimate survival adjusted for covariates. The Cox proportional hazards model is commonly used for analyzing time-to-event data and can also handle time-dependent covariates in its extended form (Cox, 1972; Crowley and Hu, 1977). While it is especially useful for estimating the hazard ratio, it suffers from limitations in estimating the baseline hazard and direct hazard. More importantly, the proportional hazard model is generally not collapsible, which means the marginal model when omitting one covariate will not have the same regression coefficients for the remaining covariates (Martinussen and Vansteelandt, 2013).

The pooled logistic regression model, on the other hand, is a fully parametric approach that provides the estimates of baseline hazard and direct hazard, while also exhibiting collapsibility (Guo and Geng, 1995). This approach simply fits a logistic regression using pooled data of repeated

observations from time-to-event data and longitudinal data. Taking a binary outcome indicating the occurance of event of interest in each time interval, pooled logistic regression is able to estimate the hazard of the event of interest during each time interval. Furthermore, it allows for adjustments of time-varying covariates collected from longitudinal data.

## 5.2 Methods

### 5.2.1 Pooled logistic regression

The pooled logistic regression using the pooling of repeated cbservations (PRO) method was developed to analyze repetitive observations gathered from the Framingham Study cohort (D'Agostino et al., 1990). The PRO approach treats relatively short time intervals as miniature follow-up studies, combining observations across all intervals to explore the short-term progression of the disease. For instance, let's consider monitoring $n_0$ patients at risk for a disease over $K$ time intervals. During the $k$-th interval, we observe $d_k$ events and a loss of $d'k$ patients for $k = 1, 2, \ldots, K$. We can represent the number of at-risk patients at the start of the $k$-th interval as $n_k$, where $n_k = n_{k-1} - d_{k-1} - d'_{k-1}$. The PRO approach combines at-risk subjects at the start of every interval, resulting in a total of $\sum_{k=0}^{K-1} n_k$ subjects. Finally, we apply logistic regression using a pooled sample comprising $\sum_{k=0}^{K-1} n_k$ observations and $\sum_{k=1}^{K} d_k$ events.

The basic assumption for this approach is that the number of events $d_k$ follows a binomial distribution given $n_k$:

$$d_k | n_k \sim \text{Binomial}(n_k, h(t_k)) \quad \text{and is independent for all } k, \tag{5.1}$$

where $h(t_k)$ represents the discrete hazard rate (Efron, 1988). The discrete hazard rate is defined as the conditional probability of observing an event during the $k$-th time interval, given that the individual has remained event-free until the start of the $k$-th interval. This enables us to estimate the

discrete hazard rate using logistic regression:

$$\log\left(\frac{h(t_k|\mathbf{x}(t_{k-1}))}{1-h(t_k|\mathbf{x}(t_{k-1}))}\right) = \beta_0(t_k) + \mathbf{x}(t_k)^\top \boldsymbol{\beta}, \tag{5.2}$$

where $\mathbf{x}(t_k)$ represents a collection of longitudinal measured covariates.

### 5.2.2 Relation of pooled logistic regression to time-dependent Cox regression

The study by D'Agostino et al. (1990) has demonstrated that, under appropriate assumptions, the pooled logistic model and the Cox model with time-dependent covariates are asymptotically equivalent.

In the pooled logistic regression, we can solve the discrete hazard rate from Equation 5.2 as follows:

$$h(t_k|\mathbf{x}(t_k)) = \frac{1}{1+\exp\{-\left[\beta_0(t_k) + \mathbf{x}(t_k)^\top \boldsymbol{\beta}_{\text{plr}}\right]\}}. \tag{5.3}$$

Let's denote $G(\mathbf{x}(t_k)) = \exp\{-\left[\beta_0(t_k) + \mathbf{x}(t_k)^\top \boldsymbol{\beta}_{\text{plr}}\right]\}$ for simplicity. When $G(\mathbf{x}(t_k))$ is small, we can approximate $h(t_k)$ through Taylor expansion:

$$\begin{aligned} h(t_k|\mathbf{x}(t_k)) &= \frac{1}{1+G(\mathbf{x}(t_k))} \\ &= 1 - G(\mathbf{x}(t_k)) + G(\mathbf{x}(t_k))^2 - G(\mathbf{x}(t_k))^3 \cdots \\ &= 1 - G(\mathbf{x}(t_k)) + o(G(\mathbf{x}(t_k))) \\ &\approx 1 - G(\mathbf{x}(t_k)). \end{aligned} \tag{5.4}$$

Note that the requirement of small values of $G(\mathbf{x}(t_k))$ can be guaranteed by assuming the probability of event occurrence in short intervals is small.

Likewise, we can approximate the hazard function under the Cox model with time-dependent covariates in a very similar way. In the continuous survival model, we can express the relationship

between the survival function and hazard function as:

$$\log S(t) = -\int_0^t h(u)\,du. \tag{5.5}$$

For the Cox regression, the hazard function is defined as:

$$\log h(t|\mathbf{x}(t)) = h_0(t)\exp[\mathbf{x}(t)^\top \boldsymbol{\beta}_{\text{cox}}], \tag{5.6}$$

where $h_0(t)$ is unspecified baseline hazard. Then the hazard function become

$$h(t_k|\mathbf{x}(t_k)) \;=\; \frac{S(t_k|\mathbf{x}(t_k))}{S(t_{k-1}|\mathbf{x}(t_{k-1}))} = \exp\left\{-\int_{t_{k-1}}^{t_k} h_0(u)\exp[\mathbf{x}(t_k)^\top \boldsymbol{\beta}_{\text{cox}}]\,du\right\}. \tag{5.7}$$

Let $H(\mathbf{x}(t_k)) = \int_{t_{k-1}}^{t_k} h_0(u)\exp[\mathbf{x}(t_k)^\top \boldsymbol{\beta}_{\text{cox}}]\,du$ for the sake of simplicity, we can further approximate the hazard function by its Taylor expansion:

$$\begin{aligned} h(t_k|\mathbf{x}(t_k)) &= \exp\left(-H(\mathbf{x}(t_k))\right) \\ &= 1 - H(\mathbf{x}(t_k)) + \frac{H(\mathbf{x}(t_k))^2}{2!} - \frac{H(\mathbf{x}(t_k))^3}{3!}\cdots \\ &= 1 - H(\mathbf{x}(t_k)) + o(H(\mathbf{x}(t_k))) \\ &\approx 1 - H(\mathbf{x}(t_k)) \end{aligned} \tag{5.8}$$

when the value of $H(\mathbf{x}(t_k))$ is small. Similarly, this requirement can be justified by assuming that the probability of an event occurring in a short interval is small.

Thus, the pooled logistic model and the Cox model with time-dependent covariates become asymptotically equivalent when the time intervals are short and the probability of an event within each interval is small. Both approximations involve two components: the baseline hazard and the effect of explanatory variables. The key difference between the two models is the way they treat baseline hazard. In the pooled logistic regression, we are able to directly quantify the baseline hazard by estimating the intercept term $\beta_0$, while the Cox model leaves the baseline hazard $h_0(t)$

unspecified.

### 5.2.3 Estimating both fixed effect and time-varying effect using discrete time-to-event data

We consider a specific scenario in which we only measure the covariates once but assume that these covariates have time-varying effects on the outcome. For patient $i$, let $Y_i$ denote the survival time and $\delta_i$ be the event indicator for $i = 1, 2, \ldots, N$. The baseline covariate is denoted as $z_{ij}$ for individual $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, P$. While we may collect continuous time-to-event data, adapting a pooled logistic regression calls for its discretization into time intervals.. We consider $K$ time intervals divided by time points $t_k$ for $k = 0, 1, 2, \ldots, K$, where $0 = t_0 < t_1 < t_2 < \cdots < t_K = \max(Y_i)$. Let's assume there are known time-varying effects $f_0(t)$ on the intercept and $f_j(t)$ on baseline covariates for $j = 1, 2, \ldots, P$. In practice, $f_j(t)$ can take various forms, such as linear time, squared time, cubed time, etc. Consequently, we formulate the pooled logistic regression as follows:

$$\log\left(\frac{h_i(t_k)}{1-h_i(t_k)}\right) \quad = \quad \beta_0 + \gamma_0 f_0(t_k) + \sum_{j=1}^{P} \beta_j z_{ij} + \sum_{j=1}^{P} \gamma_j z_{ij} f_j(t_k) \tag{5.9}$$

for $i = 1, 2, \ldots, N$ and $t_k \leq Y_i$. The parameters to be estimated include $\boldsymbol{\theta} = [\beta_0, \beta_1, \ldots, \beta_P, \gamma_0, \gamma_1, \ldots, \gamma_P]^\top$, where the $\beta$s represent the fixed effect of baseline covariates on the outcome, and the $\gamma$s represent the time-varying effect.

### 5.2.4 Data wrangling for efficient pooled logistic regression

Fitting the pooled logistic regression as illustrated by Equation 5.9 necessitates to augment the original design martix $\mathbf{Z}$ from $\mathbb{R}^{N\times(1+P)}$ to $\mathbf{Z}^{(\text{aug})} \in \mathbb{R}^{N'\times 2(1+P)}$, where $N' = \sum_{i=1}^{N} K_i$ and $K_i$ is the largest

integer such that $t_{K_i} \leq Y_i$:

$$\mathbf{Z}^{(\text{aug})} = \begin{bmatrix} 1 & z_{11} & \dots & z_{1P} & g(t_1) & f_1(t_1)z_{11} & \dots & f_P(t_1)z_{1P} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & z_{11} & \dots & z_{1P} & g(t_{K_1}) & f_1(t_{K_1})z_{11} & \dots & f_P(t_{K_1})z_{1P} \\ 1 & z_{21} & \dots & z_{2P} & g(t_1) & f_1(t_1)z_{21} & \dots & f_P(t_1)z_{2P} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & z_{21} & \dots & z_{2P} & g(t_{K_2}) & f_1(t_{K_2})z_{21} & \dots & f_P(t_{K_2})z_{2P} \\ 1 & z_{31} & \dots & z_{3P} & g(t_1) & f_1(t_1)z_{31} & \dots & f_P(t_1)z_{3P} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{N1} & \dots & z_{NP} & g(t_{K_N}) & f_1(t_{K_N})z_{N1} & \dots & f_P(t_{K_N})z_{NP} \end{bmatrix}.$$

The reason for requiring a wider design matrix is that we are estimating both fixed effects and time-varying effects for intercept and all baseline covariates. The operation of doubling the design matrix is not the main issue, the real challenge arises from pooling observations over time repeatedly. This can lead to $N'$ being significantly larger than $N$, with the extreme scenario being $N' = NK$.

The full log-likelihood of the pooled logistic regression is derived as follows:

$$\begin{aligned} l(\boldsymbol{\theta}) &= \sum_{i=1}^{N}\sum_{k=1}^{K_i}\Big\{\delta_i \log h_i(t_k) + (1-\delta_i)\log(1-h_i(t_k))\Big\} \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K_i}\left\{Y_i \log\left(\frac{h_i(t_k)}{1-h_i(t_k)}\right) + \log(1-h_i(t_k))\right\} \\ &= \sum_{s=1}^{N'}\left\{Y_s\,[\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta}]_s - \log\Big\{1+\exp\left([\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta}]_s\right)\Big\}\right\}. \end{aligned} \tag{5.10}$$

In practice, we can avoid constructing the large augmented design matrix $\mathbf{Z}^{(\text{aug})}$ when estimating the log-likelihood, as the values of baseline covariates remain constant over time. Given that we are working with discrete time-to-event data, we can list all possible values of time-varying effects at $K$ specified time points in $K$ vectors: $\mathbf{f}_k = [f_0(t_k), f_1(t_k), \dots, f_P(t_k)]$ for $k = 1, 2, \dots, K$. Then, the

entries of $\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta}$ can be derived as follows:

$$\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta} = \begin{bmatrix} \mathbf{z}_1^\top\boldsymbol{\beta} + \left(\mathbf{z}_1^\top \odot \mathbf{f}_1\right)\boldsymbol{\gamma} \\ \vdots \\ \mathbf{z}_1^\top\boldsymbol{\beta} + \left(\mathbf{z}_1^\top \odot \mathbf{f}_{K_1}\right)\boldsymbol{\gamma} \\ \mathbf{z}_2^\top\boldsymbol{\beta} + \left(\mathbf{z}_2^\top \odot \mathbf{f}_1\right)\boldsymbol{\gamma} \\ \vdots \\ \mathbf{z}_2^\top\boldsymbol{\beta} + \left(\mathbf{z}_2^\top \odot \mathbf{f}_{K_2}\right)\boldsymbol{\gamma} \\ \mathbf{z}_3^\top\boldsymbol{\beta} + \left(\mathbf{z}_3^\top \odot \mathbf{f}_1\right)\boldsymbol{\gamma} \\ \vdots \\ \mathbf{z}_N^\top\boldsymbol{\beta} + \left(\mathbf{z}_N^\top \odot \mathbf{f}_{K_N}\right)\boldsymbol{\gamma} \end{bmatrix}, \tag{5.11}$$

where $\odot$ represents the element-wise vector-vector multiplication.

We utilize the cyclic coordinate descent (CCD) algorithm to iterate through each covariates and employ a Newton approach to update the covariate while holding all other covariates contants (Genkin et al., 2007; Mittal et al., 2014). Specifically, we iteratively update the entries of $\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta}$ using CCD, as shown below:

$$[\mathbf{Z}\boldsymbol{\theta}]_s^{(\text{new})} = \begin{cases} [\mathbf{Z}\boldsymbol{\theta}]_s^{(\text{old})} + z_{sj}\Delta\beta_j & \text{for fixed effects } j = 0, 1, \ldots, P \\ [\mathbf{Z}\boldsymbol{\theta}]_s^{(\text{old})} + z_{sj}f_j(t_k)\Delta\gamma_j & \text{for time-varying effects } j = 0, 1, \ldots, P \end{cases}, \tag{5.12}$$

where $s = \sum_{i'=1}^{i-1} K_{i'} + k$ for $k = 1, \ldots, K_i$ and $i = 1, \ldots, N$.

Through this data wrangling, we have successfully saved memory by reducing the design matrix from $\mathbf{Z}^{(\text{aug})} \in \mathbb{R}^{N' \times 2(1+P)}$ back to $\mathbb{R}^{N \times (1+P)}$, resulting in a $\frac{2N'}{N}$-fold memory reduction.

## 5.3 Results

In this section, we investigate the risk of developing cough of two antihypertensive drug classes: angiotensin-converting enzyme inhibitor (ACEI) and thiazide or thiazide-like diuretic (THZ). We extract data from the Optum® de-identified Electronic Health Record dataset. Our analysis includes a total of $N = 407,828$ hypertension patients who initiated ACEis or THZs treatment with $9,666$ baseline covariates. Observation time span from 1 to $3,631$ days, with a median duration of 89 days. We discretize time-to-event data into 5-day intervals, 10-day intervals, and 20-day intervals, resulting in a pooled dataset with $N' = 12,229,077$ rows, $N' = 6,168,188$ rows, and $N' = 3,233,896$ rows, respectively. Through efficient data wrangling, we avoid constructing an extensive design matrix in $\mathbb{R}^{N' \times P}$, resulting in a remarkable 8-fold to 30-fold reduction in memory usage.

We include all baseline characteristics and treatment covariates in the pooled logistic model with $\ell_1$ regularization on all covariates except the treatment covariate and intercept. We employ a 10-fold cross-validation to search for optimal tuning parameters. Table 5.1 reports the hazard ratio (HR) estimates with their 95% bootstrapped percentile intervals (BPIs) across different settings of discretizing time-to-event data.

| Interval days | $N'$ | HR (95% [BPI]) |
|---|---|---|
| 5 | 12,229,077 | 0.68 (0.41, 0.74) |
| 10 | 6,168,188 | 0.68 (0.62, 0.96) |
| 20 | 3,233,896 | 0.68 (0.63, 0,96) |

Table 5.1: The data contains $407,828$ individuals with $9,667$ covariates (including treatment). We discretize time-to-event data into 5-day intervals, 10-day intervals, and 20-day intervals.

## 5.4 Discussion and Future work

This chapter outlines a memory-efficient approach for fitting pooled logistic regression models using discrete time-to-event data. The methodology has been implemented in the open-source `R` package `Cyclops`. This work presents a valuable tool that enables the pooled logistic regression analysis on

massive sample sizes even with limited computing resources.

There are several potential improvements to consider. Firstly, while the current work is capable of exploring the time-varying effects on baseline covariates, it lacks the inclusion of time-varying covariates collected longitudinally. We could extend the current work to incorporate time-varying covariates through more intricate data wrangling, although there might not be as much opportunity for memory saving in handling time-varying covariates due to their non-constant nature over time. Second, the current work primarily focuses on enhancing memory efficiency. Parallelization could assist in improving time efficiency when fitting massive sample-sized pooled logistic regression models. Specifically, the computation of $\mathbf{Z}^{(\text{aug})}\boldsymbol{\theta}$ could be accelerated using intricate operations including segmented-reduction.

# CHAPTER 6

# Discussion and Future directions

This dissertation presents three efficient statistical computing advances for conducting large-scale observational analyses. In Chapter 3, I developed a time- and memory-efficient GPU implementation to fit massive sample-sized Cox proportional hazards models and Fine-Gray subdistribution hazard models, achieving one to two orders of magnitude speedup. Chapter 4 extended the work from the previous chapter to incorporate stratified Cox models, Cox models with time-varying covariates, and Cox models with time-varying coefficients, using an efficient parallel segmented scan algorithm. Chapter 5 was inspired by the need for doubly-robust casual inference in large-scale observation data and delivers a memory-efficient approach for fitting massive sample-sized pooled logistic regression. Taken together, these advances empower the observational healthcare researcher with many more analysis choices than previously tractable at scale.

There is room to further improve the computational efficiency of PS-adjusted outcome modeling at scale in the future. Firstly, while we usually employ an iterative optimization approach in fitting outcome models, this dissertation focuses on improving computational efficiency within a single iteration. To further enhance scalability, one may explore iterative optimization approaches other than the cyclic coordinate descent (CCD) method to improve convergence. Given that the CCD approach used in this dissertation lacks a convergence proof in the context of $\ell_1$ regularization, one may explore other iterative optimization approaches that exhibit faster convergence to reduce computational demands.

Secondly, while this dissertation emphasizes on parallelization within a single outcome model, there is potential to extend parallelization to a distributed framework. Observational data are often

stored in different location, and privacy concerns may hamper the data sharing across machines. Therefore, it is worthwhile to explore a communication-efficient distributed method for large-scale and high-dimensional Cox model. Such an endeavor has the potential to foster collaborative research and enhance the scalability of PS-adjusted outcome modeling on a broader scale.

Finally, such a growth of analysis choices begs the emerging question: which model or models should one use for a specific research study tied to a set of observational data sources to provide the most reliable evidence?

I conclude this dissertation by laying possible future research to help answer this conundrum. In comparative effectiveness estimation, one aims to estimate the effect of one exposure (the target treatment) on the risk of an outcome, compared to another exposure (the comparator treatment) (Schuemie et al., 2020). Existing observational health care data, such as administrative claims and electronic health records, enabled numerous large-scale comparative effectiveness studies (Chan You et al., 2021; Kim et al., 2020). However, various sources of systematic error threaten the reliability of estimation results in such observational study, including study design, selection bias, confounding, model misspecification, and measurement error.

An issue in study design is the neglect of competing risks. Competing risks arise in time-to-event data when subjects are also at risk of another type of event that precludes the event of primary interest. For example, many comparative safety studies model the adverse event of certain medication as the primary outcome, so death due to non-medication cause is a natural competing risk. Such competing risk information is often available in observational data, but the standard Cox proportional hazards model used in comparative effectiveness study ignores competing events. In contrast, the Fine-Gray subdistribution hazard model takes the competing risk events into consideration.

There exists a large body of prior work comparing the Cox model to the Fine-Gray model for relative risk estimation (Ranstam and Robertsson, 2017; Noordzij et al., 2013; Lau et al., 2009; Hsu et al., 2017; Feinstein et al., 2012; Feakins et al., 2018) when competing risk information is available. Most of this work lives in the setting of randomized trials in which systematic error is minimized, was developed for a specific use-case or lacks real-world evidence verification. Moreover, while one

can assume that the true relative risk of negative controls is 1, the true relative risk of the real-world positive controls (whose true relative risk are not equal to one) are often unknown.

Using the tools I developed in Chapter 3, one can now explore whether including a competing risk under the Fine-Gray model for comparative effectiveness estimation reduces systematic error as compared to the usual Cox model in the large-scale setting.

Consider evaluating the Cox and Fine-Gray models in comparative effect estimation using cohort study of new-users. Each control can be defined as a quintuplet containing target, comparator, outcome, competing outcome, and effect size. One can take death as competing outcome for all negative and positive controls. We can also define competing outcome for a specific primary outcome, for instance, considering acute myocardial infarction as a competing outcome for the hospitalization of heart failure. For negative controls, neither the target treatment nor the comparator treatment is believed to cause the corresponding outcome, thus the true hazard ratio of the target treatment versus the comparator treatment on the outcome is 1 (Lipsitch et al., 2010; Schuemie et al., 2014). For example, let $y$ denote the outcome counts, the footnote $t$ denote the target treatment, and $c$ denote the comparator treatment, then one has

$$y_{ti}(t) = y_{ci}(t), \forall i \in \{1, ..., N(t)\}, \tag{6.1}$$

where $N(t)$ is the number of subjects at risk at the beginning of an interval. Therefore the true hazard ratio become

$$h = E\left(\lim_{\Delta t \to 0} \frac{\sum_i y_{ti}([t, t+\Delta t])/N_t(t)}{\sum_i y_{ci}([t, t+\Delta t])/N_c(t)}\right) = 1. \tag{6.2}$$

One can create synthetic positive controls to address the limitations associated with real-world positive controls (Schuemie et al., 2018). One can construct synthetic positive controls based on negative controls by intentionally injecting simulated additional outcomes in the target treatment cohort until it achieved the desired hazard ratio. For example, if one wants to generate three positive

controls with target hazard ratios of $h = 1.5$, $h = 2$, and $h = 4$ starting from a negative control with hazard ratio of $h = 1$, one artificially increases the number of outcomes $y_{ti}$ in the target cohort until the desired $h$ was achieved.

For each combination of method, database, and control, one then generates a hazard ratio as an effect size estimate and a 95% confidence interval to quantify the uncertainty associated with the estimate. One then computes a set of metrics based on these estimates to evaluate the performance of regression and classification for both negative and positive controls.

- Mean squared error (MSE): MSE between the log of hazard ratio point-estimate and the log of the true hazard ratio.
- Coverage: How often the true hazard ratio is within the 95% confidence interval.
- Mean precision: Mean precision is computed as $1/(\text{standard error})^2$. A higher precision is desired as it indicates a narrower confidence interval.
- Area under the Receiver Operating Characteristic Curve (AUC): AUC checks the ability to discriminate between positive controls and negative controls based on the point estimate of hazard ratio.
- Type 1 error and type 2 error: Type 1 error measures how often the null was rejected (at $\alpha = 0.05$) for negative controls, while the type 2 error measures how often the null was not rejected (at $\alpha = 0.05$) for positive controls.
- Non-estimable: The proportion of the controls that the method was not able to produce an estimate, since in some cases one cohort observed zero outcomes or there were no objects left after propensity score matching.

Confounding adjustment is necessary to achieve cohort balance in a new-user cohort study design using observational data. The propensity score (PS) estimates the probability that a subject was assigned to a specific treatment (target treatment). Various confounding adjustment strategies based on PS can be used in evaluation setting. All evaluations capture a large set of baseline patient

covariates in the year prior to exposure for PS estimation, and PS values are computed using large-scale regularized logistic regression (Suchard et al., 2013).

One can consider four variants of PS adjustment (Austin, 2011):

- 1-on-1 matching.
- Variable ratio matching (Rassen et al., 2012).
- Stratification.
- Inverse probability of treatment weighting (IPTW) with trimming (Brookhart et al., 2013; Xu et al., 2010).

One can also consider two variations of the survival outcome model:

- An outcome model (Cox or Fine-gray regression) with treatment as the only covariate.
- A full outcome model including all covariates that are also included in the PS with regularization on all variables except the treatment variable to achieve double robustness (Funk et al., 2011).

After executing the above design variants to both Cox and Fine-Gray models, we can evaluate the experiment's performance metrics to identify the superior model and design choice across diverse scenarios. For example, we can assess model choices for estimating medication effectiveness in the context of competing risks by reviewing the experiment results for heart failure with stroke or acute myocardial infarction as competing events. Similarly, we can investigate design choices for analyzing rare and acute safety outcomes by examining the experiments related to acute pancreatitis.

Bibliography

Austin, P. C. (2011). An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate Behavioral Research 46*(3), 399–424.

Barney, B. et al. (2010). Introduction to parallel computing. *Lawrence Livermore National Laboratory 6*(13), 10.

Beam, A. L., S. K. Ghosh, and J. Doyle (2016). Fast Hamiltonian Monte Carlo using GPU computing. *Journal of Computational and Graphical Statistics 25*(2), 536–548.

Blelloch, G. E. (1990). *Vector Models for Data-Parallel Computing*. Cambridge: MIT press.

Bramante, C. T., N. E. Ingraham, T. A. Murray, S. Marmor, S. Hovertsen, J. Gronski, C. McNeil, R. Feng, G. Guzman, N. Abdelwahab, et al. (2021). Metformin and risk of mortality in patients hospitalised with COVID-19: a retrospective cohort analysis. *The Lancet Healthy Longevity 2*(1), e34–e41.

Brookhart, M. A., R. Wyss, J. B. Layton, and T. Stürmer (2013). Propensity score methods for confounding control in nonexperimental research. *Circulation: Cardiovascular Quality and Outcomes 6*(5), 604–611.

Casella, G., M. Ghosh, J. Gill, and M. Kyung (2010). Penalized regression, standard errors, and Bayesian lassos. *Bayesian Analysis 5*(2), 369–411.

Chan You, S., H. M. Krumholz, M. A. Suchard, M. J. Schuemie, G. Hripcsak, R. Chen, S. Shea, J. Duke, N. Pratt, C. G. Reich, et al. (2021). Comprehensive comparative effectiveness and safety of first-line $\beta$-blocker monotherapy in hypertensive patients: A large-scale multicenter observational study. *Hypertension 77*(5), 1528–1538.

Chatterjee, A. and S. N. Lahiri (2011). Bootstrapping lasso estimators. *Journal of the American Statistical Association 106*(494), 608–625.

Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological) 34*(2), 187–202.

Crowley, J. and M. Hu (1977). Covariance analysis of heart transplant survival data. *Journal of the American Statistical Association 72*(357), 27–36.

D'Agostino, R. B., M.-L. Lee, A. J. Belanger, L. A. Cupples, K. Anderson, and W. B. Kannel (1990). Relation of pooled logistic regression to time dependent Cox regression analysis: the Framingham heart study. *Statistics in Medicine 9*(12), 1501–1515.

Dicpinigaitis, P. V. (2006). Angiotensin-converting enzyme inhibitor-induced cough: ACCP evidence-based clinical practice guidelines. *Chest 129*(1), 169S–173S.

Efron, B. (1988). Logistic regression, survival analysis, and the Kaplan-Meier curve. *Journal of the American Statistical Association 83*(402), 414–425.

Feakins, B. G., E. C. McFadden, A. J. Farmer, and R. J. Stevens (2018). Standard and competing risk analysis of the effect of albuminuria on cardiovascular and cancer mortality in patients with type 2 diabetes mellitus. *Diagnostic and Prognostic Research 2*(1), 1–9.

Feinstein, M., H. Ning, J. Kang, A. Bertoni, M. Carnethon, and D. M. Lloyd-Jones (2012). Racial differences in risks for first cardiovascular events and noncardiovascular death: the atherosclerosis risk in communities study, the cardiovascular health study, and the multi-ethnic study of atherosclerosis. *Circulation 126*(1), 50–59.

Fine, J. P. and R. J. Gray (1999). A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association 94*(446), 496–509.

Flay, B. R., A. Biglan, R. F. Boruch, F. G. Castro, D. Gottfredson, S. Kellam, E. K. Mościcki, S. Schinke, J. C. Valentine, and P. Ji (2005). Standards of evidence: Criteria for efficacy, effectiveness and dissemination. *Prevention Science 6*(3), 151–175.

Funk, M. J., D. Westreich, C. Wiesen, T. Stürmer, M. A. Brookhart, and M. Davidian (2011). Doubly robust estimation of causal effects. *American Journal of Epidemiology 173*(7), 761–767.

Genkin, A., D. D. Lewis, and D. Madigan (2007). Large-scale Bayesian logistic regression for text categorization. *Technometrics 49*(3), 291–304.

Guo, J. and Z. Geng (1995). Collapsibility of logistic regression coefficients. *Journal of the Royal Statistical Society. Series B (Methodological)*, 263–267.

Harris, M., S. Sengupta, and J. D. Owens (2007). Parallel prefix sum (scan) with CUDA. *GPU Gems 3*(39), 851–876.

He, K., J. Zhu, J. Kang, and Y. Li (2022). Stratified Cox models with time-varying effects for national kidney transplant patients: A new blockwise steepest ascent method. *Biometrics 78*(3), 1221–1232.

Hernán, M. A. and J. M. Robins (2006). Estimating causal effects from epidemiological data. *Journal of Epidemiology & Community Health 60*(7), 578–586.

Hernán, M. A. and J. M. Robins (2016). Using big data to emulate a target trial when a randomized trial is not available. *American Journal of Epidemiology 183*(8), 758–764.

Holbrook, A. J., P. Lemey, G. Baele, S. Dellicour, D. Brockmann, A. Rambaut, and M. A. Suchard (2021). Massive parallelization boosts big Bayesian multidimensional scaling. *Journal of Computational and Graphical Statistics 30*(1), 11–24.

Hripcsak, G., P. B. Ryan, J. D. Duke, N. H. Shah, R. W. Park, V. Huser, M. A. Suchard, M. J. Schuemie, F. J. DeFalco, A. Perotte, et al. (2016). Characterizing treatment pathways at scale using the OHDSI network. *Proceedings of the National Academy of Sciences 113*(27), 7329–7336.

Hripcsak, G., M. J. Schuemie, D. Madigan, P. B. Ryan, and M. A. Suchard (2021). Drawing reproducible conclusions from observational clinical data with OHDSI. *Yearbook of Medical Informatics 30*(01), 283–289.

Hsu, J. Y., J. A. Roy, D. Xie, W. Yang, H. Shou, A. H. Anderson, J. R. Landis, C. Jepson, M. Wolf, T. Isakova, et al. (2017). Statistical methods for cohort studies of CKD: survival analysis in the setting of competing risks. *Clinical Journal of the American Society of Nephrology 12*(7), 1181–1189.

Kaplan, E. L. and P. Meier (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association 53*(282), 457–481.

Kawaguchi, E. S., J. I. Shen, M. A. Suchard, and G. Li (2021). Scalable algorithms for large competing risks data. *Journal of Computational and Graphical Statistics 30*(3), 685–693.

Kim, Y., Y. Tian, J. Yang, V. Huser, P. Jin, C. G. Lambert, H. Park, S. C. You, R. W. Park, P. R. Rijnbeek, et al. (2020). Comparative safety and effectiveness of alendronate versus raloxifene in women with osteoporosis. *Scientific Reports 10*(1), 1–10.

Ko, S., H. Zhou, J. J. Zhou, and J.-H. Won (2022). High-performance statistical computing in the computing environments of the 2020s. *Statistical Science 37*(4), 494–518.

Lau, B., S. R. Cole, and S. J. Gange (2009). Competing risk regression models for epidemiologic data. *American Journal of Epidemiology 170*(2), 244–256.

Lipsitch, M., E. T. Tchetgen, and T. Cohen (2010). Negative controls: a tool for detecting confounding and bias in observational studies. *Epidemiology (Cambridge, Mass.) 21*(3), 383.

Little, R. J. and D. B. Rubin (2000). Causal effects in clinical and epidemiological studies via potential outcomes: concepts and analytical approaches. *Annual Review of Public Health 21*(1), 121–145.

Madigan, D., P. Ryan, S. Simpson, and I. Zorych (2010). Bayesian methods in pharmacovigilance. *Bayesian Statistics 9*, 421–438.

Madigan, D., P. E. Stang, J. A. Berlin, M. Schuemie, J. M. Overhage, M. A. Suchard, B. Dumouchel, A. G. Hartzema, and P. B. Ryan (2014). A systematic statistical approach to evaluating evidence from observational studies. *Annual Review of Statistics and Its Application 1*, 11–39.

Martinussen, T. and S. Vansteelandt (2013). On collapsibility and confounding bias in Cox and Aalen regression models. *Lifetime Data Analysis 19*(3), 279–296.

Merrill, D. (2015). Cub. *NVIDIA Research*.

Merrill, D. and M. Garland (2016). Single-pass parallel prefix scan with decoupled look-back. *NVIDIA, Tech. Rep. NVR-2016-002*.

Micikevicius, P. (2009). 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pp. 79–84.

Mittal, S., D. Madigan, R. S. Burd, and M. A. Suchard (2014). High-dimensional, massive sample-size Cox proportional hazards regression for survival analysis. *Biostatistics 15*(2), 207–221.

Mueller-Using, S., T. Feldt, F. S. Sarfo, and K. A. Eberhardt (2016). Factors associated with performing tuberculosis screening of HIV-positive patients in Ghana: LASSO-based predictor selection in a large public health data set. *BMC Public Health 16*(1), 1–8.

Ngwa, J. S., H. J. Cabral, D. M. Cheng, M. J. Pencina, D. R. Gagnon, M. P. LaValley, and L. A. Cupples (2016). A comparison of time dependent Cox regression, pooled logistic regression and cross sectional pooling with simulations and an application to the framingham heart study. *BMC Medical Research Methodology 16*, 1–12.

Nickolls, J., I. Buck, M. Garland, and K. Skadron (2008). Scalable parallel programming with CUDA. *Queue 6*(2), 40–53.

Noordzij, M., K. Leffondré, K. J. van Stralen, C. Zoccali, F. W. Dekker, and K. J. Jager (2013). When do we need competing risks methods for survival analysis in nephrology? *Nephrology Dialysis Transplantation 28*(11), 2670–2677.

NVIDIA (2023). CUDA C++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

Ranstam, J. and O. Robertsson (2017). The Cox model is better than the Fine and Gray model when estimating relative revision risks from arthroplasty register data. *Acta Orthopaedica 88*(6), 578–580.

Rassen, J. A., A. A. Shelat, J. Myers, R. J. Glynn, K. J. Rothman, and S. Schneeweiss (2012). One-to-many propensity score matching in cohort studies. *Pharmacoepidemiology and Drug Safety 21*, 69–80.

Ray, W. A. (2003). Evaluating medication effects outside of clinical trials: new-user designs. *American Journal of Epidemiology 158*(9), 915–920.

Rennich, S. (2011). CUDA C/C++ streams and concurrency. In *GPU Technology Conference*.

Robins, J. M. and A. Rotnitzky (1992). Recovery of information and adjustment for dependent censoring using surrogate markers. In *AIDS Epidemiology*, pp. 297–331. Springer.

Rosenbaum, P. R. and D. B. Rubin (1983). The central role of the propensity score in observational studies for causal effects. *Biometrika 70*(1), 41–55.

Rosenbaum, P. R. and D. B. Rubin (1984). Reducing bias in observational studies using subclassification on the propensity score. *Journal of the American Statistical Association 79*(387), 516–524.

Ryan, P. B., M. J. Schuemie, S. Gruber, I. Zorych, and D. Madigan (2013). Empirical performance of a new user cohort method: lessons for developing a risk identification and analysis system. *Drug Safety 36*, 59–72.

Schneeweiss, S. and J. Avorn (2005). A review of uses of health care utilization databases for epidemiologic research on therapeutics. *Journal of Clinical Epidemiology 58*(4), 323–337.

Schuemie, M. J., M. S. Cepeda, M. A. Suchard, J. Yang, Y. Tian, A. Schuler, P. B. Ryan, D. Madigan, and G. Hripcsak (2020). How confident are we about observational findings in healthcare: a benchmark study. *Harvard Data Science Review 2*(1).

Schuemie, M. J., G. Hripcsak, P. B. Ryan, D. Madigan, and M. A. Suchard (2018). Empirical confidence interval calibration for population-level effect estimation studies in observational healthcare data. *Proceedings of the National Academy of Sciences 115*(11), 2571–2577.

Schuemie, M. J., P. B. Ryan, W. DuMouchel, M. A. Suchard, and D. Madigan (2014). Interpreting observational studies: why empirical calibration is needed to correct p-values. *Statistics in Medicine 33*(2), 209–218.

Schwartz, J. T. (1980). Ultracomputers. *ACM Transactions on Programming Languages and Systems (TOPLAS) 2*(4), 484–521.

Sengupta, S., M. Harris, M. Garland, et al. (2008). Efficient parallel scan algorithms for GPUs. *NVIDIA, Santa Clara, CA, Tech. Rep. NVR-2008-003 1*(1), 1–17.

Shoaibi, A., S. P. Fortin, R. Weinstein, J. A. Berlin, and P. Ryan (2021). Comparative effectiveness of famotidine in hospitalized COVID-19 patients. *The American College of Gastroenterology 116*(4), 692–699.

Shortreed, S. M. and A. Ertefaie (2017). Outcome-adaptive lasso: variable selection for causal inference. *Biometrics 73*(4), 1111–1122.

Suchard, M. A., M. J. Schuemie, H. M. Krumholz, S. C. You, R. Chen, N. Pratt, C. G. Reich, J. Duke, D. Madigan, G. Hripcsak, et al. (2019). Comprehensive comparative effectiveness and safety of first-line antihypertensive drug classes: a systematic, multinational, large-scale analysis. *The Lancet 394*(10211), 1816–1826.

Suchard, M. A., S. E. Simpson, I. Zorych, P. Ryan, and D. Madigan (2013). Massive parallelization of serial inference algorithms for a complex generalized linear model. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 23*(1), 10.

Suchard, M. A., Q. Wang, C. Chan, J. Frelinger, A. Cron, and M. West (2010). Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. *Journal of Computational and Graphical Statistics 19*(2), 419–438.

Terenin, A., S. Dong, and D. Draper (2019). GPU-accelerated Gibbs sampling: a case study of the Horseshoe Probit model. *Statistics and Computing 29*(2), 301–310.

Thackham, M. and J. Ma (2020). On maximum likelihood estimation of the semi-parametric Cox model with time-varying covariates. *Journal of Applied Statistics 47*(9), 1511–1528.

Therneau, T. M., P. M. Grambsch, T. M. Therneau, and P. M. Grambsch (2000). *The Cox Model*. Springer.

Tian, L., D. Zucker, and L. Wei (2005). On the Cox model with time-varying regression coefficients. *Journal of the American Statistical Association 100*(469), 172–183.

Tian, Y., M. J. Schuemie, and M. A. Suchard (2018). Evaluating large-scale propensity score performance through real-world and synthetic data experiments. *International Journal of Epidemiology 47*(6), 2005–2014.

Tutz, G., M. Schmid, et al. (2016). *Modeling Discrete Time-to-event Data*. Springer.

Wu, T. T., K. Lange, et al. (2008). Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics 2*(1), 224–244.

Xu, S., C. Ross, M. A. Raebel, S. Shetterly, C. Blanchette, and D. Smith (2010). Use of stabilized inverse propensity scores as weights to directly estimate relative risk and its confidence intervals. *Value in Health 13*(2), 273–277.

Xu, Y. and W. Yin (2017). A globally convergent algorithm for nonconvex optimization based on block coordinate update. *Journal of Scientific Computing 72*(2), 700–734.

Yang, J., M. J. Schuemie, X. Ji, and M. A. Suchard (2023). Massive parallelization of massive sample-size survival analysis. *Journal of Computational and Graphical Statistics* (just-accepted), 1–23.

Zhang, Z., J. Reinikainen, K. A. Adeleke, M. E. Pieterse, and C. G. Groothuis-Oudshoorn (2018). Time-varying covariates and coefficients in Cox regression models. *Annals of Translational Medicine 6*(7).

Zhou, H., K. Lange, and M. A. Suchard (2010). Graphics processing units and high-dimensional optimization. *Statistical Science: A Review Journal of the Institute of Mathematical Statistics 25*(3), 311.

Zucker, D. M. and A. F. Karr (1990). Nonparametric survival analysis with time-dependent covariate effects: a penalized partial likelihood approach. *The Annals of Statistics 18*(1), 329–353.