Utilizing High Level Synthesis to Accelerate Pattern Detection for Positive Selection based on Linkage Disequilibrium

Hidde Hoorweg
University of Twente
Enschede, Netherlands
h.j.hoorweg@student.utwente.nl

***Abstract*—Positive selection is an interesting concept in population genetics, and with more genetic data available new and better tools to detect selective sweeps indicative of positive selection are needed. Selective sweeps are detectable by looking for linkage disequilibrium (LD) and other statistical metrics. RAiSD uses LD and other statistical measures to detect a selective sweep. However with LD it only considers it a LD match when the r^2 value is 1, and all other values are floored to 0. This raises the question whether the performance of RAiSD could be improved by choosing the r^2 threshold somewhere else, for example 0.8. This is implemented, but the increase in accuracy of the code is not worth the increase in computational time with the tested data sets. This could be investigated further, but a decrease in compute time is required for this. So an FPGA accelerator is developed to try to decrease the computation time and realise the potential of the system.**

***Index Terms*—High level synthesis, Linkage Disequilibrium, accelerator, Selective sweep**

I. Introduction

Positive selection is a phenomenon which occurs when a certain allele has a higher probability of producing offspring compared to competing alleles. Over time this allele tends to be more prominent within the population. If given enough time the whole population will tend to get this allele. This increase in frequency of the favourable allele is called a selective sweep. A selective sweep is thus an indication that a certain allele has better change of producing more offspring. This is a useful indication for genetic research and population genetics. Thus being able to see a selective sweep and by extension positive selection is essential. The challenge is to detect the difference between neutral alleles and gene locations where recently positive selection has occurred. Figure 1 shows some of the fingerprints of a selective sweep, namely the fixation of the beneficial mutation, the reduction of variance close to the beneficial allele and strong correlations between neutral alleles [2].

When looking at the genetic data of a population, there is a lot of genetic data which is not interesting for the purpose of detecting a recent selective sweep. This is because most of the genetic data is equal for everyone inside the population. Only the locations where there is a difference between specimen is of interest when detecting a selective sweep. Because if all specimen have the same genetic data at a location, no correlation or other data can be extracted from this. This is achieved by only looking at the single-nucleotide polymorphisms (SNP) of the population, which are the different nucleotides at the same location for different specimen. Another possible reduction in datasize is to convert the base pairs (A, C, G, T), to a binary presentation, because it is assumed to only have two possible base pairs at one location. This allows a computer to process the data faster, because a binary representation of the different SNP's.

RAiSD [3] is a program for detecting selective sweeps. This program considers three different factors to detect positive selection, namely the variance, the site frequency spectrum (SFS) and the linkage disequilibrium (LD). The source code of this program is public and open.

High level synthesis (HLS) tools are tools which allows the developer to work at a higher level of abstraction on the FPGA. This allows the development time to decrease, and the tools tend to be more flexible. Currently there are some accelerators developed for the detection of selective sweeps [5][6]. However to the knowledge of the author none of these accelerators use a HLS for development.

The motivation for developing a more accurate implementation of the LD algorithm and to develop a accelerator for this is that it could have a potential to speed up the process of genetics research. It also tests the possibility of HLS tools to develop these accelerators, and this has the possibility to reduce the development of other accelerators. This would allow to use more datasets and see if this shows better results.

II. Background

Linkage Disequilibrium (LD) is the correlation between two different alleles in a population. Two different loci are in linkage disequilibrium when the frequency of association is either higher or lower than what is expected with random change. When considering two different loci i, j in a given populations, consider an allele A at locus i, and an allele B at locus j. p_A is the frequency of allele A occuring in the population, p_B is similarly the frequency for allele B and lastly p_{AB} is the frequency of both A and B occurring at the same time. Then the coefficient of linkage disequilibrium is given by

$$D_{AB} = p_{AB} - p_A p_B \tag{1}$$

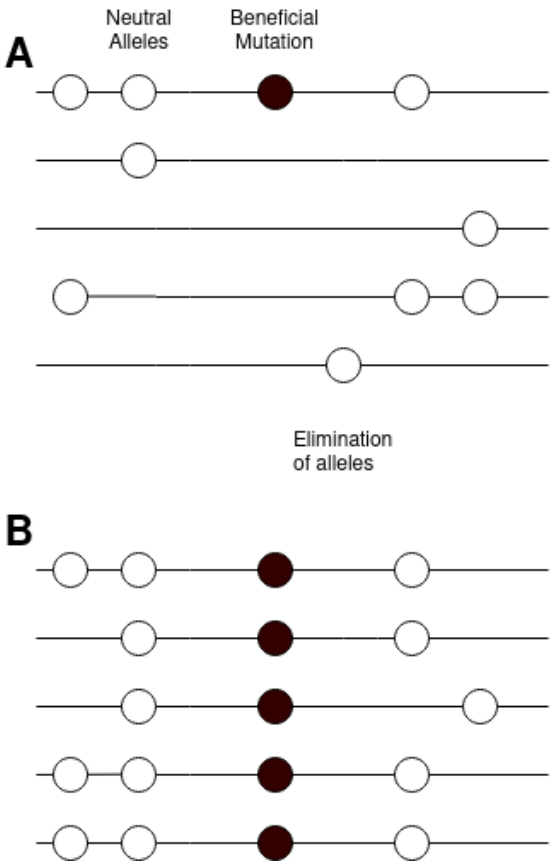

Fig. 1. A: The first specimen in the population has the beneficial allele
B: After the selective sweep has finished and the population has the same beneficial allele

This disequilibrium constant can be different for different combinations of alleles with fixed loci. However when you simplify the situation that there are only 4 alleles (a, A, b, B), two at each location the values of LD are related a following:

$$D_{AB} = -D_{Ab} = -D_{aB} = D_{ab} \tag{2}$$

The sign of the constant isn't of interest with regards of detecting a recent selective sweep. This is due to the fact that the sign of the coefficient is a indication if the two alleles are more present together than expected from random change, and a negative sign indicated that the alleles tend to occur separate from each other. The absolute value is of more interest, because a higher absolute value indicates a higher degree of linkage disequilibrium. One problem with this definition is that it is very closely related to the frequencies of the two different alleles, and thus it is hard to compare the linkage disequilibrium between different loci of the same group, due to the fact that the frequency of the original alleles is different, the value of linkage disequilibrium also changes. To try to mitigate this problem the correlation coefficient is used to normalize the values for LD, meaning that the values can be compared in a meaningful way.

$$r^2_{A,B} = \frac{(P_{A,B} - P_A P_B)^2}{P_A P_B (1 - P_A)(1 - P_B)} \tag{3}$$

This formulation of the Linkage Disequilibrium also has the added benefit of always being positive, meaning that comparing values is easier.

III. Related Work

A. Positive selection detection

There are currently many competing tools for analysing genetic data to detect positive selection [15]. Pavlidis et al [15] compile some of the more used tools, and try to compare the differences between them. There are three basic mathematical tools which are used to detect an occurrence of recent positive selection, namely site frequency spectrum (SFS), linkage disequilibrium (LD), and diversity reduction. Pavlidis et al [15] evaluated four different tools, SweepFinder [14], SweepFinder2 [9], SweeD [16] and OmegaPlus [1] [4], which consist of both sequential and parallel implementations, and utilise either SFS of LD.

Positive selection can be measured using multiple factors, the main three are the local genetic variation, SFS and LD [3]. As shown by Alachiotis and Pavlidis [3], combining these factors yields superior results, and with some optimization the computational load is also reduced compared to other state of the art tools. The LD part of this code will be changes to test the potential of testing the LD value to a different threshold. There is the tool based on the μ statistic [3], which accelerates this tool by utilizing a FPGA to decrease processing time of some intensive calculations. This tool is developed by Alachiotis et al [5], and this implementation was able to decrease the processing time by up to two orders of magnitude on both simulated and real data. It achieves this by splitting up the process by a CPU which parses the genetic data into a set of SNPs, and a FPGA which calculates the variation, SFS and LD, using a sliding window.

B. HLS and FPGA acceleration

Recently the field of HLS is developing quickly with both academic and commercial HLS languages being developed [12].

Nane et al [12] made a survey on the different HLS, and tried to differentiate between the different tools. They classified the languages into four classes, namely Object Oriented, Procedural, C-extended and New languages. It also compared the performance of three academic languages, BAMBU [18], DWARV [13] and LEGUP [7], and compared it to a commercial language.

Chrysos et al [8] studies the possibilities of FPGA's to accelerate different tools. For this the authors looked at multiple algorithms, and stated some algorithms which have a FPGA implementation. Here it is observed that the computation time can be reduced, while significantly reducing power consumption of the system. There is also the potential of Graphical processing

units (GPU), which are not configurable hardware, but GPU's excel at parallel processing, able to increase performance up to 40 times compared to classical CPU processing [8]. This also introduces the concept of hybrid systems, with both a FPGA and a GPU to choose the best accelerator depending on the algorithm, or dividing the algorithm up into a part GPU part FPGA.

Alachiotis et al [6] implemented a FPGA based LD accelerator. This accelerator utilises an automatic RTL generator to fully utilise the FPGA. This implementation was up to 50 times faster than a modern 6-core CPU. This accelerator thus reduces the compute time, but it also considers LD in a binary fashion, meaning that all r^2 not equal to 1 are considered to be 0, reducing compute time more, but also reducing the output resolution, and thus potentially losing some information.

IV. System design

A. Extending RAiSD

The program RAiSD [3] is used as the original source to test the behaviour of the different ranges of LD. This is achieved by replacing the functions which return if the two inputs are a LD match. These functions take as input two different arrays of SNP's, in a binary format. It also has as input the size of this array and the amount of SNP's to consider. One considers the situation that if allele A is present, B is also present, effectively looking at if the arrays A and B are identical. The other functio flips one of the arrays and then checks if the two arrays are then equal. This checks the situation if A is present, B is not and vise versa. So these two functions are able to detect LD if the r^2 value is equal to 1, but isn't sensitive for any other r^2 value.

To change the behaviour of the code, another function is called as opposed to the two described above, which returns a float representing the value of r^2, allowing the code to more precisely determine an appropriate threshold as opposed to only considering $r^2 = 1$. This new function also doesn't need to consider the binary opposite, due to the fact that the calculation of r^2 doesn't depend on the definition of which alleles are 1 and 0, as seen in equation 3.

This new version of the code is tested against four different data sets, and the μ_{stat} is compared between different runs with different LD thresholds and different sets.

B. LD calculation

The LD function which was used above to test the functionality of changing the LD threshold is now used as a basis for the accelerator. A copy is made as a reference function to make sure that the behaviour of the accelerator and the original function remains identical. The definition of r^2 as seen in equation 3 uses a lot of numbers with a range of $[0, 1]$, which has to be implemented by either a float or a double. This takes a lot more resources on a FPGA, so equation 3 is rewritten in equation 4.

$$D_{i,j} = P_{i,j} - P_i P_j$$

$$P_i = \frac{popcnt(i)}{S}, P_j = \frac{popcnt(j)}{S}, P_{i,j} = \frac{popcnt(i\&j)}{S}$$

$$r^2_{i,j} = \frac{(P_{i,j} - P_i P_j)^2}{P_i P_j (1 - P_i)(1 - P_j)} = \frac{\left(\frac{popcnt(i\&j)}{S} - \frac{popcnt(i)}{S}\frac{popcnt(j)}{S}\right)^2}{\frac{popcnt(i)}{S}\frac{popcnt(j)}{S}(\frac{S}{S} - \frac{popcnt(i)}{S})(\frac{S}{S} - \frac{popcnt(j)}{S})}\frac{S^4}{S^4} = \frac{(popcnt(i\&j)S - popcnt(i)popcnt(j))^2}{popcnt(i)popcnt(j)(S - popcnt(i))(S - popcnt(j))} \quad (4)$$

In equation 4 is used to represent the bitwise AND operation between two words, and the function $popcnt$ is used to represent the popcount function, which is the amount of 1 in a integer. The advantage of this rewriting is that most of the calculations are done using integers. Only at the last step, with the division do the integers have to be converted to floats and only 1 float operation has to be performed.

The design of the accelerator can be seen in Figure 2, where the dataflow of the accelerator is shows with two different popcount cores. The pop count part of the hardware is used to determine the values of $popcnt(i), popcnt(j)$ and $popcnt(i \cap j)$ from equation 4. The big challenge of determining these values is that it is dependant on the sample size, and the fact that a long integer only contains 64 bits. This means that the hardware has to determine the pop count of possibly more than 1 long integer. This calls for a iterative approach, but a iterative for loop with a variable amount of cycles is difficult to synthesize to hardware. This problem was solved by implementing a different amount of popcount cores, which work in parallel on different parts of the data, and at the end the sum of the output of these cores is taken and used to determine the value of r^2.

The second part of the processing is not influenced by the input data size, but the arithmetic uses a lot of floating point hardware, which is slower than integer arithmetic [10]. To take advantage of this fact, the calculation for r^2 is changed slightly to allow more integer arithmetic to be used, see equation 4.

V. Results

A. Experimental setup

The data sets which are used are a subset used to test the original code for RAiSD [3] [11]. The data sets 1, 20, 40 and 60 are used from RAiSD. The metrics used to test the output are the accuracy, success rate, true positive rate (TPR) and run-time. The computer which was used to run RAiSD and get the run-time values contains a AMD Ryzen 7 3700X paired with 16 GB of RAM. The FPGA board which was used for the accelerator is the ZedBoard (xc7z020-clg484-1) [19]. This FPGA was configured to run with a cycle time of 10 ns.

B. LD with accuracy

In Figure 3 the accuracy of RAiSD is plotted against different r^2 threshold, with multiple data sets. It can be shown that the a reduction of r^2 threshold doesn't change the accuracy significantly until the threshold is reduced to below 0.2.

In Figure 4 the success rate of the different runs of RAiSD are plotted. Again there is not a significant difference between the

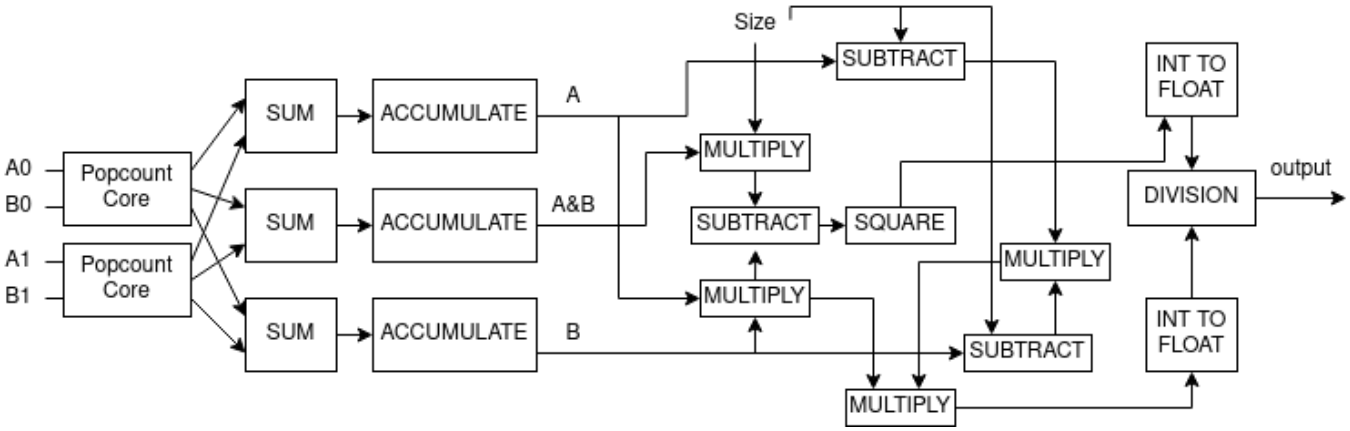

Fig. 2. Design of the FPGA accelerator, with two popcount cores

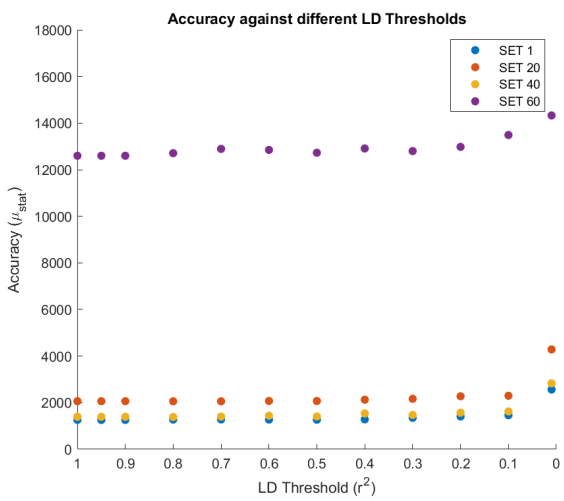

Fig. 3. Accuracy of RAiSD with different threshold of LD which are considered a match

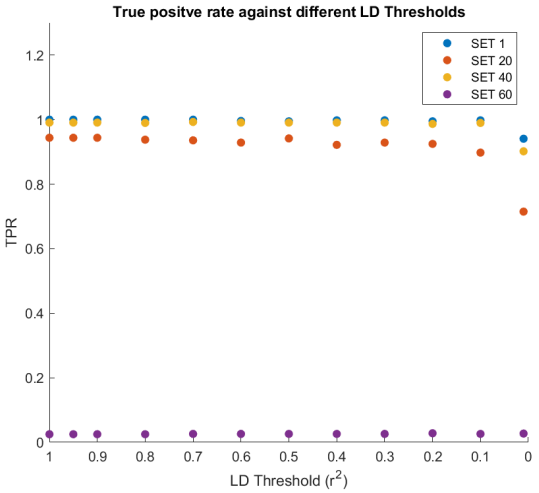

Fig. 5. The TPR of RAiSD with different data sets and different thresholds

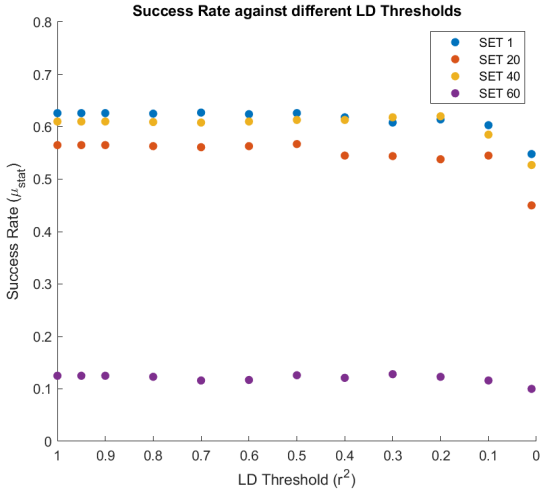

Fig. 4. Success rate of RAiSD with different thresholds of LD

runs with a threshold between 0.5 and 1.0. However below 0.5 The success rate starts to decrease below 0.5 and this reduction accelerates as the threshold is reduces to below 0.2.

The True Positive Rate (TPR) is perhaps one of the most important metrics to consider, because it considers how sensitive RAiSD is to a selective sweep which is the goal of the software. It can be seen in Figure 5 that for high LD threshold the results are highest, and that the TPR tends to reduce with lower LD threshold. However these differences are not large until the LD threshold reduces to below 0.2, an then the TPR starts to plummet, making the output significantly worse.

Lastly the time to compute the different samples is compared. The computational increases compared to the original RAiSD code due to the fact that the code is calculating the actual r^2 value as opposed to comparing if two samples are equal or exact opposite. This causes the code slow down as seen in Figure 6. However with a lower LD threshold, there are more matching samples, and due to the fact that these samples are combined, the code performs fewer comparisons. This causes

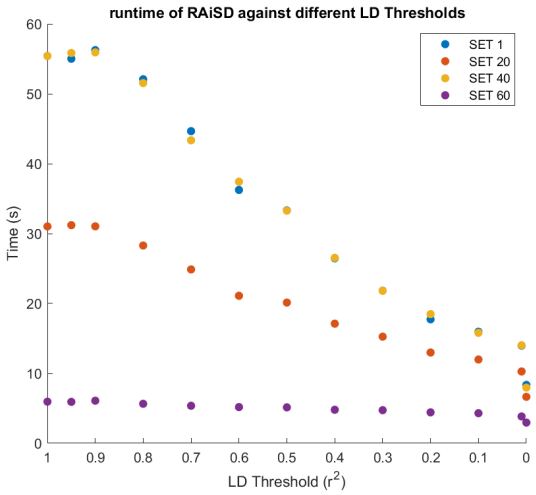

Fig. 6. Genetics variation at A: A new occurrence of a beneficial mutation B: After the selective sweep has occurred

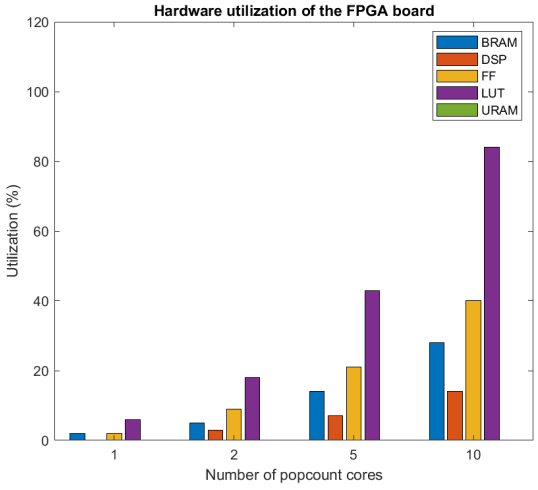

Fig. 7. Hardware requirements of the accelerator with different amount of popcount cores

the code to decrease computation time with a reduced LD threshold. Initially the change in computational time is around a factor of 2, and the original code is as fast as a LD threshold between 0.3 and 0.4.

C. Accelerator performance

The design is tested with a input size of 1000 long integers for every input cycle, which has to be reduced to a single float as a output. The accelerator requires more hardware the more popcount cores are used, which is expected (see Figure 7). When 10 pop count cores are used, 84% of the LUT's are used, as seen in Table II. A problem with the 10 core design is that it requires a input bitrate of 53 Gbit/s (see Table I). But the FPGA used to design the hardware only has a PCIe 2.0x8 interface [19], which only is able to achieve a bitrate of 32 Gbit/s [17]. Thus of the different designs, the 5 popcount design has the highest throughput while the input fits on a 32 Gbit/s PCIe 2.0x8 interface. But the required throughput of the 5 core design is fairly close to the max theoretical throughput, so any design changes which require more throughput will over saturate the PCIe interface.
The 5 core design, with a input sample size of 1000 64 bit words has a interval of 219 cycles I.

VI. Conclusion

One of the most important metrics of RAiSD is the TPR. To get at least comparable results to the original code a high LD threshold has to be chosen (Figure 5). The same applies to the success rate of RAiSD (Figure 4). However these results are comparable to the original code and the runtime is significantly higher to the original code (Figure 6). Combining these facts it is currently not worth the additional computational load on a computer to calculate the LD as opposed to checking if it is equal or opposite.
The FPGA accelerator has potential to increase the performance of the system, but the accelerator hasn't been tested in combination with RAiSD to determine the difference in computation time. So the full potential of the accelerator is not known, but the fact that it is able to fully saturate the PCIe 2.0x8 shows that the throughput of the system is fairly high.

VII. Discussion

RAiSD was designed with either a full LD match or no match at all. One way this is obvious is that the program will group matching pattern and consider it as a single pattern. This makes sense when a LD match means that it is totally equal. But when considering LD values which are not a full match, this optimization will throw away unique patterns. This in turn could reduce the performance of the system. Possibly there are other similar simplifications in the code which expect a LD pattern to be either a full match or no match at all, and when the code matches two non identical patterns it will actually reduce the performance. However these factors were not tested due to time constraints.
A obvious improvement is to integrate the accelerator into the RAiSD software and actually compare the different times required to complete the analysis of the data. This would show the potential time savings or maybe the fact that the latency associated with using a external PCIe device would be more significant than the reduced time to compute. Another thing with the current implementation of the accelerator is that the routing of the design has not been done, which could reveal errors or decrease the final performance of the accelerator.

References

[1] N. Alachiotis, A. Stamatakis, and P. Pavlidis. "OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets". In: *Bioinformatics* 28.17 (July 2012), pp. 2274–2275. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts419. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/17/

TABLE I
DIFFERENT INTERVAL AND CYCLES WITH A INPUT DATASIZE OF 1000

popcount cores	Throughput (MHz)	interval (cycles)	latency (cycles)	input bit rate (Gbit/s)
1	0.097	1036	1035	6.2
2	0.193	519	549	12.35
5	0.457	219	250	29.2
10	0.84	119	152	53

TABLE II
HARDWARE REQUIREMENT FOR DIFFERENT AMOUNT OF POPCOUNT CORES

1 popcnt (number)	1 popcnt(%)	2 popcnt (number)	2 popcnt (%)	5 popcnt (number)	5 popcnt (%)	10 popcnt (number)	10 popcnt (%)
8	2	16	5	40	14	80	28
0	0	8	3	17	7	32	14
3101	2	10011	9	22905	21	42980	40
3402	6	9761	18	23255	43	44953	84
0	0	0	0	0	0	0	0

2274/16904955/bts419.pdf. URL: https://doi.org/10.1093/bioinformatics/bts419.

[2] Nikolaos Alachiotis and Pavlos Pavlidis. "RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors". In: *Communications Biology* 1.1 (2018). DOI: 10.1038/s42003-018-0085-8.

[3] Nikolaos Alachiotis and Pavlos Pavlidis. "RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors". In: *Communications Biology* 1.1 (June 2018). DOI: 10.1038/s42003-018-0085-8. URL: https://doi.org/10.1038/s42003-018-0085-8.

[4] Nikolaos Alachiotis, Pavlos Pavlidis, and Alexandros Stamatakis. "Exploiting Multi-grain Parallelism for Efficient Selective Sweep Detection". In: *Algorithms and Architectures for Parallel Processing*. Springer Berlin Heidelberg, 2012, pp. 56–68. DOI: 10.1007/978-3-642-33078-0_5. URL: https://doi.org/10.1007/978-3-642-33078-0_5.

[5] Nikolaos Alachiotis and Gabriel Weisz. "High Performance Linkage Disequilibrium". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2016). DOI: 10.1145/2847263.2847271.

[6] Nikolaos Alachiotis et al. "Accelerated Inference of Positive Selection on Whole Genomes". In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)* (2018). DOI: 10.1109/fpl.2018.00041.

[7] Andrew Canis et al. "LegUp". In: *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '11*. ACM Press, 2011. DOI: 10.1145/1950413.1950423. URL: https://doi.org/10.1145/1950413.1950423.

[8] Grigorios Chrysos et al. "Reconfiguring the Bioinformatics Computational Spectrum: Challenges and Opportunities of FPGA-Based Bioinformatics Acceleration Platforms". In: *IEEE Design and Test* 31.1 (2014), pp. 62–73. DOI: 10.1109/mdat.2013.2284191.

[9] Michael DeGiorgio et al. *SWEEPFINDER2: Increased sensitivity, robustness, and flexibility*. 2015. arXiv: 1505.07142 [q-bio.PE].

[10] Don Lahiru Nirmal Hettiarachchi, Venkata Salini Priyamvada Davuluru, and Eric J. Balster. "Integer vs. Floating-Point Processing on Modern FPGA Technology". In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, Jan. 2020. DOI: 10.1109/ccwc47524.2020.9031118. URL: https://doi.org/10.1109/ccwc47524.2020.9031118.

[11] R. R. Hudson. "Generating samples under a Wright-Fisher neutral model of genetic variation". In: *Bioinformatics* 18.2 (Feb. 2002), pp. 337–338. DOI: 10.1093/bioinformatics/18.2.337. URL: https://doi.org/10.1093/bioinformatics/18.2.337.

[12] Razvan Nane et al. "A Survey and Evaluation of FPGA High-Level Synthesis Tools". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.10 (2016), pp. 1591–1604. DOI: 10.1109/tcad.2015.2513673.

[13] Razvan Nane et al. "High-Level Synthesis in the Delft Workbench Hardware/Software Co-design Tool-Chain". In: *2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE, Aug. 2014. DOI: 10.1109/euc.2014.28. URL: https://doi.org/10.1109/euc.2014.28.

[14] R. Nielsen. "Genomic scans for selective sweeps using SNP data". In: *Genome Research* 15.11 (Nov. 2005), pp. 1566–1575. DOI: 10.1101/gr.4252305. URL: https://doi.org/10.1101/gr.4252305.

[15] Pavlos Pavlidis and Nikolaos Alachiotis. "A survey of methods and tools to detect recent and strong positive selection". In: *Journal of Biological Research-Thessaloniki* 24.1 (2017). DOI: 10.1186/s40709-017-0064-0.

[16] Pavlos Pavlidis et al. "SweeD: Likelihood-Based Detection of Selective Sweeps in Thousands of Genomes". In: *Molecular Biology and Evolution* 30.9 (June 2013),

pp. 2224–2234. DOI: 10.1093/molbev/mst112. URL: https://doi.org/10.1093/molbev/mst112.

[17] *PCI Express®Base Specification*. v2.1. PCI-SIG. Apr. 2009.

[18] Christian Pilato and Fabrizio Ferrandi. "Bambu: A modular framework for the high level synthesis of memory-intensive applications". In: *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, Sept. 2013. DOI: 10.1109/fpl.2013.6645550. URL: https://doi.org/10.1109/fpl.2013.6645550.

[19] *Zynq-7000 SoC Data Sheet: Overview*. DS190. v1.11.1. Xilinx. July 2018.