Combining OpenMP, MPI, and Homomorphic Encryption for a Privacy-Preserving, Parallelized GWAS

Noah L. Schrick
12 December 2022

Introduction

Advancements in numerous focus areas of biotechnology and bioinformatics has led to a growth and adoption of new research algorithms and a push for large, real-world studies [2], [14-15]. Among these studies, the identification of significantly different single-nucleotide polymorphisms (SNPs) through Genome-Wide Association Studies (GWAS) has seen increased prioritization. As the authors of [15] discuss, historically, these results have been shared among researchers and government entities to foster innovation and encourage collaboration for future research. However, the authors of [6] discuss that the concern for privacy has slowed the exchange of biomedical data. Since identifying information can be extracted from GWAS data, there are large concerns for the implications of leakage of this medical data [2-6, 9-10, 15-16, 18].

With the growth and adoption of Biobanks, and with the increased focus on GWAS, the study size has increased significantly. The authors of [1] discuss that studies can involve the solution of billions to trillions of correlated generalized least squares problems. The authors continue to state that for representative studies, there are multiple, large grids of GLS problems that must be solved, requiring a large amount of computing power. The authors state that there is an associated runtime of $O(n^3)$ per problem. Increasing the size of GWAS studies has led to a push for better, optimized approaches that can be employed.

This work presents an approach to mitigate both difficulties in the form of fully homomorphic encryption (FHE) to address the privacy concerns, and parallelization through MPI to distribute across multiple compute nodes, and OpenMP to parallelize at the node to address the runtime concerns.

Related Work

With the increased concerns for privacy of medical data, numerous works have addressed various privacy aspects in a number of ways. One approach is through policies and regulations, and the authors of [2], [6], and [15] each highlight advantages and disadvantages of employing a bureaucratic approach. The authors of [2] discuss multiple entity areas of concern, noting that each area requires a different solution. These entity areas range from users and researchers, to the storage and operation of data, the data owners, and the computation of the data itself. A challenge presented by a policy and regulation approach is that the enforcement and implementation varies country by country, and entity by entity. A government entity requires different approaches than a commercial entity, or a research-driven entity. Another approach is through Secure Multiparty Computation. The authors of [6] and [16] both present discussions of this approach, and provide results of their solutions. This approach introduces infrastructure overhead and communication costs through independent computation servlets and process servlets..

A third approach is through Garbled Circuits (GC), as seen by the authors of [2], [4], and [6]. This approach serves as a cryptographic protocol through gates that only reveal the output of the communication. The authors of [2] and [6] also discuss Secure Hardware as an approach to mitigating privacy concerns. This approach maintains a higher barrier of entry in both capital cost and in specialized skill-sets.

One of the two more popular approaches is through Differential Privacy, as discussed by the authors of [2], [6], [9], and [15]. This approach works with statistical data, rather than the raw data. Through differential privacy, it is possible to share the patterns and differences of data, rather than sharing the sensitive data itself. Alternatively, there is Homomorphic Encryption, as discussed ih the works of [2], [4], [5], [10], [16], and [18]. This approach allows for computation to be performed on the encrypted data itself. This allows for data owners to encrypt the sensitive medical data prior to sharing, and allows for the computation to be securely performed without requiring the unencrypted, cleartext data. This approach is accessible, with numerous open-source implementations. This approach has seen greater usage in GWAS, and has historically been successful.

The table below lists a few of the privacy-preserving options that could be employed for GWAS computations, along with the disadvantages that led this work to opt for homomorphic encryption.

Privacy-Preserving Approach	**Disadvantages**
Policies and Regulations	• Enforcement and implementation varies country-by-country, entity-by-entity
Secure Multiparty Computation (SMPC)	• Infrastructure overhead and communication costs ○ Independent process servlets ○ Independent computation servlets
Garbled Circuit (GC)	• Communication Costs
Secure Hardware	• High barrier to entry ○ Capital Cost ○ Specialized knowledge and skillsets
Differential Privacy	• Still in early-stage adoption with GWAS data

Table 1: Privacy-preserving approaches

Regarding runtime improvements, there are two main areas of focus. The first of these areas is to employ machine learning, as discussed by the authors of [7], [12], and [14]. The authors of [7] discuss various supervised machine learning approaches. The authors discuss approaches such as regression, classification, ensemble learning, and neural networks, and highlight other works that have employed these approaches. The authors continue to discuss the future applications of machine learning through identification of SNPs, disease risk assessment, and non-linear SNPs detection. The authors of [12] likewise conducted a survey of machine learning approaches, and highlighted a wide range of models. The authors also noted the performances of each work, along with the advantages and disadvantages of the given model.

The other main focus area of runtime improvement is through parallelization. The different authors of [4-9] discuss various techniques for this application to GWAS. Parallelization was performed in both task-parallelism and data-parallelism, where the distribution of work was attempted in multiple ways. The approaches mainly employ OpenMP for shared memory systems, and some works employed MPI to distribute the work across compute nodes. Some of these works incorporated a parallel FHE along with a parallelized computation technique, achieving the benefit of a faster runtime along with a privacy-preserving solution.

Methodology

The work performed by the authors of [4] was used as the foundation for this work's implementation. The original authors made use of PALISADE, an open-source lattice cryptography library. The original authors also contributed by implementing a Residue Number System variant of the Cheon–Kim–Kim–Song (CKKS) HE scheme, which is documented and pushed to the PALISADE source code. For conducting GWAS, the original authors used two approaches: a logistic regression approximation, and a chi-square test. They then employed OpenMP in two areas: the first is for parallelizing the encryption of the GWAS data, and the second for the computation on the encrypted data.

The approach of this work leverages the PALISADE library as well as the RNS variant implemented by the author of [4]. The implementation of OpenMP by the original authors was also utilized. A flow diagram of this work's approach can be seen below. Later discussion will provide insight on the flow diagram, but it is of note that the initialization, sample max computations, FHE initial computations, and the distribution of work itself is highly negligible compared to the later parts of this diagram.

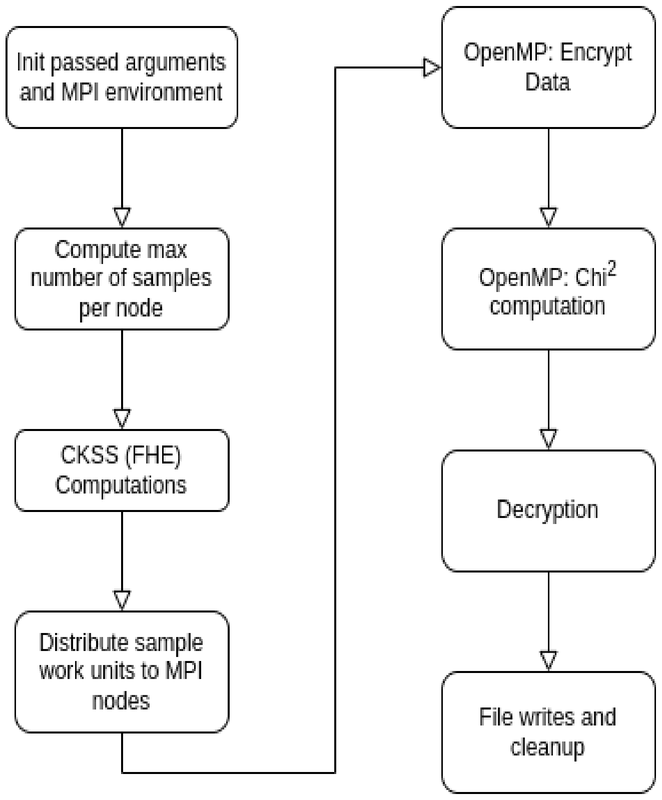
Init passed arguments and MPI environment
Compute max number of samples per node
CKSS (FHE) Computations
Distribute sample work units to MPI nodes
OpenMP: Encrypt Data
OpenMP: Chi^2 computation
Decryption
File writes and cleanup

Figure 1: Program Flow Diagram

For distributing to compute nodes, all nodes, including the root node, were assigned a set of data to work with in a data-parallelism approach. Each node completed the same tasks in the same manner. Each node generated a set of keys, and each node would read its set of assigned data from the GWAS file. However, due to the size of the data, the nodes were unable to read in all data due to memory capacity constraints. As a result, each node reads in a portion of the file proportional to its memory capacity. To this end, each node would read in the first line of the data file and compare its memory consumption to the total memory capacity of the node, accounting for the multi-core nature of the nodes. Each node would then read in batches of the file until it completed its portion of the computation. The parallelization approach can be seen below.

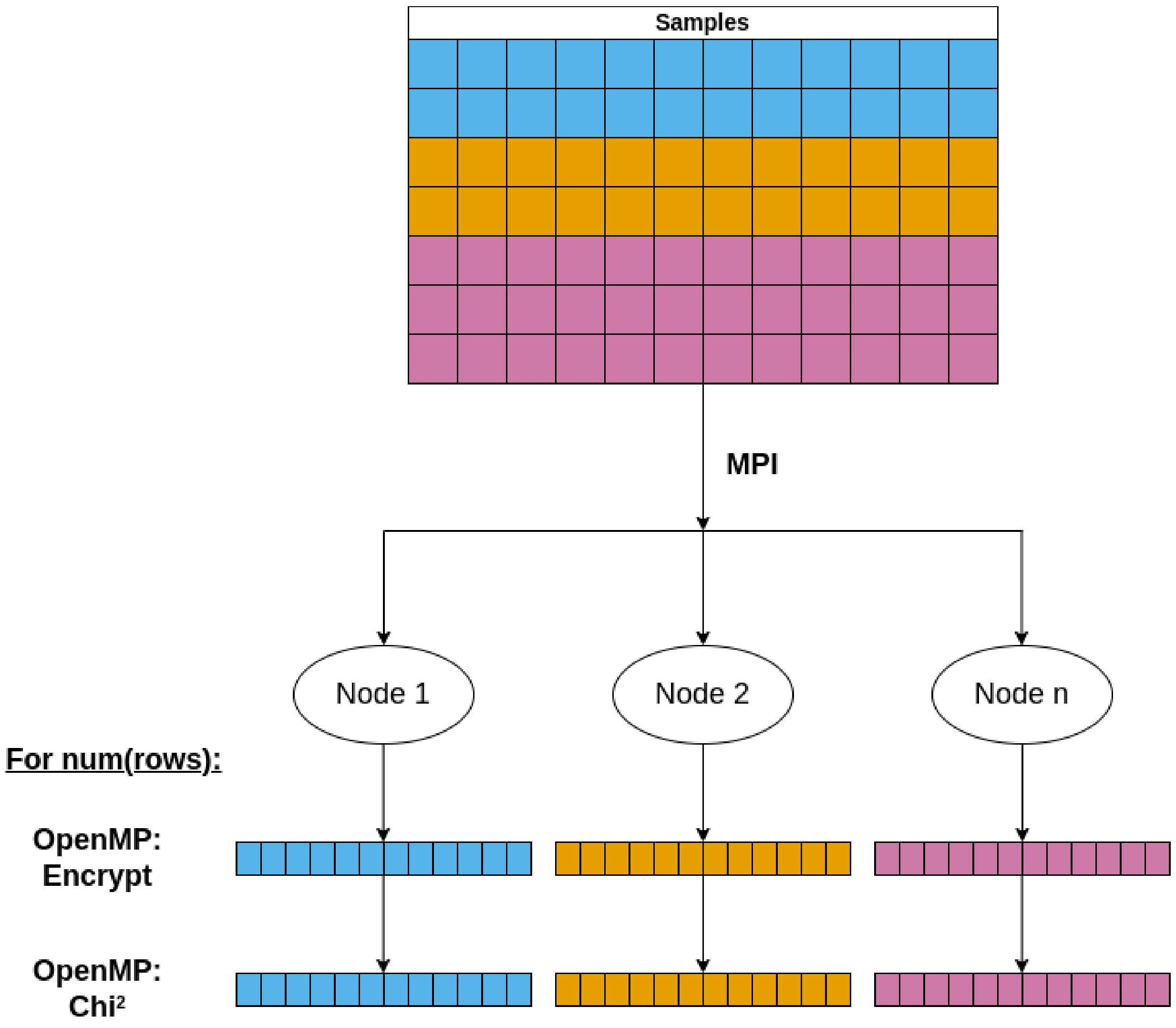

Figure 2: Parallelization Diagram

After computation and decryption, each node reported its runtime of each task to the root node. In addition, each node also wrote their results to a file in the output directory. It is of note that this work did not combine the results – they were left independent and separate. It is expanded upon in the Future Works section, but it is intended to conduct meta-analysis to combine the resulting p-values.

The data used for this work was artificial GWAS data generated in R. There were a total of 500,000 samples, with 10,000 total variables. Each node was responsible for an equal share of this data. The testbed used for this work is the University of Tulsa's HAMM3R compute cluster. The cluster is a 13 node cluster, with 12 nodes serving as dedicated compute nodes, and 1 node serving as the login node. Each compute node has a CentOS release 6.9 OS, two 8-core Intel Xeon E5-2620 v3s with hyperthreading totaling to 2 threads per core, two Intel Xeon Phi Co-Processors, one FPGA Nallatech PCIe-385n A7 Altera Stratix V, and 64GB of RAM. Each node is connected with a 10Gbps Infiniband interconnect.

Results and Analysis

Results were captured and analyzed with a few considerations and with general disclaimers. The first is that, as previously mentioned, the results of each node's GWAS computation were not combined, and were left independent. For full accuracy and better comparison of the original work by the authors of [4], these results should be combined and the time taken to combine results should be included in the total runtime. The second disclaimer is that no base serial benchmarking without parallelization through OpenMP was conducted. The base of these results is compared with a single compute node with OpenMP enabled. Measuring the total speedup when OpenMP and MPI are used would allow for a larger-scale comparison when no parallelization is used, as will be discussed in the Future Works section.

The raw timing data can be seen in the table below.

Nodes	Key Gen Time	Encoding and Encryption Time	Computation Time	Decryption and Decoding Time	File IO Time	Total Runtime
1.000	0.055	42100.000	4231.690	20.536	640.881	47003.700
2.000	0.055	21269.150	2239.320	11.081	311.926	23836.900
3.000	0.047	1813.133	744.020	4.823	205.425	2771.090
4.000	0.048	1364.435	584.453	3.662	161.781	2117.155
5.000	0.047	1090.648	463.536	2.927	130.895	1690.190
6.000	0.047	910.367	382.350	2.504	107.299	1404.363
7.000	0.050	781.140	324.703	2.114	94.767	1204.246
8.000	0.041	598.455	256.315	1.615	77.110	934.705
9.000	0.047	606.837	247.711	1.676	73.261	930.740
10.000	0.047	545.668	229.495	1.440	97.578	875.287

Table 2: Raw Timing Data in Seconds

This table can be seen in figure format below, along with its corresponding trendline and R^2 value.

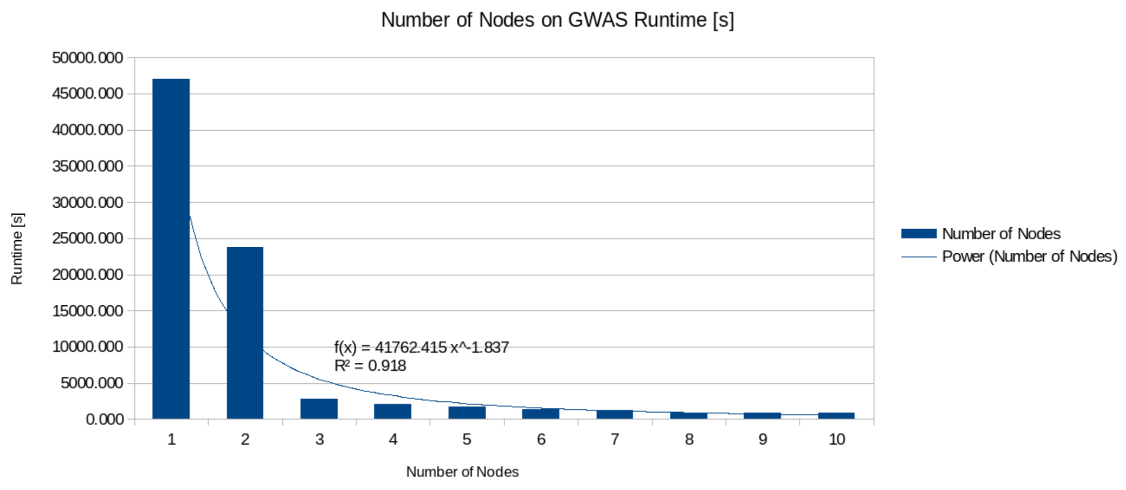

Figure 3: Number of Nodes on GWAS Runtime in seconds

A better metric for analyzing performance is through speedup. This work used speedup according to Amdahl, which can be found through:

$$S_{latency}(s)=\frac{1}{(1-p)+\frac{p}{s}} \qquad (1)$$

Due to the low overall time contribution of the serial components, there is a negligible difference with the inclusion of the serial portion. The speedup Figure can be seen below, along with its trendline equation and R^2 value. The speedup obtained is logarithmic, and while using nodes 3 through 7 fell under the logarithmic curve, nodes 8, 9, and 10 were above the curve. Though the addition of nodes is unlikely to change the trendline function, the Future Works section discusses that including more nodes could result in a better-fitting trendline. This speedup is notable in that these values were obtained despite serializing the data, and not reading the entirety of the data into memory.

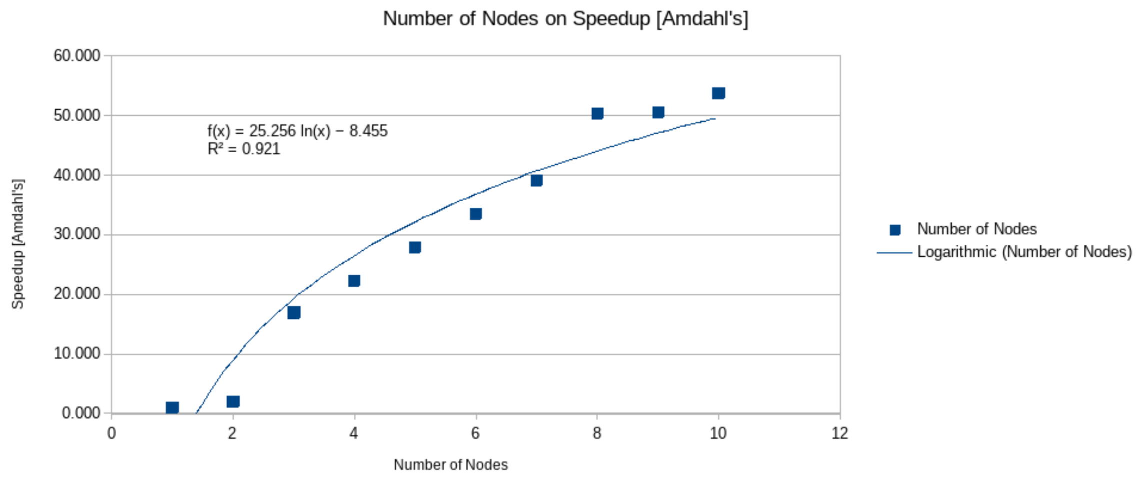

Figure 4: Number of Nodes on Speedup

The timing of each task was also captured. The main tasks included the time taken to generate the cryptographic key pairs, the time taken for the encoding and encryption of the GWAS data, the time taken to compute the chi^2 results using the homomorphic library computations, the time taken to decrypt and decode the encrypted data, and the time taken to read from the raw data as well as to write the results. The timing data was captured across all node tests, and a percentage was obtained as a proportion of the total runtime. These percentages were then averaged across all node tests to obtain a general average task percentage. The results can be seen in Figure 5. It is evident in the Figure that the time taken to generate the key pairs and the time taken to decode and decrypt was substantially more irrelevant than the encryption and computation time. Later discussion can be seen in the Future Works section, but the comparison of computation time to the encoding and encryption time is of note. Though the key generation time was negligible, the encryption of the data itself was the largest factor for the overall runtime, despite being parallelized with both OpenMP and MPI. Even though the usage of OpenMP and MPI does result in a maximum speedup for just the encryption portion of 77.15x, it still contributes to a significant portion of the runtime.

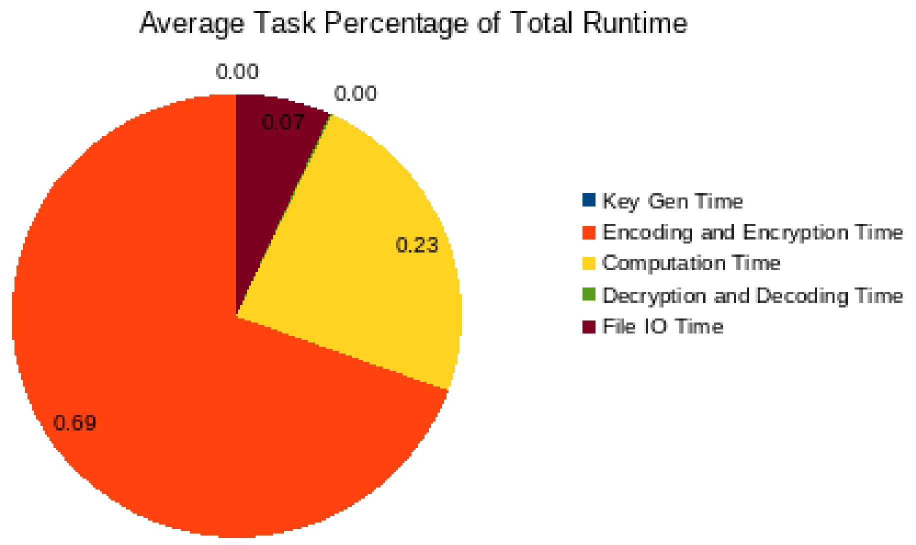

Figure 5: Average Task Percentage

Future Work

Various avenues are available to expand upon this work. Notably, this work used a single set of artificial GWAS data composed of 500,000 samples and 10,000 variables. For a larger-scale study, the data size could be expanded to include more samples, and could alter the number of variables. Expanding the size of the data could also show how File IO time would change. In addition to expanding the size of the data, numerous data size sets could be used. Starting with a low set of samples, and increasing to a large amount of samples and repeating for the number of variables allows for the efficiency of parallelization to be measured. By varying the problem size and obtaining speedup values for each test, conclusive results can be drawn regarding the efficiency of parallelization for this problem. Likewise, scalability results can be measured, indicating whether the parallelization of this problem is weakly or strongly scalable.

For better real-world applicability, using actual GWAS data is preferable to the artificial data. Real data also introduces other challenges and quirks that can't be fully captured with artificial data. In addition, the individual GWAS results computed by each node should be combined to single, conclusive p-values and odds-ratios. Leaving each result separate does not allow for a true comparison of this work to that seen in the work of [4]. A few approaches can be taken to combine the results, with meta-analysis presenting itself as the likely option. Since each result is from the same set of data, it is possible to use meta-analysis without great concern of inaccuracy. PLINK offers meta-analysis computations, and C++ bindings exist for its libraries. It is possible to include the meta-analysis into the program itself, and avoid the need for separate post-processing. To further extend real-world applicability, rather than reading in CSV files, this work could add functionality to read PLINK files using the aforementioned C++ bindings for the PLINK libraries.

Other avenues for future work can examine alternative cryptographic libraries. SEAL, Helib, and FHE all present themselves as viable options. In addition, if PALISADE were to be used in future work, different variants could be used. This work used the RNS variant presented by the authors of [4], though many other options could be used instead. Using other libraries and/or variants could examine the effects on encryption time with another lens for encryption security.

Conclusion

This work combined OpenMP, MPI, and homomorphic encryption with the PALISADE lattice cryptography library to implement a privacy-preserving, parallelized GWAS. This work achieved logarithmic speedup with respect to the number of MPI compute nodes despite serializing the input data rather than reading it entirely into memory. The speedup obtained through this approach reached upwards of 53.7x with 10 compute nodes compared to the OpenMP-only implementation, with optimism of higher speedups when increasing the memory capacity of nodes and when increasing the number of compute nodes.

References

[1] D. Fabregat-Traver and P. Bientinesi, “Computing Petaflops over Terabytes of Data: The Case of Genome-Wide Association Studies,” ACM Trans. Math. Softw., vol. 40, no. 4, p. 27:1-27:22, Jul. 2014, doi: 10.1145/2560421.
[2] M. M. A. Aziz et al., “Privacy-preserving techniques of genomic data-a survey,” Brief Bioinform, vol. 20, no. 3, pp. 887–895, May 2019, doi: 10.1093/bib/bbx139.
[3] A. B and S. S, “A survey on genomic data by privacy-preserving techniques perspective,” Comput Biol Chem, vol. 93, p. 107538, Aug. 2021, doi: 10.1016/j.compbiolchem.2021.107538.
[4] M. Blatt, A. Gusev, Y. Polyakov, and S. Goldwasser, “Secure large-scale genome-wide association studies using homomorphic encryption,” Proceedings of the National Academy of Sciences, vol. 117, no. 21, pp. 11608–11613, May 2020, doi: 10.1073/pnas.1918257117.
[5] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan, “Optimized homomorphic encryption solution for secure genome-wide association studies,” BMC Med Genomics, vol. 13, no. Suppl 7, p. 83, Jul. 2020, doi: 10.1186/s12920-020-0719-9.
[6] S. D. Constable, Y. Tang, S. Wang, X. Jiang, and S. Chapin, “Privacy-preserving GWAS analysis on federated genomic datasets,” BMC Med Inform Decis Mak, vol. 15 Suppl 5, p. S2, 2015, doi: 10.1186/1472-6947-15-S5-S2.
[7] D. O. Enoma, J. Bishung, T. Abiodun, O. Ogunlana, and V. C. Osamor, “Machine learning approaches to genome-wide association studies,” Journal of King Saud University - Science, vol. 34, no. 4, p. 101847, Jun. 2022, doi: 10.1016/j.jksus.2022.101847.
[8] K. L. Keys, G. K. Chen, and K. Lange, “Iterative hard thresholding for model selection in genome-wide association studies,” Genet Epidemiol, vol. 41, no. 8, pp. 756–768, Dec. 2017, doi: 10.1002/gepi.22068.
[9] M. Kim, Y. Song, B. Li, and D. Micciancio, “Semi-Parallel logistic regression for GWAS on encrypted data,” BMC Med Genomics, vol. 13, no. Suppl 7, p. 99, Jul. 2020, doi: 10.1186/s12920-020-0724-z.
[10] W.-J. Lu, Y. Yamada, and J. Sakuma, “Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption,” BMC Med Inform Decis Mak, vol. 15 Suppl 5, p. S1, 2015, doi: 10.1186/1472-6947-15-S5-S1.
[11] S. Meftah, B. H. M. Tan, K. M. M. Aung, L. Yuxiao, L. Jie, and B. Veeravalli, “Towards high performance homomorphic encryption for inference tasks on CPU: An MPI approach,” Future Gener. Comput. Syst., vol. 134, no. C, pp. 13–21, Sep. 2022, doi: 10.1016/j.future.2022.03.033.
[12] H. L. Nicholls, C. R. John, D. S. Watson, P. B. Munroe, M. R. Barnes, and C. P. Cabrera, “Reaching the End-Game for GWAS: Machine Learning Approaches for the Prioritization of Complex Disease Loci,” Front Genet, vol. 11, p. 350, 2020, doi: 10.3389/fgene.2020.00350.
[13] H. Paik, Y. Cho, S. B. Cho, and O.-K. Kwon, “MPI-GWAS: a supercomputing-aided permutation approach for genome-wide association studies,” Genomics Inform, vol. 20, no. 1, p. e14, Mar. 2022, doi: 10.5808/gi.22001.
[14] A. Petrini et al., “parSMURF, a high-performance computing tool for the genome-wide detection of pathogenic variants,” Gigascience, vol. 9, no. 5, p. giaa052, May 2020, doi: 10.1093/gigascience/giaa052.
[15] Y. Sei and A. Ohsuga, “Privacy-preserving chi-squared test of independence for small samples,” BioData Mining, vol. 14, no. 1, p. 6, Jan. 2021, doi: 10.1186/s13040-021-00238-x.
[16] J. J. Sim, F. M. Chan, S. Chen, B. H. Meng Tan, and K. M. Mi Aung, “Achieving GWAS with homomorphic encryption,” BMC Med Genomics, vol. 13, no. Suppl 7, p. 90, Jul. 2020, doi: 10.1186/s12920-020-0717-y.
[17] X. Wu et al., “A novel privacy-preserving federated genome-wide association study framework and its application in identifying potential risk variants in ankylosing spondylitis,” Brief Bioinform, vol. 22, no. 3, p. bbaa090, May 2021, doi: 10.1093/bib/bbaa090.
[18] M. Yang et al., “TrustGWAS: A full-process workflow for encrypted GWAS using multi-key homomorphic encryption and pseudorandom number perturbation,” Cell Syst, vol. 13, no. 9, pp. 752-767.e6, Sep. 2022, doi: 10.1016/j.cels.2022.08.001.