Faster Biobank-Scale Genomics through Sparse Linear Algebra on Genotype Representation Graph

1st Yifan Li†
Department of Computer Science
Cornell University
Ithaca, USA
yl3722@cornell.edu

2nd Qingyao Sun†
School of Operations Research and Information Engineering
Cornell University
Ithaca, USA
qs234@cornell.edu

3rd Drew C. DeHaas
Department of Computational Biology
Cornell University
Ithaca, USA
dcd239@cornell.edu

4th Xinzhu Wei
Department of Computational Biology
Cornell University
Ithaca, USA
xw583@cornell.edu

5th Giulia Guidi*
Department of Computer Science
Cornell University
Ithaca, USA
gg434@cornell.edu

* Corresponding author. E-mail: gg434@cornell.edu

† These authors contributed equally to this work.

I. Introduction

Over the past decade, the scale of population genomics data has increased dramatically. The genotype representation graph (GRG) was recently introduced as a compact substrate for biobank-scale genotype data [2, 3, 7], achieving 30–50× compression compared to the standard `.vcf.gz` format. Central operations in population genetics can be performed directly on the GRG through efficient traversal, running orders of magnitude faster than equivalent operations on tabular data [2, 3, 7]. However, achieving ideal scalability and hardware portability needed for biobank-scale analyses remains an open challenge.

Hardware accelerators like GPUs can deliver superior performance through massive parallelism, but unleashing that power requires a parallel-friendly computing scheme tailored to the specific problem. GRG traversal is no exception: its graph-structured reuse of shared ancestry does not map onto GPU hardware as naturally, and realizing GRG's advantages on GPUs requires a computing scheme suited to this structure. In this work, we argue that sparse linear algebra primitives can serve as a bridge between GRG traversal and efficient accelerator-based computation [4]. The inherent sparsity of GRG-native operations makes sparse linear algebra a natural fit for mapping them onto accelerator hardware, allowing developers to leverage highly optimized sparse primitives from hardware vendors and research communities [1, 5, 6].

II. Method

A. Background

A Genotype Representation Graph (GRG) is a Directed Acyclic Graph (DAG) where each variant allele is associated with a node, and edges connect samples to the variants they carry. This graph encodes phased whole-genome polymorphisms by explicitly capturing shared ancestry among samples via ancestral nodes, from which compression arises. See Fig 2 for an illustration.

The central computational primitive in GRG-based analyses is the genotype matrix–vector product. Given the genotype matrix $G \in \{0,1\}^{n\times m}$, where n is the number of haplotypes and m is the number of mutations, the goal is to compute Gv for some vector $v \in \mathbb{R}^m$, or $G^\top w$ for $w \in \mathbb{R}^n$.

B. GRG-based Computation as Sparse Linear Algebra

Instead of formulating the matrix-vector product through whole GRG traversal as in prior work [2], we directly express these products as sparse linear algebra on the GRG's adjacency matrix A. As a DAG, the nodes in a GRG can be relabeled with a topological order. In this ordering, A is strictly lower triangular, and it can be shown that the genotype matrix is a submatrix of the transposed reachability matrix $T^\top := (I - A)^{-\top}$. Concretely, $G = ST^\top M^\top$, and the genotype matrix–vector product decomposes as:

$$Gv = S(T^\top(M^\top v)), \qquad G^\top w = M(T(S^\top w)),$$

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, using NERSC award ASCR-ERCAP0030076. This material is based upon work supported by the National Science Foundation under Grant IIS-2435801.

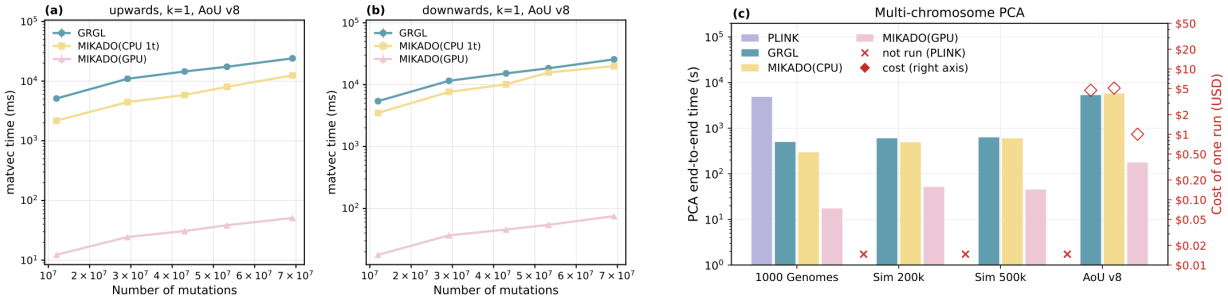


Fig. 1. Performance comparison. Our method is labeled as MIKADO(CPU) and MIKADO(GPU). GRGL is the GRG baseline. Figure (a): Kernel runtime for up direction traversal on five chromosomes from the All of us v8 phased dataset. Figure (b): Kernel runtime for down direction traversal on the same dataset as (a). Figure (c): PCA end-to-end runtime on four different datasets. PLINK is the non-GRG baseline.

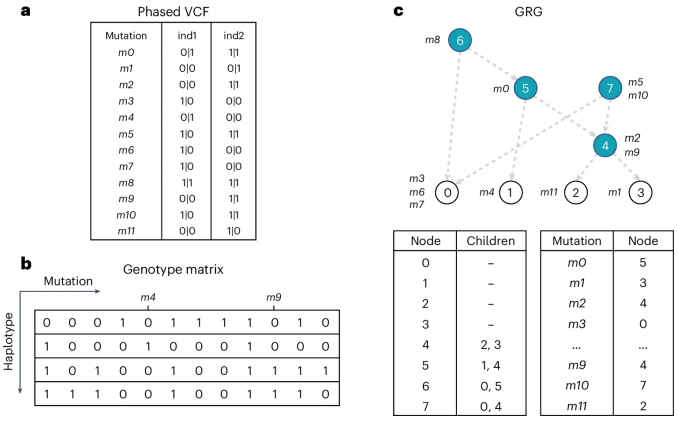


Fig. 2. Same genotype data in three different formats. **a**, A standard VCF file. **b**, A genotype matrix. **c**, A GRG representation. Courtesy of [2].

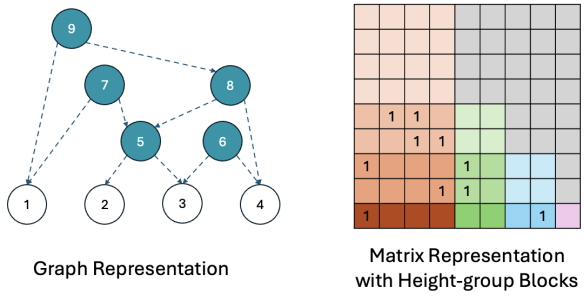


Fig. 3. A GRG and its corresponding matrix representation.

where S and M are selector matrices that map between graph nodes and samples or mutations, respectively. Thus, applying S, M, and their transposes corresponds to simple gather and scatter operations. The computational core is applying $T = (I - A)^{-1}$ to a vector, which is a sparse triangular solve (SpTRSV).

Furthermore, note that A is strictly *block* lower triangular: since the GRG is a DAG, each node has a well-defined *height*, defined as its longest path to a leaf. Partitioning nodes by height yields $H + 1$ levels, where H is the graph diameter, and A inherits a block structure from this partition: edges connect only higher-level nodes to lower-level nodes, so each level depends only on levels already processed, as shown in Fig 3. Forward/backward substitution thus reduces to a sequence of SpMVs (one per pair of levels) that propagate values up or down through the graph. Each SpMV within a level is independent and embarrassingly parallel, and the small graph diameter keeps the number of sequential synchronization steps modest. This formulation exposes fine-grained parallelism: SpMVs within a height level execute concurrently, and our pipelined wavefront scheduling strategy allows blocks at level k to begin executing as soon as their specific upstream dependencies are finalized, rather than waiting for the entire preceding level.

III. Result

We conduct experiments on NERSC Perlmutter GPU Nodes (one EPYC 7763 CPU, four A100 GPUs), with a set of simulated chromosomes (500K individuals). For a single traversal kernel, GPU with block formulation achieves up to 470× speedup compared to CPU baseline. For end-to-end PCA runs, GPUs achieve more than 10× speedup, as shown in Fig 1.

IV. Discussion

In this work, we show that GRG-based computation can be reformulated as sparse linear algebra, which is more broadly applicable to genotype DAG structures. Our two-step reformulation—from graph traversal to SpTRSV, and from SpTRSV to a pipelined blocked SpMV— decouples computation from graph representation, recasting GRG traversal as a primitive (SpMV) for which mature, hardware-tuned routines already exist. Such approach inherits performance and portability from a rapidly evolving accelerator software stack. With modern compute devices such as GPUs, our method can achieve speedups of hundreds of times compared to baselines. We expect this to accelerate genomics research and open up possibilities for new methods.

Acknowledgment

We thank Max Xiaohang Zhao, Adam R. Boyko, and Shaila A. Musharoff for insightful discussions and helpful feedback on this work.

References

[1] AMD. *ROCm*. swmath: 25796. URL: https://github.com/ROCm/rocm-libraries.

[2] Drew DeHaas, Ziqing Pan, and Xinzhu Wei. "Enabling efficient analysis of biobank-scale data with genotype representation graphs". In: *Nature computational science* 5.2 (2025), pp. 112–124.

[3] Drew DeHaas et al. "General, orders-of-magnitude faster whole-genome analysis with genotype representation graphs". In: *bioRxiv* (2026). DOI: 10.64898/2026.04.10.717786. eprint: https://www.biorxiv.org/content/early/2026/04/11/2026.04.10.717786.full.pdf. URL: https://www.biorxiv.org/content/early/2026/04/11/2026.04.10.717786.

[4] Giulia Guidi et al. "Parallel string graph construction and transitive reduction for de novo genome assembly". In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. IEEE, 2021, pp. 517–526.

[5] Intel. *MKL*. swmath: 6975. URL: http://software.intel.com/en-us/intel-mkl.

[6] NVIDIA, Péter Vingelmann, and Frank H. P. Fitzek. *CUDA*. swmath: 3258. URL: https://developer.nvidia.com/cuda-toolkit.

[7] Aditya Syam, Chris Adonizio, and Xinzhu Wei. "Fast Phenotype Simulation for Genotype Representation Graphs". In: *Bioinformatics Advances* (2026), vbag040.