CHAPTER 1

INTRODUCTION

Admixture Analysis is a tool to study the stratification of population. It is useful for genetic association studies [1] such as admixture mapping [2][3][4], subspecies classification [1] or genetic barrier detection [5]. The result of this analysis is the proportion of each ancestral genetic group. The aim of the analysis is to infer the mixing ratio of each unknown ancestral group from the known current genetic population data. Tang proposed a method for an individual admixture analysis [6]. This analysis can infer the admix ratio from one ancestry groups of each individual. The calculation can be a tool to infer the Hapmap tree [7][8]. Furthermore, this analysis was applied in some epidemiology genetic such as, obesity, breast cancer, the skin pigment of women who receive genetic from Hispanic and African-American [9] or diabetes of African-American [10]. However, the data to infer the stratification should be large to gain more accuracy. The large dataset takes very long time to process. Some dataset takes several months.

Admixture analysis is an ancestry estimation tool based on a maximum likelihood relation. Expectation Maximization (EM) [11][12][13] is a method to solve the likelihood relation. There are basic mathematic operators to be used in EM: random, addition, subtraction, division and multiplication. While other methods are based on numerical methods such as Newton-Raphson, NR, [14] or Block Relaxations, BR, 0. Most of NR and BR operators are matrix operators, the matrix operators were used to compute a Jacobian or Hessian matrix from the relation.

Field Programming Gate Arrays, FPGA [21], is a choice to speed up the calculation because the operators of FPGA work in fully parallel mode and they have low power consumption. Furthermore each unit of calculation of FPGA operates independently. The different calculation can be executed at the same time. Most other parallel processors are homogeneous. Currently, most of calculation speed can be gained by using parallel paradigm such as GPGPU [22][23] or Cloud Computing from Amazon [24]. Most of the techniques are done in the software level, so they cannot control each unit of calculation directly. Every

commands are passed to Operating System which in turn accesses the hardware level. This work presents a method to control each unit of calculation directly.

1.1 Related works

Related works in this dissertation were separated into two parts. The first part is the development of algorithms on the likelihood calculation. The second part is the development of calculation technologies.

1.1.1 Algorithm Development

Admixture analysis has been developed for a long time since 1986[3], starting from allele frequency comparison which is the first approach of this field. The work was based on prior knowledge techniques. Various types of genetics representation were applied with the comparison technique such as microsatellite or Short Tandem Repeats (STR) counting. STR is a repeat of genetic base. The length of the tandem is 2-6 bases. The number of repeats is not more than 100 repeats. The tandems are show in Figure 1

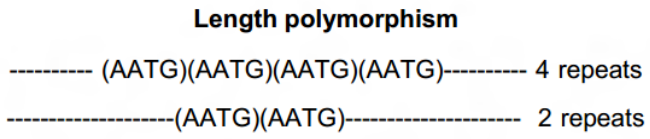

Figure 1 Tandem repeat

STR was also used in forensic work in the step of PCR. Inferring the admixture of each individual was based on the known tandem data. The tandem data should be a common sequence of the parental genome and current genome. Determining the admix ratio of each individual requires the knowledge of ancestry group marker. In 1991 Long [1] proposed the algorithm to analyze the admix ratio using F-statistic, F_{st},[25], and Chi-square statistic applied to candidate the markers to analyse the admixture of the SNPs data. The selected markers by Long require a genomic knowledge; the markers to be used by Long must be an ancestry informative marker (AIM). An AIM [26] requires many of biology knowledge and prior knowledge of the ancestry population genetic. The result that obtained from Long is a group level admixture ratio. The information of F_{st} represents an admix ratio in group level. F_{st} is a statistical tool to measure the correlation between groups of genetic data. F_{st} is based on a deviation of the allele

frequency represents in binomial distribution. In 2003, Carrie and et al. [26] analyzed the accuracy of the analysis of STR and SNP. The work stated that SNPs data is better than the STR data for the admixture calculation. In 2000, Pritchard [27] proposed the method of calculation by applying the Markov Chain to the admixture calculation. Pritchard et al. introduced the STRUCTURE version 1. The STRUCTURE version 1 is software to infer models from the admixture group of data. This group of genomic data was inherited from some group of ancestral group. The proposed method is a kind of statistical method that does not required ancestry genomic knowledge, such as AIM or genetic inheritance knowledge. In 2003 Falush et al. [29], proposed a STRUCTURE version 2. Multi loci of genetic data were applied with this version of STRUCTURE. The calculation was based on Linkage Disequilibrium and Allele frequency. The STRUCTURE software gains a lot of accuracy in the population level. The calculation can only be applied in the population level and it takes a lot of calculation time. Chikhi et al. proposed likelihood based calculation in 2000 [28]. Figure 2 shows the mechanic of the mixture model calculation of two groups of ancestry.

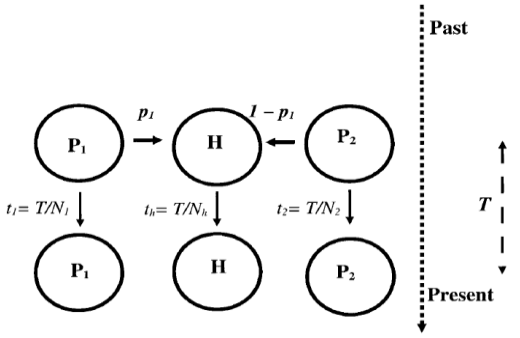

Figure 2 Mixing of two groups of ancestry when the time is changed

This method based on calculation of mixture models as Eq 1 that is based on Bayesian's network. The work proposed a speed up method for the calculation by Monte Carlo method. The work is an inference of genetic data in a population level but the genetic knowledge were applied in the calculation. The likelihood in this work is based on linear combination of each group of population. Three parameters were proposed in Eq 1, the first parameter is probability of ancestry group (p_i), the second parameter is admix ratio (x_i) and the last parameter is generation of mixing (t_j). The calculation is based on current data, D to infer p and x.

Eq 1. $$L(p_1, p_2, t_1, t_2, t_h, x_1, x_2|D)$$

In 2005, Tang [6] proposed a log-likelihood model for admixture calculation. The proposed method computes this model by Expectation Maximization, EM. FRAPPE is software by Tang to calculate the admixture ratio in Individual level. Tang proposed two methods in his work. The first method is EM and the second is Newton-Raphson (NR). EM was selected in his work because the implementation of NR is more difficult as the number of ancestral groups grows. Both of the approaches cannot guarantee the global maxima for non-convex relation. To solve any non-convex relationship the calculation must be re-run by changing the initial point. In 2009, Alexander et al 0 proposed a method of block relaxation for the Admixture by log-likelihood relation. ADMIXTURE was provided by Alexander. ADMIXTURE is software for admixture analysis. Quasi-Newton-Raphson (QNR) was applied to solve the likelihood relation. The calculation achieves the speed of calculation via QNR. The NR is a Quadratic time algorithm. Both of iterative methods cannot guarantee any of global convergence with non-convex problems. For NR and QNR, the starting point of calculation must be known. Most of NR guarantee a good convergence if the starting point is close to the answer.

Most of the works above are based on single-core machines and they are slow on a large dataset. Every proposed methods may get stuck at a local maximum. Using parallel programming can reduce the computation time but it is very hard to achieve good performance across variable size of dataset because there were too many choices to consider. There were two major division of work in parallel computing, first is communication and second is computation. Both of them should be balanced to gain most speed.

1.1.2 Technology of Calculation

Recently, there are a lot of computing technologies such as multi-core CPU, GPU (Graphics Processing Unit), FPGA. Moore's Law states that the number of transistors on a device doubles every two years because more transistors can be fabricated in a die. An improvement of clock frequency is another factor of performance gain. This dissertation proposed a calculation of Admixture by FPGA. Many works showed the performance of FPGA is better than the performance of GPU and Multi Cores, especially the floating point calculation. In [15], Stephanie and Jason proposed acceleration of phylogenetic tree calculation by FPGA. This work achieves 10x speed up relative to the calculation by software only. The computing tasks were divided into two tasks, the first task is the performance-critical task and the second

task is non-performance-critical. The first task was assigned to a FPGA and the second task was assigned to a general CPU. The non-performance-critical is data manipulation task such as data swapping, sampling or moving data. While the performance-critical is the task that requires fixed point and floating point calculation. Finding a phylogenetic tree is an application of this work. Bayes' Net was applied for the calculation. Markov Chain Monte Carlo (MCMC) was applied to speed up the calculation of the Bayes' Net. The problem is formulated as a state-space-search. MCMC guesses a reasonable direction of the search to the state-space-tree. The direction at each node was obtained from the calculation of every nodes under that node. FPGA was applied to calculation the reasonable direction of the current node then trace to the selected direction until reach the leaves node. Virtex-2 Pro 100 FPGA was used as the acceleration device. The device operates at 165 MHz, but it can be operated at 310 MHz using Virtex-6 SX 475 FPGA technology. Several of FPGA applications were applied with streaming data and signal processing such as [16]. Peter and Desmond proposed a technique of decoder calculation. The FPGA was used to process the Alamouti decoder [16] to increase system capacity and performance throughput of multiple input and output antennas (MIMO). There was a report to show the floating point operation performance of FPGA vs. CPUs in [17]. This work showed the trends of both single and double precision floating point performance of the FPGA above the CPUs. The benchmark in this work is LINPACK, the linear algebra of floating point calculation and matrix operations. In 2010, Christian et al did an application to compare the overall performance on FPGA Multi-Core CPUs and GPU [19]. The application is a real-time network intrusion detection system. ReMIND [20] is a detection tool in this work. The main concept of this work is matching subsequence in packet contents by efficient sorting. This work proposed and implemented a merge-sort, bitonic-sort and insertion-sort for all testing platforms. FPGA in this work is Virtex-5 with 0.55 GHz clock frequency, two Quad-core Xeon 5472 3GHz with OpenMP C/C++ and Nvidia Quadro FX 5600 1.35GHz of clock with CUDA. Multi-core CPUs gain highest performance in this work at 2Gbits per second, while the second rank is FPGA with 2Gbit per second of operation time but 1Gbit per second of data transferring and the bottom rank is GPU at 8Mbit per second. Moreover, the data format of the FPGA has more freedom than simple data format of general processors units [18]. Chunk of memory may be implemented for each calculation unit in the high density FPGA that can get rid of the bottleneck between processing unit and memory unit especially using multi processing units on a single

data. In a general purpose processing unit, there is only a single memory connection through memory. This architecture aimed to save space on chip and routing space on a circuit, but it increases the latency of memory accessing. The IEEE 754 floating point [45] was applied in this dissertation, because it is broadly used in all the current floating point calculation units such as AMD or Intel FPU [17]. EM NR and BR calculation are based on iteration method. The result is dependent on a good initial point and the number of iteration. The convergence of any method may requires more than ten thousand iterations of calculation. The huge number of iteration can cause a lack of precision from the cumulative error of fixed point representation.

1.2 Problem statement

Although Moore's law claimed the hardware resource scaled at 2x every year. But the demand of calculation is increased much more than that. Most of calculation devices have weakness because of lack of hardware, such as RAM is not enough or loss of precision in a low bit of fixed point number format. This dissertation proposed a systematic improvement of EM on FPGA that is scalable by the number of ancestor group. In admixture calculation, the number of individual and the number of marker were fixed but the number of ancestral group is unknown. This is a problem for both software application and hardware application to prepare a space of RAM in software application or the number of Register in hardware application for allocation the huge size of variables. The advantage of FPGA is a freedom of design such as scattering of memory into a small chunk of memory per calculation unit. Although the clock frequency of an FPGA is not competitive with CPU from Intel or AMD or GPU, but the FPGA can create a large amount of calculation unit to simultaneously operate. Scattering of memory can reduce the bottleneck of general processing unit and memory unit. It integrates the memory unit into the calculation unit, same as register of Intel 808x or local memory of GPGPU. The design these Scattering units are scalable by number of ancestry groups.

CHAPTER 2

METHODS AND BACKGROUND

2.1 Mixture models

Mixture model [30] is a probabilistic model derived from the existing and known data (Figure 3). There were four genetic models from four groups of population in Figure 3. Each model can be represented by P_i. H_i represents a hybrid from parental groups. Ci represents the Child. The number in each circle denotes the number of population in each group of population. From Figure 3 there were 100 individuals in population 1, 500 individuals in population 2, 400 individuals in population 3 and 1000 individuals in population 4. It is not the case that all of the population from each group transfer their genetic to their child, for example H_1 received 50 individual from P_1 and 200 individuals from P_2. This relation can be written as $H_1 = 0.2P_1 + 0.8P_2$. While H_2 received only genetic from P_2 so $H_2 = 1.0*P_2$. Hardy-Weinberg law [31][32] can be applied to express this fact.

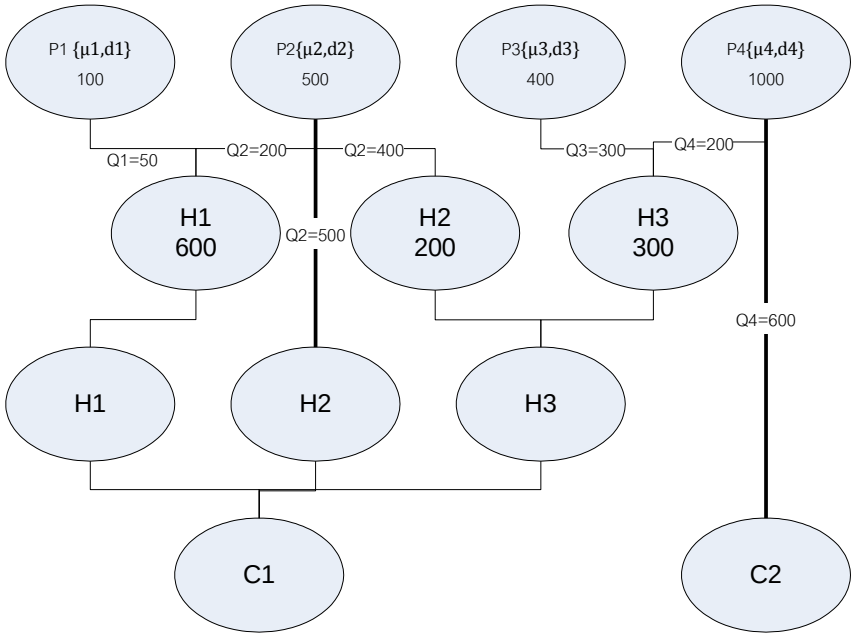

Figure 3 The genetic mixing from four population in each time step

C2 population in Figure 3 may directly inherited from only one group of ancestry as $C_2 = 1xP_4$ although the population is not the same as their ancestry, while population of C1 may mixed well with every groups of ancestry, the roots of the tree are P1, P2, P3 and P4 that formed C1.

2.2 Hardy-Weinberg Law [31][32]

The law explains the ratio of genetic from generation to generation. It is preserved and will not change over time. The allele frequency was persevered over time. The law can be used to explain the effect of Mendel's Pea Experiment [33].

2.2.1 Allele and SNP [34][35]

Allele is a set of possible genetic base in a locus of diploid. There were "Major Allele" represents the majority of allele in a locus and another set called "Minor Allele". The Major and Minor Allele may not express in a phenotype such as phenotype of blood group representation the blood type "A" may come from genotype "AO" or genotype "AA". Homozygosity wild type is the genotype that composed of two major alleles. The genotype composed of two different alleles was called Heterozygosity. Its variant is a genotype that composed of two minor-alleles. To determine whether the gene is dominance or recessive depended on an expression of the genotype. SNP was used to represent the possibility of chromosome in a locus. There were various way to determine the value of SNPs. This dissertation uses 0, 1 and 2 to represent the Homozygosity wild type, Heterozygosity and Homozygosity variant respectively.

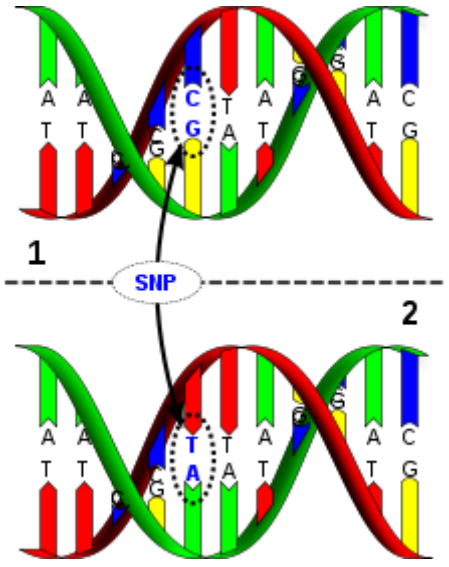

Figure 4 Show a chunk of chromosome and SNPs.

The two upper traits in Figure 4 were called genetic copy or “copy”. For example, in blood type the first copy may represent blood type “A” and the lower trait also represent the blood type “A” then this genetic has genotype “AA”, the dominance “A”.

2.2.2 Diploid [36]

Cells have two homologous copies of each chromosome such as a chromosome of mammal. Bacterial has only one copy of chromosome, called Haploid. In the mating of diploid a child will receive two copies of chromosome one from the mother and one from the father. Nearly all mammals are diploid organisms although all individuals have some small fraction of cells that display polyploidy.

2.2.3 Binomial Distribution [36]

Genetic data in SNPs format was used in this dissertation. Any loci of SNPs data composed of only two possible alleles such as either A or T. If the amount of A is more than T then A is a major allele while another is minor allele. The number of individual is N. The composition in a loci can be either AA or AT or TA or TT. AA was called Homozygosity wild type. AT or TA obtained the same genotype were called Heterozygosity wild type and TT was called Homozygosity variant. Binomial relationship can be applied with SNPs data relation in a locus as Eq 2

$$\text{Eq 2.} \qquad N^2=(A+T)^2 = AA + AT+TA + TT = A^2 + 2AT + T^2$$

From Allele and Diploid and Hardy-Weinberg Law, a locus of SNPs data can be represented by binomial distribution model [37] as P_i. There were two dependent parameters: means, μ_i and variance,δ_i.of a binomial distribution model in a locus. The mixed model may have different ratio from each ancestral group. θ_i represents the ratio of each ancestral model. Figure 5 shows the result of mixing of four binomial models. In general, the result of mixing model can be written as the relation in Eq 3.

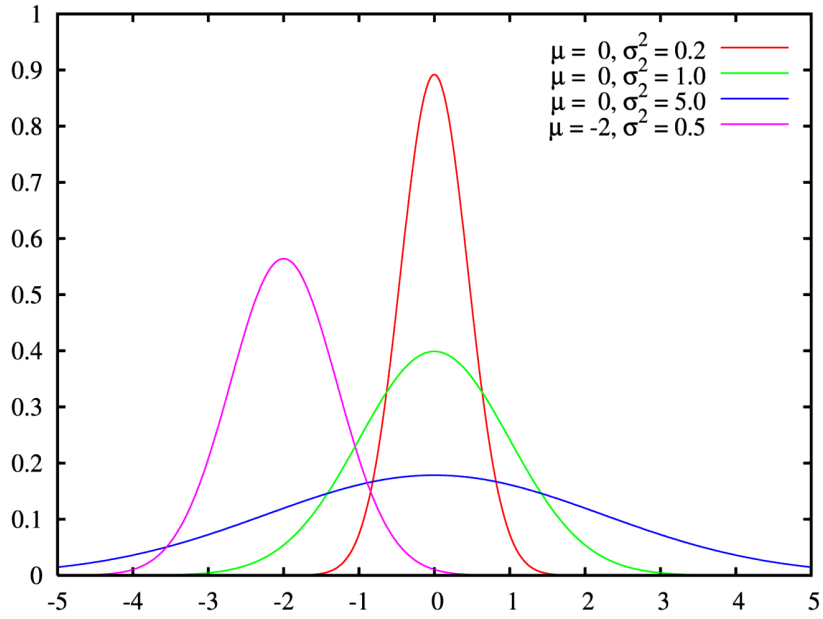

Figure 5 The mixture models that composed of four difference binomial distribution models

Eq 3. $$R(\mu_i, \delta_i, \theta_i) = \sum_{n}^{i=1} \theta_i * P(\mu_i, \delta_i)$$

Mixing model was applied in STRUCTURE [29]. It is based on Markov Chain to represent the mixed model that is to find the θ_i. This approach yields low accuracy [30]. It also does not consider the nature of genetic inheritance. Inheritance of genetic is not locus independence. Each parental genome is divided into chromosome segments, similar to a chunk of genome, and transfers to child.

2.3 Likelihood Model

Likelihood calculation [38] is an inverse calculation of mixture calculation. In mixture calculation, the goal is to infer the child from the known ancestry genetic distribution. For Admixture, given a mixed data, the goal is to infer the ratio of that mixing. The SNPs data do not provide any of ancestry genetic model representation and each model mixing ratio. Likelihood calculation is an inferring method to infer each unknown model representation and each unknown model ratio from the current data. The likelihood relation was proposed by Hanis in 1986 [10]. Hanis computed partial maximum likelihood with the likelihood relation. Tang applied EM with the likelihood relation. EM is a statistic based tool that was proposed by Dempster [11]. EM algorithm is a popular tool to solve the problem involving incomplete data and latent variables. EM [11] can be written as Eq 4, for likelihood model in each loci. **D** is a current

genetic data. $\mu_i,\ \theta_i$ are latent variables. μ_i is an allele frequency in each ancestral group *i*. θ_i is each ancestral group proportion. δ_i in Eq 3 can be ignored because in the loci of SNPs the Hardy-Weinberg Equilibrium is maintained, so the likelihood relation can be represented as Eq 4.

Eq 4. $$L(\mathrm{D}, \mu_i, \theta_i) = \sum_n^{i=1} \theta_i * P(\mu_i)$$

EM algorithm is an iterative procedure to compute the Maximum Likelihood based on log likelihood, so the whole dataset can be written as Eq 5 after applying the log-likelihood relation.

Eq 5. $$L(\mathrm{D}, \mu, \theta) = \sum_I^{i=1} \sum_M^{m=1} log \sum_K^{k=1} \theta_{ik} * \mu_{ki}$$

Dataset in this dissertation is a Single Nucleotide Polymorphisms, SNPs. SNPs is a composite of human genome data that composed of Major allele and Minor allele in each locus. There were two threads in each SNP, each thread is called copy. This means that the part of that genetic was a copy of the parent. The likelihood for mixing of genetic model in a binomial distribution is as Eq 6

Eq 6. $$L(P, Q) = \sum_I \sum_M \{q_{ik} * p_{km}^2 + 2 * q_{ik} * p_{km}(1 - p_{km}) + q_{ik} * ((1 - p_{km})^2\}$$

Eq 7 is the log-likelihood of Eq 6 and after applying the SNP value in each marker "m" of each individual "i".

Eq 7. $$L(P, Q) = \sum_I \sum_M \{g_{im} log \sum_K q_{ik} * p_{km} + (2 - g_{im}) log \sum_K q_{ik} * (1 - p_{km})$$

The SNPs data, G, consists of I individuals M markers, g_{ij}. Each marker of each loci has genotype either 0, 1 or 2 representing the homozygous wide type heterozygous wide type and homozygous variant type, respectively. A 0 means the first copy is "A" and the second copy is "A", A 1 means the first copy is "A" while the second copy is "a" and 2 means both the first and second copy are "a". The relationship has the term $g_{im} log \sum_K q_{ik} * p_{km}$ represents the effect of only major allele. The term $(2 - g_{im}) log \sum_K q_{ik} * (1 - p_{km})$ represents the effect of minor

allele. q_{ik} denotes the mixing ratio from the k^{th} ancestral group in the i^{th} individual. p_{km} denotes the allele frequency of major allele in the m^{th} loci of k^{th} ancestral group, $(1-p_{km})$ denotes the minor allele of the m^{th} loci of k^{th} ancestral group.

Tang [6] proposed a method to solve the likelihood relation by Newton-Raphson (NR) as Eq 8. NR requires a differential calculation to compute the Jacobian's matrix, used in the relation. However the matrix operator in NR is difficult to scale when implemented in FPGA. Although the NR method guarantees the quadratic time of convergence [39] it does not guarantee the global solution and it is also dependent on the initial starring point.

Eq 8. $$(P^{n+1}, Q^{n+1}) = (P^n, Q^n) - \frac{L(P^n,Q^n)}{\frac{\Delta L(P^n,Q^n)}{\Delta(P^n)}} - \frac{L(P^n,Q^n)}{\frac{\Delta L(P^n,Q^n)}{\Delta(Q^n)}}$$

Alexander proposed a block relaxation method that is based on quasi Newton Raphson. This method also does not work well with FPGA implementation. Tang proposed the calculation step of EM in three steps. They are Initialize value Step, Maximization Step and Expectation Step. The EM method must add an observable variable a_{im} for the first copy and b_{im} for the second copy at m^{th} loci and i^{th} individual. a_{im} and b_{im} can represent the group of ancestral value in each loci of each individual as well.

Initial Step: Randomly assign "1" to only one cell of k at the i^{th} individual m^{th} loci of a_{im} and b_{im}.

Maximization Step: Maximize the value of P by Eq 9 and maximize the value of Q by Eq 10

Eq 9. $$p_{mAk}^{n+1} = \frac{\sum_I g_{im} a_{imk}^n}{\sum_I g_{im} a_{imk}^n + \sum_I (2-g_{im}) b_{imk}^n}$$

Eq 10. $$q_{ik}^{n+1} = \frac{1}{2M} \sum_M [a_{imk}^n + b_{imk}^n]$$

Expectation step: Re-expect the value of "A" and "B" in each marker of individual from maximized value of ancestry allele frequency and admixed ratio. The expectation step calculates the ancestry groups' ratio at m^{th} marker from m^{th} allele and mixing ratio of i^{th} individual as: Eq 11 and Eq 12.

Eq 11. $$a_{imk}^n = \frac{q_{ik}^n p_{mAk}^n}{\sum_K q_{ik}^n p_{mAj}^n}$$

Eq 12. $$b_{imk}^n = \frac{q_{ik}^n (1-p_{mAk}^n)}{\sum_K q_{ik}^n (1-p_{mAj}^n)}$$

The operators in EM are simple mathematics operators. The calculations needs a high precision that FPGA support through the floating point calculation block. Convergent criterion is calculated by Eq 13

Eq 13. $$\text{“}L(P^{n+1},Q^{n+1})-L(P^{n},Q^{n}) < e\text{”}$$

“e” is stopping criteria. The iteration stops when the current likelihood value is not different from the previous likelihood value.

CHAPTER 3

DESIGN AND IMPLEMENTATION

This dissertation presents a method to speed up the calculation by Field Programmable Gate Arrays (FPGA) [21]. Fully parallel operators are a prefer choice of this dissertation. To achieve the performance of fully parallel calculation unit it is necessary to unroll the loop of calculation and then assign each task to each calculation unit. The limit of hardware resource causes an insufficient number of calculation units. This dissertation applies the systolic array computing to achieve scalability of the calculation. A systolic array [40] was applied to overcome a limitation of hardware resource. A systolic array is an arrangement of processors in an array where data flows synchronously across the array. The systolic array also has a specialized form of parallel computing. Each cell of arrays is an independent computing unit that computes data and stores data independently. Finally each cell shares information to the cell beside itself in the specified direction. This dissertation applied an IP-core floating point operators from the library of Xilinx [43][44] that are based on the IEEE754 standard [45] to gain more precision than fixed point calculation. A USB circuit is used as a communication channel between FPGA and PC. The synthesis and implementation tools are Xilinx ISE Design Suit v13.2 [46] with Verilog compiling language. Verilog is used to synthesis a calculation circuit and java language is used to generate a Verilog coding of the circuits from a design script.

The method of a calculation starts from loading SNPs data from host CPU to FGPA. This step allocates only 2-bit per allele copy. All possible SNPs in this dissertation are {0, 1, 2}. For unknown data, it was biased to be 0. The number 0, 1 and 2 represent Homogeneous wild type, Heterogeneous and Homogeneous variant respectively. The register size of I×M×K 32-bits were allocated for "A_{imk}" and "B_{imk}". The P-value, Q-value and AB-value are represented by floating point single precision IEEE754 format. The Q-value allocates I×K×32-bit amount of FPGA register. The P-value allocates amount of M×K×32-bit and the AB-value allocates 2×I×M×K×32-bit of FPGA's register as shown in Figure 6. The array of registers is provided to speed up the calculation by hardware calculation as described below.

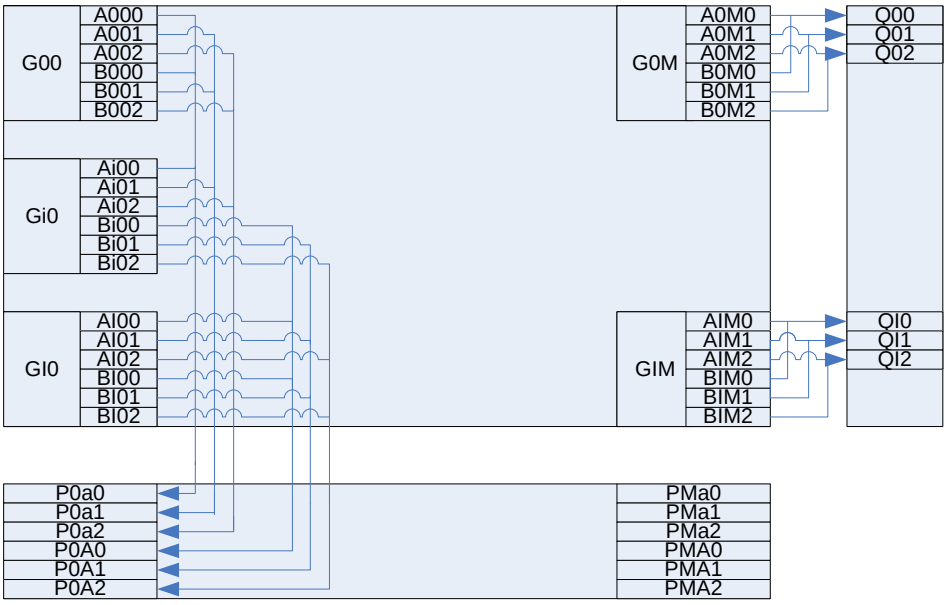

Figure 6 The register allocation in FPGA circuit

The units of registers in the calculation of the equations, in previous chapter, were represented in this Figure. There were several types of register representation in this dissertation. Floating point format was applied with register units of Aimk Bimk Qik and Pmlk while unsigned short format was applied with register unit of Gim. The vertical arrow in Figure 6 represents dependencies of Gim Aimk and Bimk on Pmlk as the Eq 9 and the horizontal arrow represent the dependencies of Aimk and Bimk on Qik as the Eq 10 as in Maximization step. The maximization step is to read the value of Pmlk and Qik and calculate as the Eq 11 and Eq 12 to update the value of Aimk and Bimk.

3.1 Floating-Point

There are many number representations, such as the fixed point number system using 1's complement arithmetic or 2's complement arithmetic. This dissertation applied the IEEE754 floating point number format for the calculation to avoid the precision loss from the cumulative operations. Figure 7 shows the composition of IEEE754 format.

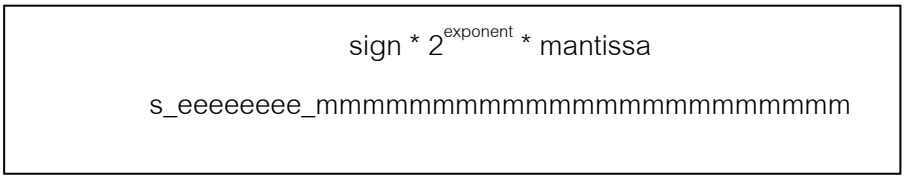

Figure 7 A 32-bit number of IEEE754 format for floating point calculation

The number format is composed of three parts. The first part is a sign bit, 1-bit wide, 0 for positive and 1 for negative. The second part is an exponent, 8-bit wide in binary number and the last part is a mantissa, 23-bit wide. To transform any decimal number to the IEEE754 floating point number, it starts from a conversion to binary number then transforms to exponential form and fills in the “m” part. The exponent is transformed into a binary number and then fill to the “e” part.

A library of Floating point operation is provided by Xilinx. The operators in this dissertation are add, sub, multiply and divide. The circuits of calculation are proposed in Section 3.2.1 for updateP circuit and Section 3.2.2 for updateQ circuit and Section 3.2.3 for updateAB circuit.

3.2 Circuit Design of EM calculation

The calculation of likelihood as proposed in Chapter 2 was implemented as circuits of FPGA. This dissertation proposed a method of fully parallel of processing unit. The main idea is to perform loop unrolling and scattering of memory to each calculation unit. The EM method of calculation to solve the likelihood relation as proposed in Chapter 2 was chosen. There were three steps of calculation. The first step of EM is randomized the initial point of solutions. This step is starting from randomly assigns the group of ancestral to each copy in any individual (called as initial step). The constraints of these values are $\sum_K A_{imk} = 1$ to determine the probability of the major allele of the m^{th} marker of i^{th} individual and $\sum_K B_{imk} = 1$ to determine the probability of the minor allele of the m^{th} marker of i^{th} individual. The second step is Maximization step as Eq 9 (updateP) and Eq 10 (updateQ). The calculation direction of Eq 9 can be simultaneously operated with Eq 10. The operation of Eq 9 is row dependent of data because the operation is based on a summation over vertical data from the first individual through the last individual of a locus. The individual admix ratio in Eq 10 is data dependent from the summation over row of each individual. The last step is Expectation step. This step re-determine the probability of ancestral group for each individual copy. The re-determining is similar to a calculation of mixture model. The step of Maximization must be completed before starting an Expectation step. The convergence of the calculation is determined by Eq 13. Each of calculation of the three steps can be implemented as follows.

3.2.1 Circuit Design for UpdateP

This circuit is used to calculate a value of ancestral allele frequency in each loci, P_{mAk}/P_{mBk} as Eq 9. It is a part of calculation to maximize the ancestry allele frequency. The calculation of each column does multiply all k variables of A_{im} by G_{im} and accumulate through the last I^{th}. Then divide each k^{th} variable of A_{im} by $A_{im}+B_{im}$

```
zero all P[m][k]
for m = to size of Marker{
        for I = 0 to size of Individual {
                for k = 0 to size of Ancestral group {
                        tmpA[k] = tmpA[k] + G[i][m] * A[i][m]
                        tmpB[k] = tmpB[k] + (2-G[i][m])*B[i][m]
                }
        }
        P[m][k] = tmpA[k] / (tmpA[k] + tmpB[k])
}
```

The calculation does a summation over individuals in a locus and the result is a new ancestral allele frequency for the next iteration. The pseudo code shows the method of updateP that is implemented on a single processing unit. For multiple of processing units the calculation can be transformed to

$$Pmk = \frac{G0m*A0mk + G1mk*A1mk + ..+ Gimk*Aimk + ..+ GImk*AImk}{(G0m*B0mk + ..+ Gimk*Bimk + ..+ GImk*BImk + G0m*B0mk + ..+ Gimk*Bimk + ..+ GImk*BImk)}$$

There were three types of circuit in updateP. The first circuit multiplies G_{im} by A_{im} and B_{im} then sends the result to the second circuit. The second circuit multiplies the current G_{im} by A_{im} and B_{im} and adds the current result with the previous result, and then sends it to the next circuit through the last individual data. The third circuit is the same as the second type of updateP but it also adds a block of addition circuit between $G_{im}\times A_{imk}$ and $G_{im}\times B_{imk}$. This is as a denominator of the divide circuit to complete the updateP calculation of each loci. The head of updateP circuit contains only 2k blocks of *mul* to multiply G_{im} by A_{im} and multiply G_{im} by B_{im}. There were 3 types of input ports. The first type is 2-bit wide for G_{0m}, the second type is 32-bit wide k ports for A0mk and the last type is 32-bit wide k ports for B0mk. The output ports are 32-bit wide 2k ports. It is shown as the top block of Figure 8. The updateP circuit contains 2k blocks of *mul* and *add* circuit to multiply G_{im} by A_{im} and multiply G_{im} by B_{im} and adds the result with the updateP

circuit. There were four types of input ports. The first type is 2-bit wide for G_{0m}. The second type is 32-bit wide k ports for A_{0mk}. The third type is 32-bit wide k ports for B_{0mk} and the last type is 32-bit wide 2k ports of the previous A_{imk} and the previous B_{imk}. The output ports are the same as the head of updateP circuit. It is shown as the middle block of Figure 8. The last updateP circuit is almost the same implementation and input/output port as an ordinary updateP circuit but the operator performs add of the ancestral major allele number with the ancestral minor allele number. The result is a norm of ancestral allele frequency. The k units of *add* block and *div* block were added to the circuit. The input ports are the same as an ordinary updateP but there are only 32-bit wide k ports of the output to the ancestry major allele frequency.

The calculation circuit is shown in Figure 8. Please note the connection of the three types of updateP circuit. The left side of the circuit is the register of each locus of individual. Aim implies major allele. Bim implies minor allele and G is the genotype value. The A_{imk} and B_{imk} have k possible values from k groups of ancestral. This value is selected to update the ancestral allele frequency of each ancestral group as P_{mAk}/P_{mBk}. The update depends on genetics value of the loci of each individual, $G_{im} = 0$ means Major-Major (AA), $G_{im} = 1$ means Major-Minor (AB) and $G_{im} = 2$ means Minor-Minor (BB). From the relationship of binomial distribution and log-likelihood (Eq 9), the genetic value 0 denote the $P_{0Ak} \mathrel{+}= A_{imk} + B_{imk}$. The genetic value 1 denotes $P_{0Ak} \mathrel{+}= A_{imk}$, $P_{0Bk} \mathrel{+}= B_{imk}$ and the genetic value 2 denotes $P_{0Bk} \mathrel{+}= A_{imk} + B_{imk}$. The ancestral allele frequencies were calculated only for Major Ancestral Allele Frequency to reduce the circuit size. The minor ancestral allele frequency is implied by 1- major ancestral allele frequency. It is shown at the bottom of Figure 8. Figure 8 shows the calculation circuit of the first marker for three ancestor groups.

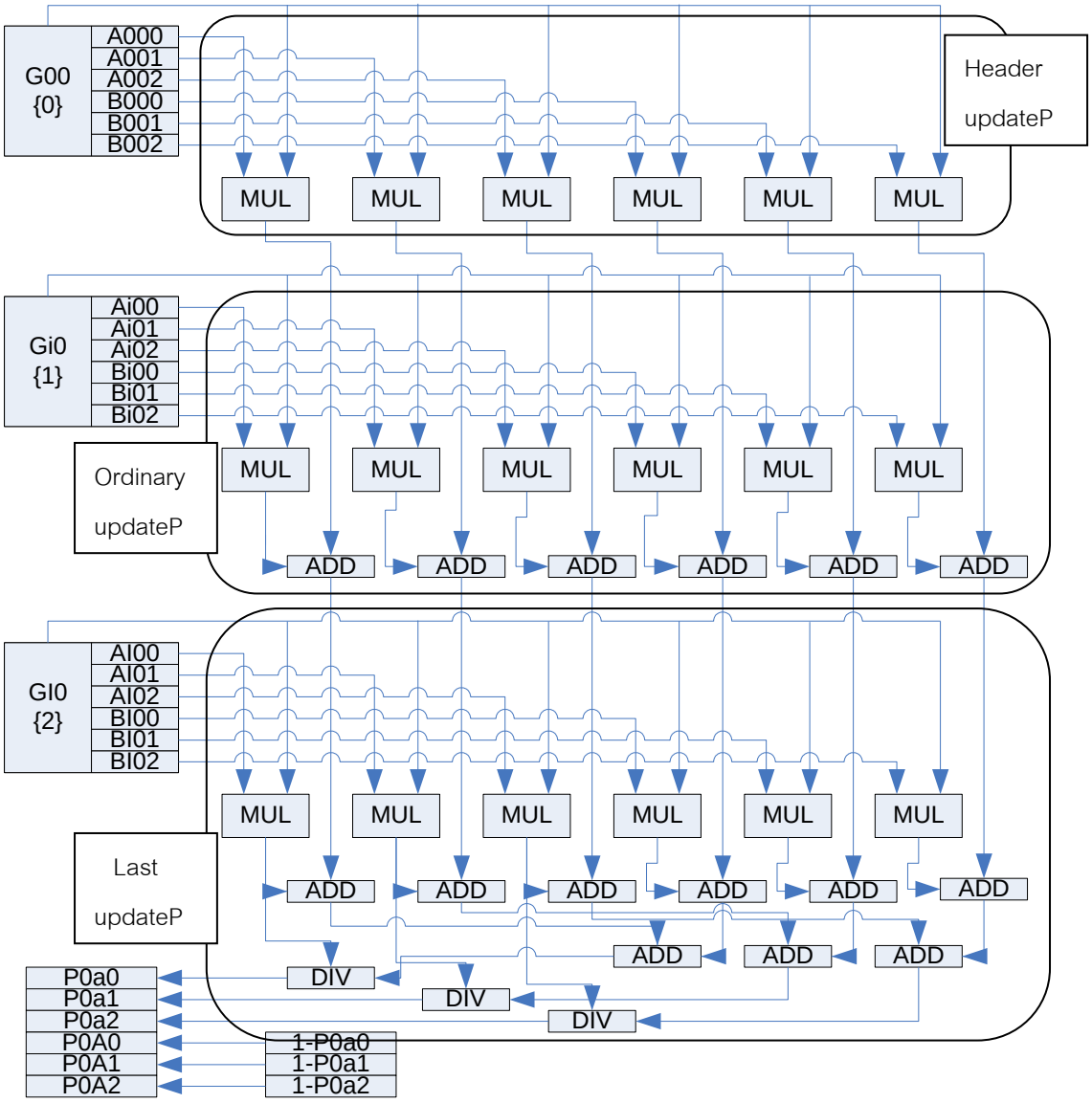

Figure 8 the “UpdateP” circuit in column 0

3.2.2 Circuit Design for UpdateQ

This circuit calculates the individual admix ratio (Q_{ik}) as Eq 10. It was used in Maximization step. To update the new value of Q_{ik}, it sums every value of A_{imk} and B_{imk} that match the k^{th} ancestral group by row (Figure 9). There is a direct connection for each k^{th} registers in the circuit of the A_{imk} and B_{imk} because each register represents the inheritance ratio of each marker of individual. For example A_{123} represents the 3^{rd} ancestry group probability of the 2^{nd} marker of 1^{st} individual. There are 32-bit wide k variables per individual to compute the mixing ratio of each ancestral group in each individual. The sum of these values in each individual is 1.0. The pseudo code of the implementation of Eq 10 is as follows.

```
zeros all Q[I][K]
for i = 0 to size of Individual{
        for m = 0 to size of markers{
                for k = 0 to number of ancestral groups
                        Q[i][k] += A[i][m][k]/2M + B[i][m][k]/2M;
        {
}
```

The computation for a single processing unit starts from clearing the admix ratio (Q_{ik}) from the previous calculation. Then cumulatively adding the value of A and B row by row through the last row. The implementation for a multiple processing unit can be implemented as follows.

```
Q00 = (A000+B000+A010+B010+..+A0m0+B0m0+..+A0M0+B0M0) / 2M
Q01 = (A001+B001+A011+B011+..+A0m1+B0m1+..+A0M1+B0M1) / 2M
Q0k = (A00k+B00k+A01k+B01k+..+A0mk+B0mk+..+A0Mk+B0Mk) / 2M
Qik = (Ai0k+Bi0k+Ai1k+Bi1k+..+Aimk+Bimk+..+AiMk+BiMk) / 2M
QIk = (AI0k+BI0k+AI1k+BI1k+..+AImk+BImk+..+AIMk+BIMk) / 2M
```

The implementation of individual admixture ratio is shown in Figure 9. There is only one type for input port. It is 32-bit wide and there are 2k x M x K ports for value of A and B and 32-bit wide k ports for the output. The circuit requires M x K units of computing in a row as shown in Figure 9. The sub unit calculations were applied for the entire row of genetic data per individual. Figure 9 shows a part of the calculation circuit that consists of three ancestral groups, K=3. The calculation block was connected to every register from the genotype (G) and the first and the second copies of every marker in an individual.

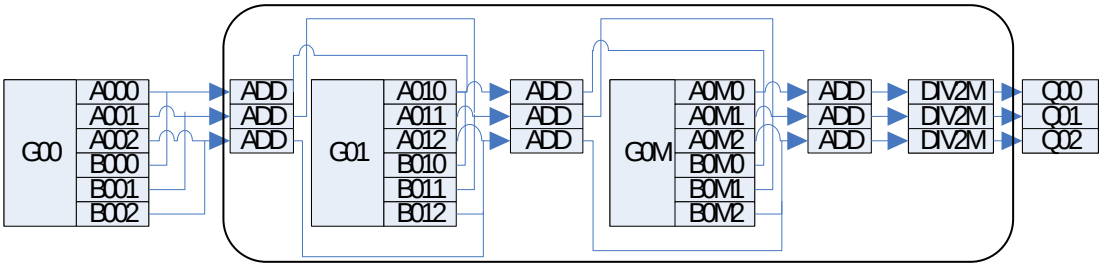

Figure 9 the circuit of UpdateQ

3.2.3 Circuit Design for UpdateAB

After the calculation of the Maximization step is completed, the next calculation is the Expectation calculation. This step computes the re-expect value of the latent variables of A_{imk} and B_{imk} that imply the group of ancestry in a locus of each copy of genetic data. The values are the probability of each marker multiplied by the mixing ratio as Eq 11 for major allele and Eq 12 for minor allele. The method of calculation is to sum over ancestral groups for $q_{ik} \times p_{mk}$ for major allele and $q_{ik} \times (1-p_{mk})$ for minor allele. Values of A_{imk} and B_{imk} depend on the genetic data to select either major or minor ancestral allele frequency value. The pseudo code of Eq 11 and Eq 12 as follows.

```
for i = 0 to size of Individual {
    for m = 0 to size of Marker {
        major = 0; minor = 0;
        for k = 0 to size of Ancestral group {
            major = P[m][k]*Q[i][k] + major
            minor = (1-P[m][k])*Q[i][k] + minor
        }
        for k = 0 to size of Ancestral group {
            A[i][m][k] = P[m][k]*Q[i][k] / major
            B[i][m][k] = (1-P[m][k])*Q[i][k] / minor
        }
    }
}
```

The first loop is a calculation of mixing model of the m^{th} marker in the i^{th} individual. The result is a denominator of each copy. This calculation was used in the second loop to represent the inheritant proportion of each copy from its ancestor. The pseudo code above shows the operation of a single processing unit. The thread of computing is started by calculating from the first marker of the first individual through the last marker of the last individual to complete the Expectation step. It is important to have the independence of data access of the calculation units to ancestral allele frequency and individual admix ratio. The design for multiple calculation units uses multiple registers to achieve this effect. Multiple registers can be accessed at the same time, assuming there were I×M units of updateAB circuits. The calculation unit of updateAB can be implemented as follows.

A[i][m][0] = P[m][0]*Q[i][0] / (P[m][0]*Q[i][0]+ P[m][1]*Q[i][1]+..+ P[m][K]*Q[i][K])

B[i][m][0] = (1-P[m][0])*Q[i][0] / ((1-P[m][0])*Q[i][0]+ (1-P[m][1])*Q[i][1]+..+ (1-P[m][K])*Q[i][K])

A[i][m][1] = P[m][0]*Q[i][1] / (P[m][0]*Q[i][0]+ P[m][1]*Q[i][1]+..+ P[m][K]*Q[i][K])

B[i][m][1] = (1-P[m][0])*Q[i][1] / ((1-P[m][0])*Q[i][0]+ (1-P[m][1])*Q[i][1]+..+ (1-P[m][K])*Q[i][K])

…

A[i][m][K] = P[m][0]*Q[i][K] / (P[m][0]*Q[i][0]+ P[m][1]*Q[i][1]+..+ P[m][K]*Q[i][K])

B[i][m][K] = (1-P[m][0])*Q[i][K] / ((1-P[m][0])*Q[i][0]+ (1-P[m][1])*Q[i][1]+..+ (1-P[m][K])*Q[i][K])

The implementation is suitable for the bottleneck-less architecture as FPGA or multiple processing units which have enough internal registers. There was 32-bit input/output bus for each unit of A and B as shown in Figure 6. The ancestral allele frequency (P_{mk}) and individual admix ratio (Q_{ik}) were directly accessed from each computing unit to update their own registers of A_{imk} and B_{imk}. The genotype selects P_{mk} to be calculated. The selection process of P_{mk} is shown in Figure 10, Figure 11 and Figure 12.

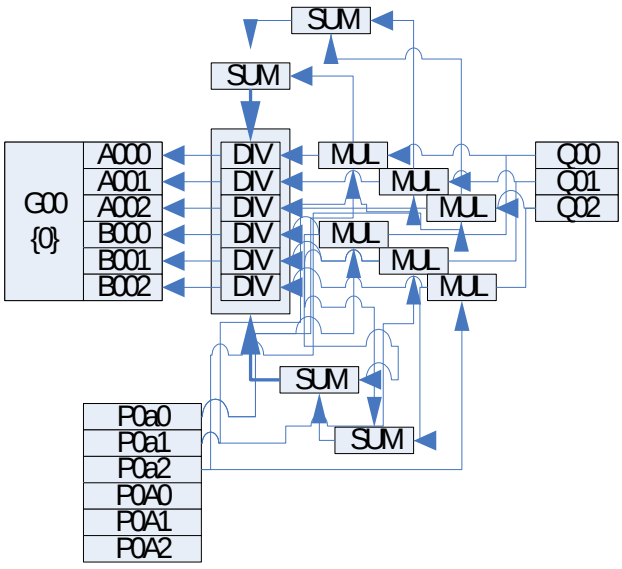

Figure 10 The circuit of UpdateAB: genotype is “0”. The genotype of each copy is Major or Homogenous wild type. Only ancestral major allele frequency was selected.

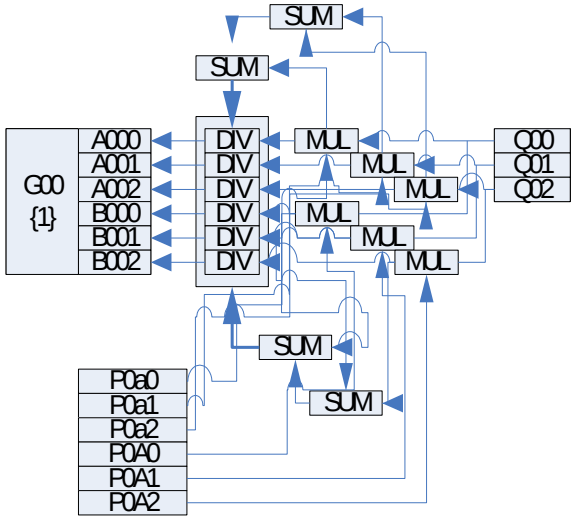

Figure 11 The circuit of UpdateAB: genotype is “1” (Heterozygous). Both Major and Minor ancestral alleles were selected.

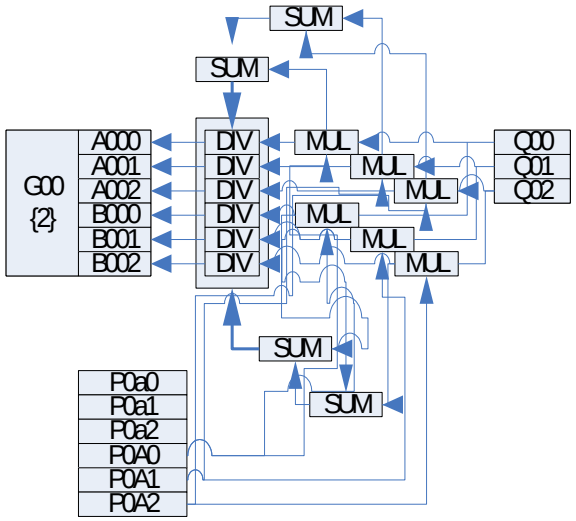

Figure 12 The circuit of UpdateAB: genotype is “2”. Only ancestral minor allele frequency was selected.

Figure 10 shows a calculation circuit when the genotype of the locus is “0”. The circuit will select only the ancestral major allele frequency (P_{mk}). When genotype is “1”, the major and minor ancestral allele frequencies are selected. When it is “2”, only ancestral minor allele frequency ($1-p_{mk}$) is selected to operate as Eq 11 and Eq 12 respectively. The architecture of this step is totally different from the calculation of Maximization step as 3.2.1 and 3.2.2. The genotype data is shared between multiple calculation units. For the proposed architecture here, there is no conflict in accessing data between multiple units. However, the mutual exclusion for sharing data access is a major problem for a conventional CPU.

Figure 8 - Figure 12 were proposed calculation circuits in this dissertation. Figure 6 is a circuit of register unit. The arrows in Figure 6 were operations of maximization step. The circuits in Figure 8 and Figure 9 perform concurrent update concurrently update the value of Pmlk and Qik.

3.2.4 Circuit Design Summary

The architecture proposed in the previous section consumes a lot of resource as registers and a lot of floating point operation units as shown in Table 1.

Table 1 The sub-calculation unit usage in each circuit

Circuit	Number block of Adder	Number blocks of multiplier	Number blocks of Divider	Number of the block in overall
updateQ	2K	-	K	I×M
updateP	2K+K	2K	K	I×M
updateAB	2(K-1)	2K	2K	I×M

There are a lot of floating operator blocks. The adder blocks have to instantiate I×M×(K+2K+K+2(K-1)) blocks of the overall circuit. The multiplier blocks have to be instantiated I×M×(2K+2K) blocks and I×M×(K+K+2K) blocks of the divider blocks. The huge amount of all floating point blocks overflows the resource available in FPGA devices. The calculation method of updateP and updateQ cause the abundant of carry ripple from data dependency in column and row. Even if the calculation units are fully implemented, there is still a bottleneck in speed up due to the ripple of carry. The Systolic architecture was applied to this dissertation. In general case, the multiplier circuit handles any value multiplies by genotypes value (0, 1 or 2). To reduce the size of the circuit, it is specialized to the fixed values of 0, 1 or 2. This specialization also reduces the time of floating point operation.

3.3 Systolic Array [40]

Systolic array was applied for this dissertation to reduce the number of calculation unit. Systolic array architecture is based on parallel processing and remarkable advances in VLSI technology. Many systolic algorithms have been designed for a great diversity of areas. This

dissertation applies systolic architecture for hardware resource saving. The calculation step is divided into two dependent steps from Expectation Maximization algorithm. The first part is Maximization step and the second part is Expectation step. The circuit of updateP and updateQ were enabled in the Maximization step and updateAB was enabled in the Expectation step. A concern of a systolic architecture is a flow of data into the calculation units. There are several ways to flow the data into a calculation unit such feeding row-by-row as Figure 13A or feeding column-by-column as Figure 13B. The strip-wise of calculation units was called bundle-wise. Figure 13C shows the bundle-wise architecture. Figure 14 shows the flow of data by block-wise method. The data were fed through the array of Data Processing Units (DPUs) in two directions. The block-wise architecture was very popular to apply with matrix multiplication [42]. The process of matrix multiplication by systolic array is shown in Figure 14. The first step vertically feeds the B[0][0] and horizontally feeds A[0][0] to the calculation node C[0][0] to multiply and accumulate as C[0][0] += A[0][0]×B[0][0]. The second step, the data of B[1][0] and A[0][1] are fed to process in C[0][0] and the result becomes C[0][0] += A[0][1]×B[1][0]. At this step the data of B[0][0] is fed to C[1][0] to be multiplied by A[1][0] and the result is C[1][0] += A[1][0]×B[0][0]. Other element of C is calculated similarly as C[0][1] = A[0][0]×B[0][1], and so on. The time complexity of matrix multiplication via systolic array is O(n) while the matrix multiplication via single thread machine takes $O(n^3)$, but the systolic architecture requires n^2 of computation nodes.

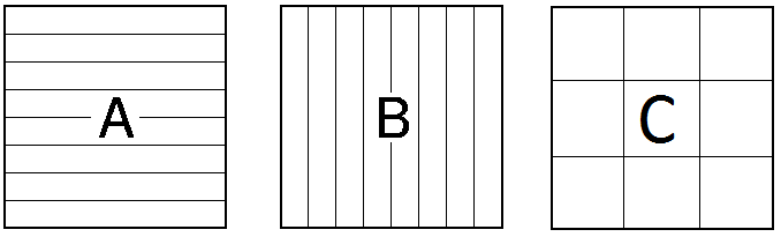

Figure 13 data partitioning for each architecture of calculation unit

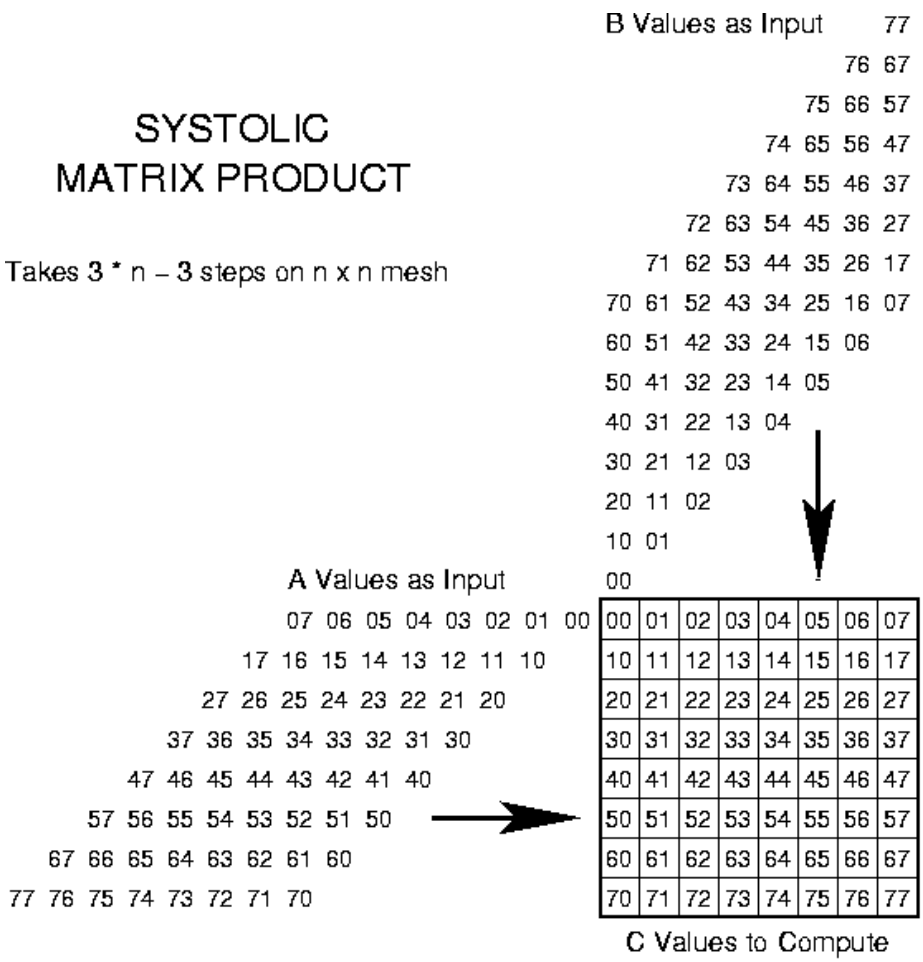

Figure 14 Data Processing Units (DPUs) of Systolic Architecture from [41]

Full placement of calculation circuit requires I×M circuits. This takes too much of resource. Placement of calculation circuits by row-wise requires M circuits. Finally placement of by column-wise requires I circuits. It is the choice in this work. Most of genetic data, the size of marker is significantly greater than the size of individual. To complete the calculation of EM, each iteration requires two steps of Expectation and Maximization. The main iterative of EM requires the first iteration or step of Maximization and another iteration is step of Expectation. Therefore, column-wise I circuits are the minimum resource required to implement this computation.

3.3.1 Circuit Design for Maximization Calculation

The original calculation of maximization step of EM was shown in Figure 15. There were I×M blocks of calculation circuits to process I×M array of genetic data. Figure 16 shows the

circuit after applying systolic architecture by column-wise for maximization step of EM calculation.

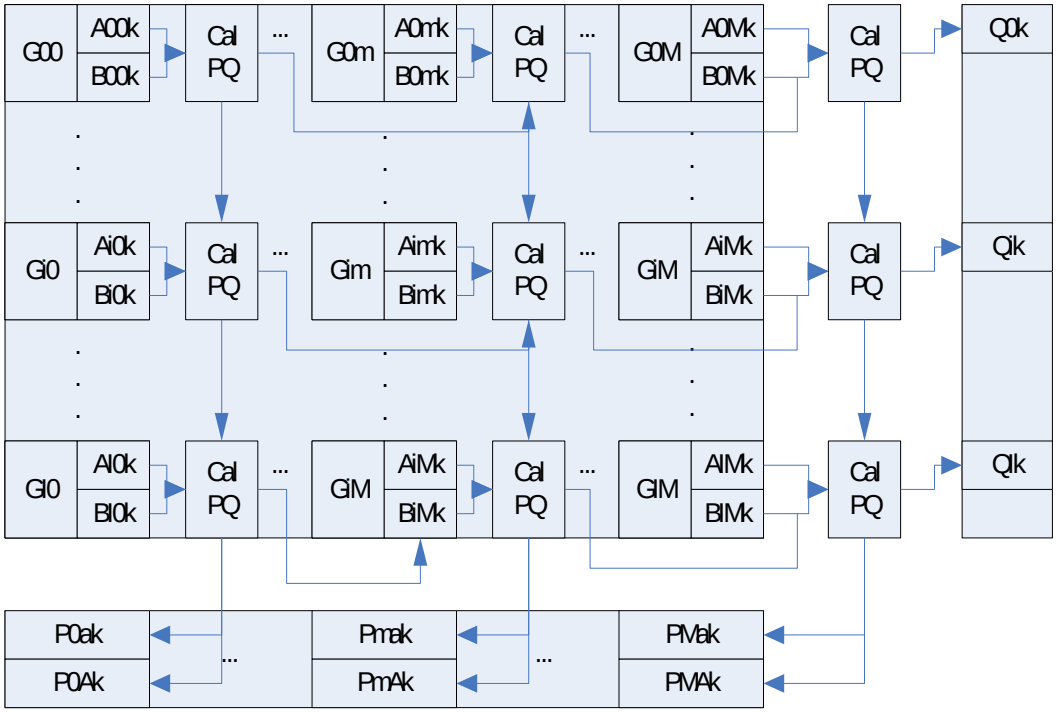

Figure 15 the original circuit of the maximization part of likelihood calculation

Common sub-calculation units are the most important part of the systolic architecture. The common sub-calculation units were described in Section 3.2. The implementation merges the sub-calculation units of updateP and updateQ for simultaneous calculation of maximization and update in the step of expectation. All of the implemented circuits are based on a column-wise architecture as shown in Figure 16.

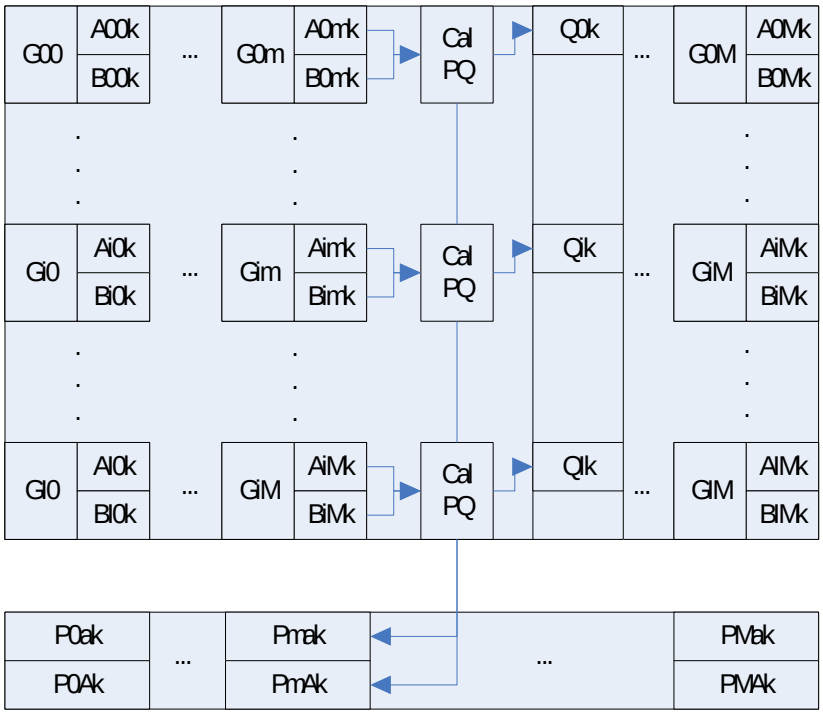

Figure 16 The circuit after applying column-wise of systolic architecture to the maximization step.

Figure 16 shows the merged circuit of updateP and updateQ into CalPQ that was placed in M circuits. The horizontal operation of CalPQ circuit performs the summation of Q[i][k] = A[i][m][k]+B[i][m][k] from the first of marker through the last of marker and divides by 2M. This operation is similar to updateQ. The horizontal operation of this circuit of each column takes O(1). It takes around 6 clocks of floating point adder. The vertical operation of the CalPQ circuit performs the summation of the first individual through the last individual of each marker. This operation takes O(I). The circuits for ancestral allele frequency calculation were adapted for more general use than the circuit updateP. In the full placement architecture, there were three types of updateP circuit. They are differing on the value of genotype. For reusability, all of three circuits must be implemented in each CalPQ circuit, because the genetic value may be changed due to each genotype of each locus. The multiplexer of each updateP circuit was changed to multiplier block as Eq 9. The vertical operation takes O(I) from a ripple carry of each sub-calculation unit because the result of A[i][m][k]×g[i][j] and B[i][m][k]×g[i][j] has been prepared in each CalPQ circuits as Figure 17. The time complexity of Maximization step is O(I×M), that is not significantly slower than the full placement circuit of CalPQ or the placement by row-wise. Registers allocation of the column-wise architecture still be the same as the full placement architecture.

The flow of calculation in Maximization step started from the first locus of genetic data. The data will be shifted to the next right column when the ancestry allele frequencies of every group were updated. The circuits of CalPQ were shifted to the second locus and so on through the M^{th} locus.

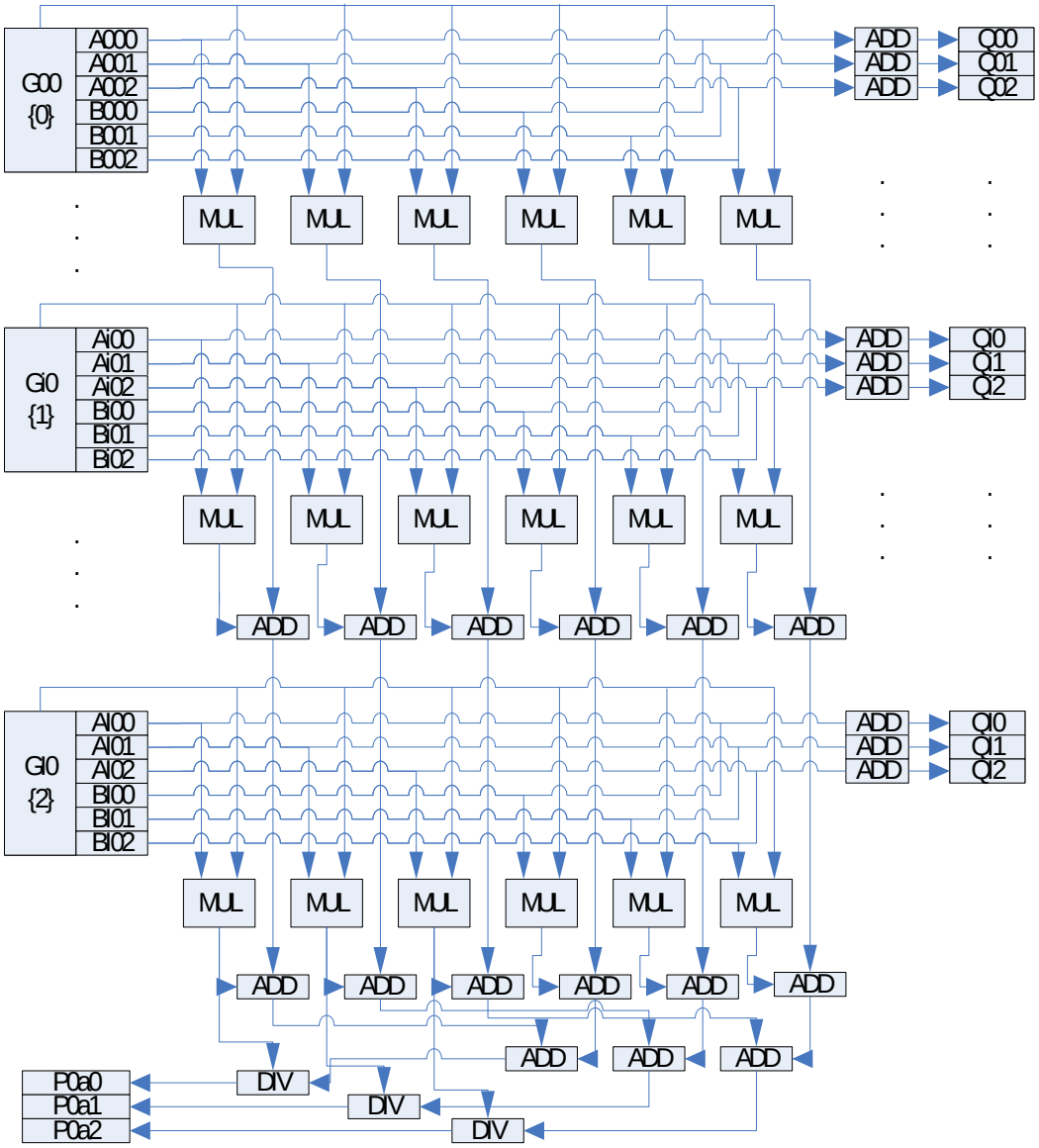

Figure 17 The circuit of CalPQ circuit of systolic architecture.

3.3.2 Circuit Design for Expectation Calculation

This calculation step starts after finishing of Maximization calculation. In this step the value of latent variables are updated after the parameters were Maximized, ancestral allele frequency and individual admix ratio have been maximized from the Maximization step. The original design of Expectation step is shown in Figure 10 - Figure 12. The different circuits were

selected depended on the genotype value. The genotype 0 selects the circuit of Figure 10, 1 selects the circuit of Figure 11 and 2 selects the circuit of Figure 12. The updateAB circuit in the previous section may save a hardware resource more than the circuit in this section but the previous circuit lacks of general use. The hardware resource is smaller and faster for multiplexer than the multiplier block. That is the reason why multiplexer was selected for the updateAB circuit. The multiplexer was changed to the operation of Eq 11 and Eq 12 for making a more general circuit of updateAB as Figure 18.

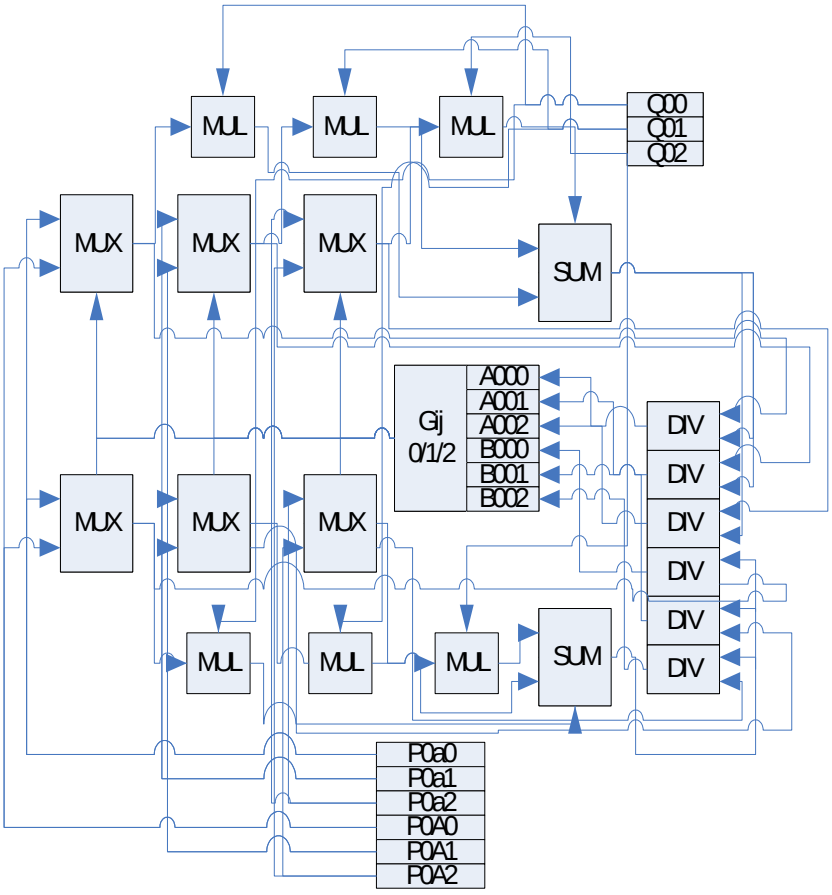

Figure 18 The circuit for expectation step.

The column-wise architecture was also applied to updateAB circuit. The time complexity of column-wise architecture is improved from O(M) to O(1). Each latent variable was updated from a separately access of ancestral allele frequency and admix ratio. There were M markers to transverse in an iteration, so the time complexity of column-wise updateAB is O(M).

The circuits of calculation are based on binary operands of floating point operators and binary operation with integer and floating point. The floating point circuit of IP-Cores from Xilinx

were applied with this dissertation. This dissertation proposed a wrapping of floating point operators thus transforms them into building blocks.

3.4 Circuit Implementation

This dissertation proposed an implementation of circuit via Verilog language. There are three types of circuit in this dissertation. The first circuit is Register Unit for genetic data, ancestral-allele-frequencies, admix-ratios and latent variables. The second circuit is calculation circuits for updatePQ and updateAB. The last circuit is control unit to control the flow of data and the check for stopping criterion.

3.4.1 Register Unit

There are four blocks of Register unit as shown in Table 2. Speed, higher accuracy and efficient hardware usage are the main aim of the design. To achieve speed up, registers are placed near the calculation units. This can reduce the bottleneck of a single data bus from data unit (RAM Cache or Register) to the calculation unit. IEEE754 floating point data format was applied with the circuit. A 32-bit data width is chosen for the implementation. The precision may not be as good as a 64-bit width but it is better than 32-bit fixed point format. Genetic data registers, only 2-bit data, are designed to fit register allocation of the circuit. The operations of both G[i][j]×A[i][j][k] and G[i][j]×B[i][j][k] were changed from 32-bit floating point operands to 32-bit floating point and 2-bit short operands.

Table 2 The size of register allocation

Genotypes	There were I×M block of registers, each block takes two bits. **Space complexity is I×M×2 bits**
Admix ratios	There were I×K block of registers, each block takes 32 bits. **Space complexity is I×K×32 bits**
Ancestral Allele frequencies	There were M×K block of registers, each block takes 32 bits. **Space complexity is M×K×32 bits**
Latent variables	There were I×M×K blocks of register, each block takes 32 bits. **Space complexity is I×M×K×32 bits**

The register for Column-wise architecture were adjusted to 32×M-bit wide register. There are I×K blocks of this register. The control unit has only the shift-left operation. These

latent variables are moved to the next 32-bit of the data and a 2-bit shift-left operation is for genotype data.

3.4.2 Calculation Unit: Wrapping Circuit

The calculation circuits are constructed from a basic sub-calculation block such as a floating point adder, a multiplier and a divider. Xilinx IP-cores provided the block of floating point operators used in this dissertation. This dissertation proposed a wrapping technique to envelop the floating operators of IP-cores for the proposed circuit. The wrapping technique is based on handshaking signal of Asynchronous data transfer. The operation time of each floating point operator is different. There was no register of the output signal to sustain the result. The original floating point timing diagram of Xilinx IP-cores is shown as Figure 19 -Figure 21.

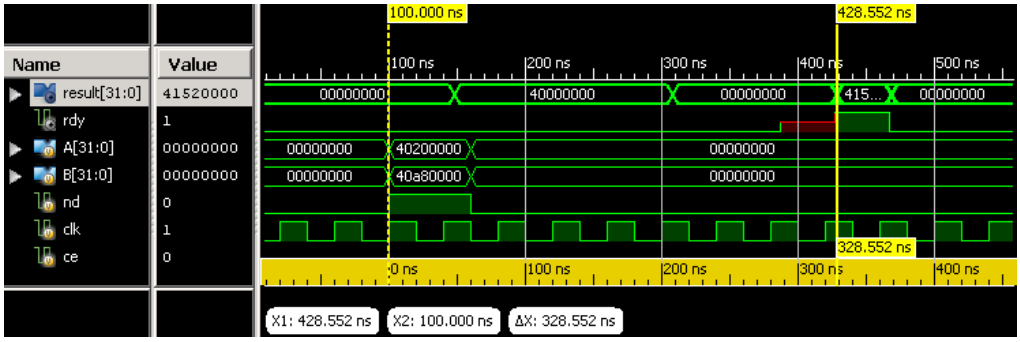

Figure 19 the timing diagram of floating point multiplier circuit

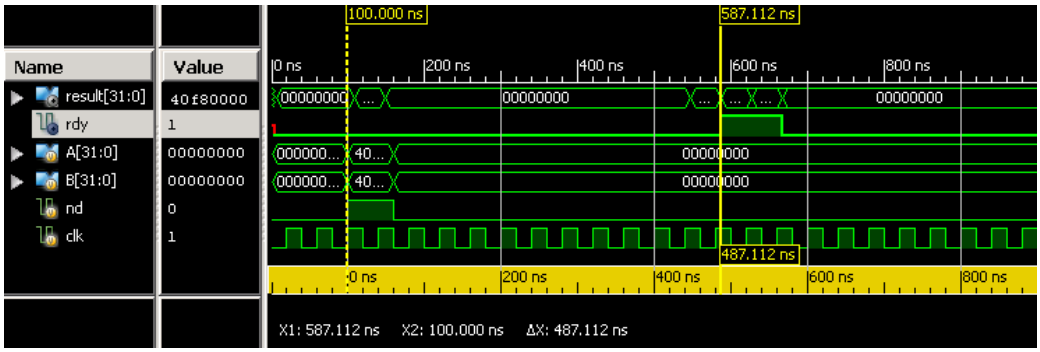

Figure 20 the timing diagram of floating point adder circuit

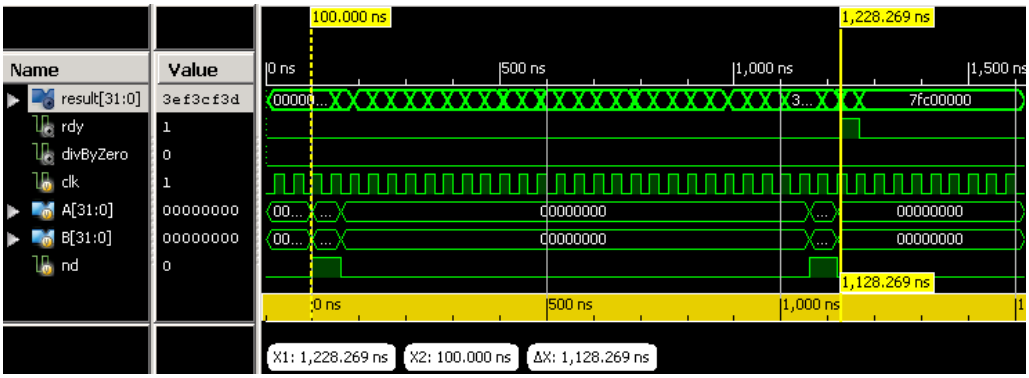

Figure 21 the timing diagram of floating point divider circuit

The timing results were obtained from the post Place&Route simulation of ISim[48]. The post Place&Route simulation is a simulation that is closed to real gate implementation in the FPGA device. The timing profiles of every device were put into the circuit in the placement step. The routing profiles of every wiring were assigned to the pins between the devices in routing step. Figure 19 - Figure 21 show the timing diagram of the original floating point operators of 2.5×5.25, 2.5+5.25 and 2.5/5.25 respectively. The timing result did not show the signal to enable the calculation blocks. The chip-enable (ce) is set to “1” and clear (sclr) is set to “0”. All of the results of the operations are valid only on the “1” of “rdy” signal. The operators will be enabled on the rising edge of new-data (nd) signal. Time of data serving is a major problem. For example, in Figure 8, the add block is waiting for the result of the multiply block while another side of incoming data was served. There are several solutions. The first solution is to build a state machine to control the timing of all calculation circuits. The second solution is applying a circuit handshaking for acknowledge and response between a communication of the blocks called wrapping block.

The state machine was applied to this dissertation first. It is easier to design and control. The work load of calculation circuit was transferred to control circuit. There were a lot of control signals through every of calculation blocks. Handshaking signals were used. The control block only monitors the signals from the last direction of each calculation circuit. The wrapping blocks as shown in Figure 22 compose of “AND” gate D-Flip-Flop (DFF) and the original calculation blocks. The AND gate was used to enable the calculation. It is the same as operation of the nd signal. The original calculation block was active when both of the input data had been served and the nd signal was raised. The AND gate was used to determine the status of each port of

the input data as shown in the Table 3. Figure 23 shows the connection of validA and validB signal to the nd signals of the calculation block.

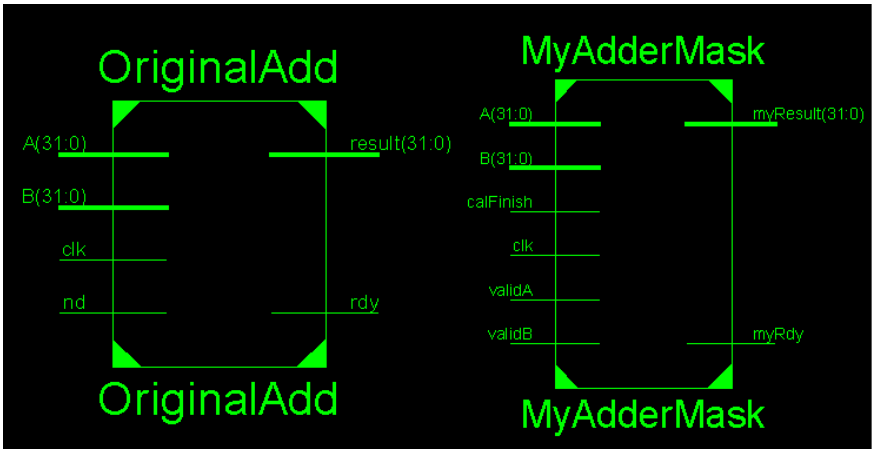

Figure 22 The original circuit and wrapping circuit of a floating point adder.

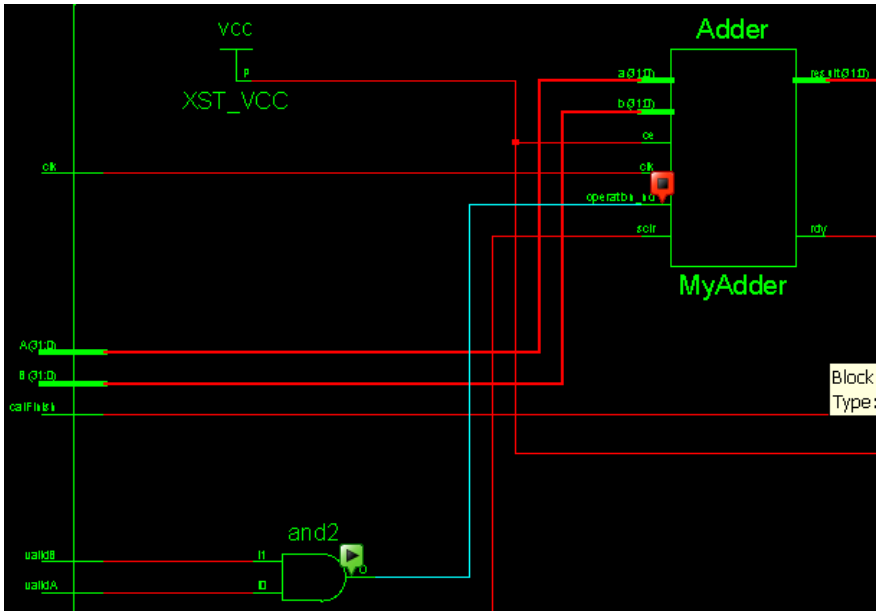

Figure 23 The input part of the wrapping block

Table 3 Control signal to enable the wrapping circuit.

validA	ValidB	Start Calculation
0	0	No
0	1	No
1	0	No
1	1	Yes

DFF were used to sustain the value of the result and rdy signals and waiting for a response from the target of calculation block. The target of the calculation responses to the rdy signal of the previous calculation block with the calFinish signal. The calFinish signal was used

to release the calculation block to accept the new input data. The calFinish signal was designed for local calculation blocks. It is not the same as the state machine controller that every signal have to be connected to the control block. The connection signals are status signals to acknowledge the actions. Figure 24 shows the myRdy signal connected to validB and the result of the Adder block was fed to the port B of the next Adder block. The calFinish signal will be raised from the next calculation after complete calculation of updateQ.

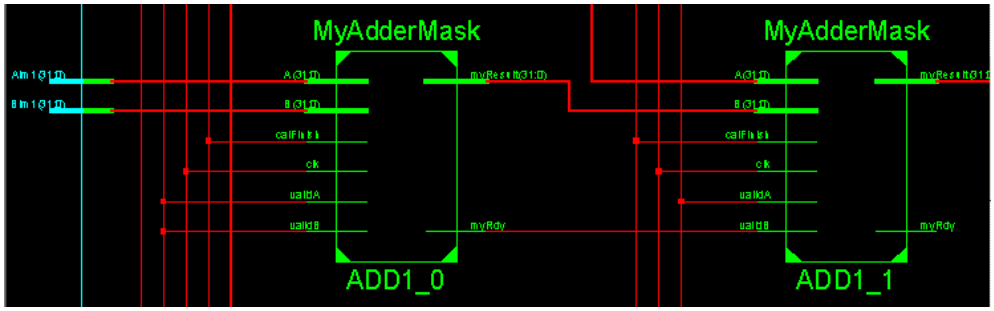

Figure 24 A part of calculation in updataQ circuit.

The calFinish signal was used to reset the value of DFF holding as shown in Figure 25.

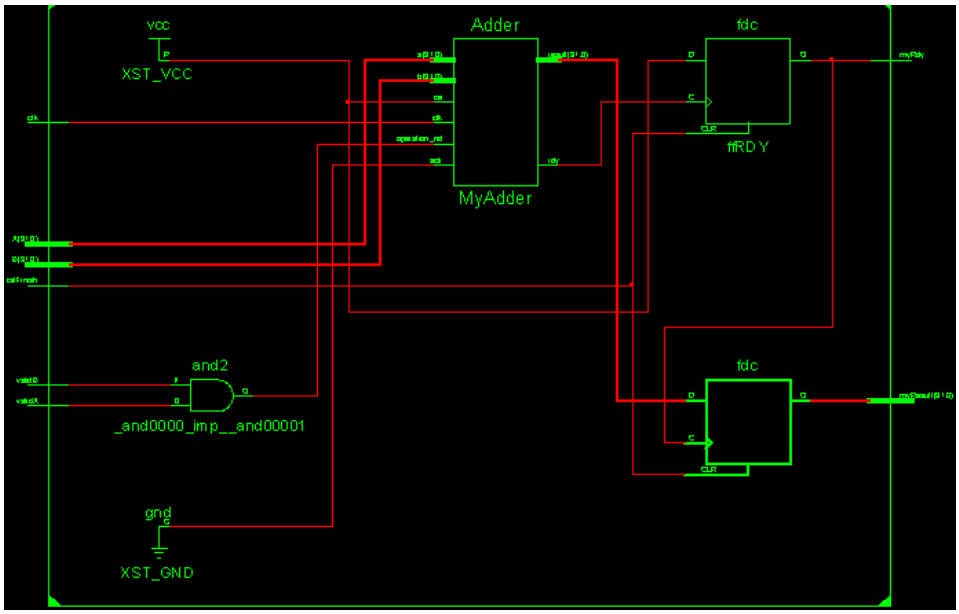

Figure 25 The wrapping circuit of adder that composes of AND gate and DFF

Figure 25 shows the connection of DFF and the output of Adder. The result of Adder will be latched into DFF on the next positive edge of myRdy signal that came from the DFF block. The DFF for myRdy signal was always connected to “1”. It is waiting for any raising edge of rdy signal of the original adder block to hold the status of complete calculation and latch a new data. Figure 26 shows the timing of the wrapped adder block. The circuit takes around 521ns and waits for the response of calFinish signal as shown in Figure 27.

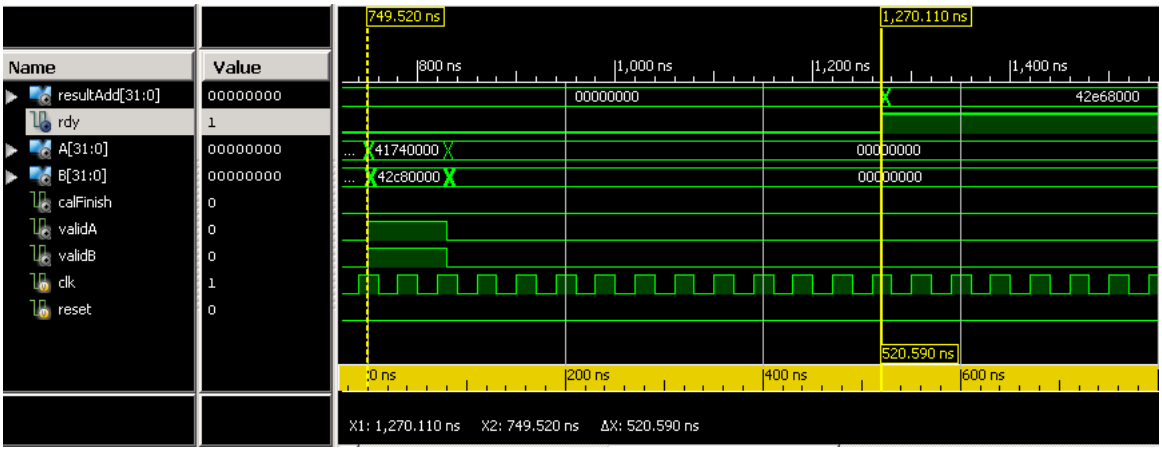

Figure 26 Timing diagram of the wrapped adder block before the calFinish signal

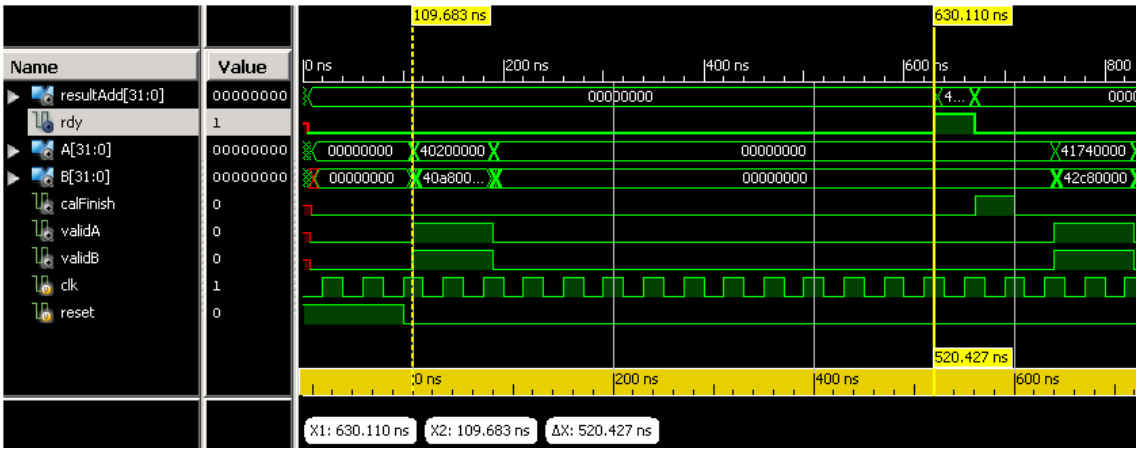

Figure 27 Timing diagram of the wrapped adder block

The timing diagram as shown in Figure 26 demonstrates that the result has not been used in the next calculation block. The resultAdd will not be changed when the block receives the calFinish signal as shown in Figure 27.

3.4.3 Control Unit

The circuit was designed to control the flow of data of the calculation of EM. The control circuit uses 32-bit rotate right register units for latent variables and 2-bit rotate right registers for genotype. OddEven is a 1-bit register. It controls the selection circuit of updateAB or updatePQ. The value of register is increasing after the end columns of data were operated. The first round OddEvent is “0” to enable the calculation of updatePQ circuit for Maximization step. The second round OddEvent is “1” to enable the calculation of updateAB circuit for Expectation step. There were three output control signals, start column, next column and last column. The start column signal was fed to clear the register value of updatePQ circuit. The updatePQ circuit is a cumulative sum from the first column through the last column of each individual admixture

calculation. It also performs a cumulative conditional sum of first row of latent variables through the last individual latent variable. The result is the ancestral allele frequency. This signal has not been connected to the updateAB circuit because there is no cumulative operation of the method. The control circuit is waiting for the myRdy signal of the last updateP circuit before feeding the next data. The second signal is nextCol. This signal generates a pulse to start the operation of updatePQ and updateAB circuit. The signal was directly connected to the input of validA and validB of every calculation blocks to prepare the first level calculation. This signal generates a pulse at the finishing of each column calculation for feeding the next data. The last signal is endCol. This signal was designed for the calculation of individual admixture. The signal enables the operation of the divider to divide the cumulative sum by 2M. The control diagram was shown in Figure 28.

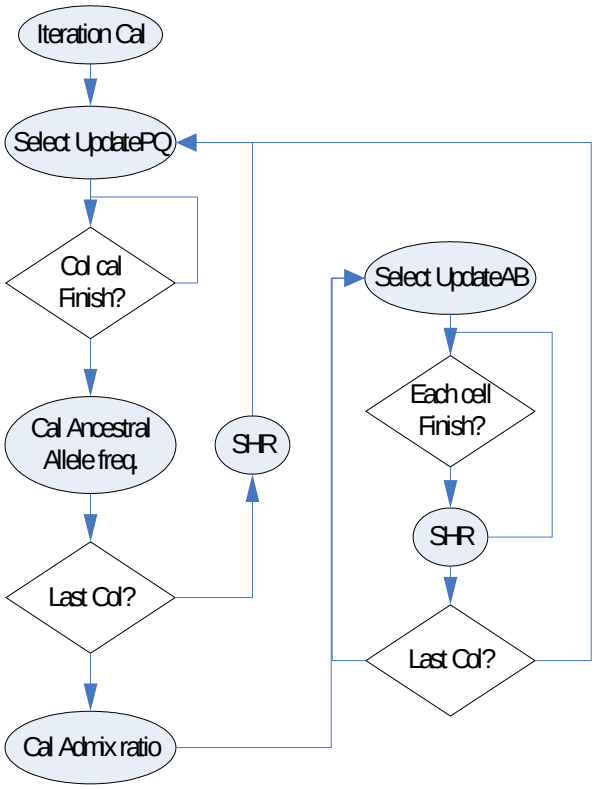

Figure 28 The control logic in an iteration of EM.

3.4.4 The sub-calculation unit of updatePQ

The sub-calculation unit of updatePQ is shown in Figure 29. The circuit was created by the connection of calculation block in Maximization step. The circuit of updateP and updateQ were grouped to operate together in each step of calculation. The circuit composes of 2K Multiplier blocks and 2K+K of Adder blocks and 32-bit K blocks of registers. There were vertical

and horizontal operation directions. The vertical direction calculates the ancestral allele frequency and the horizontal operation performs admix-ratio calculation. The circuit was created from floating point operator blocks and registers. The register was used as temporary variables for the cumulative sum in the horizontal operation. It is called PrevQ. This registers are reset on every startCol signal from the control Unit. The circuit was enabled by the pulse of nextCol signal. The vertical outputs compose of ancestral allele frequency and rdy signal. The rdy signal will be a status of the circuit and was connected to the control unit to acknowledge the circuit operation. This signal was connected to the single pulser circuit called calFinish signal as shown in Figure 25.

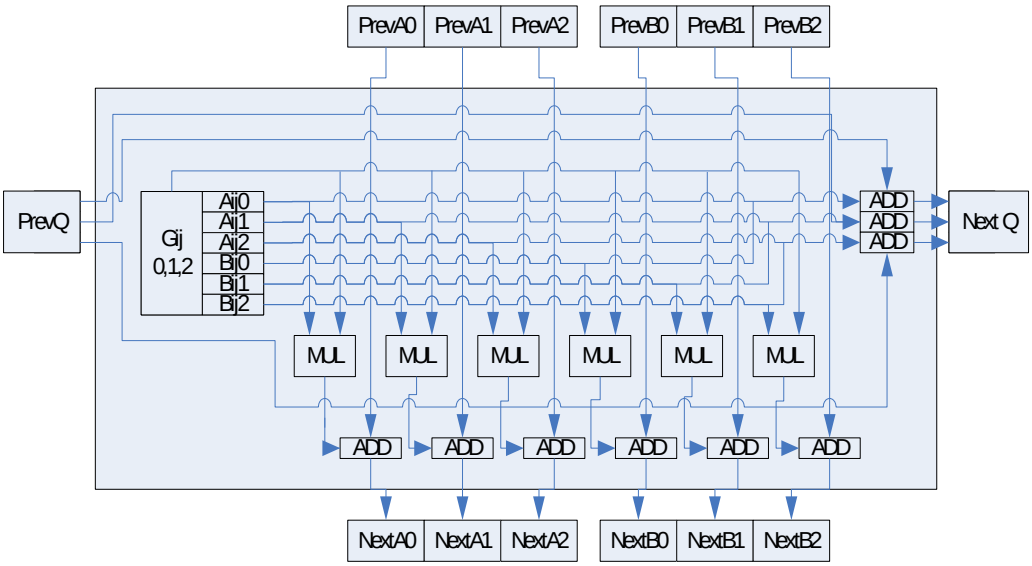

Figure 29 The connection of sub-unit calculation for each cell in updatePQ circuit

The calFinish signal resets the value of every register in updatePQ circuit to prepare them for next data operation. The temporary register was used in systolic architecture while the full circuit has no registers. The column-wise architecture has been applied after the sub-calculation units of updatePQ circuits. The number of sub-calculation units is the number of Individual, I as shown in Figure 16.

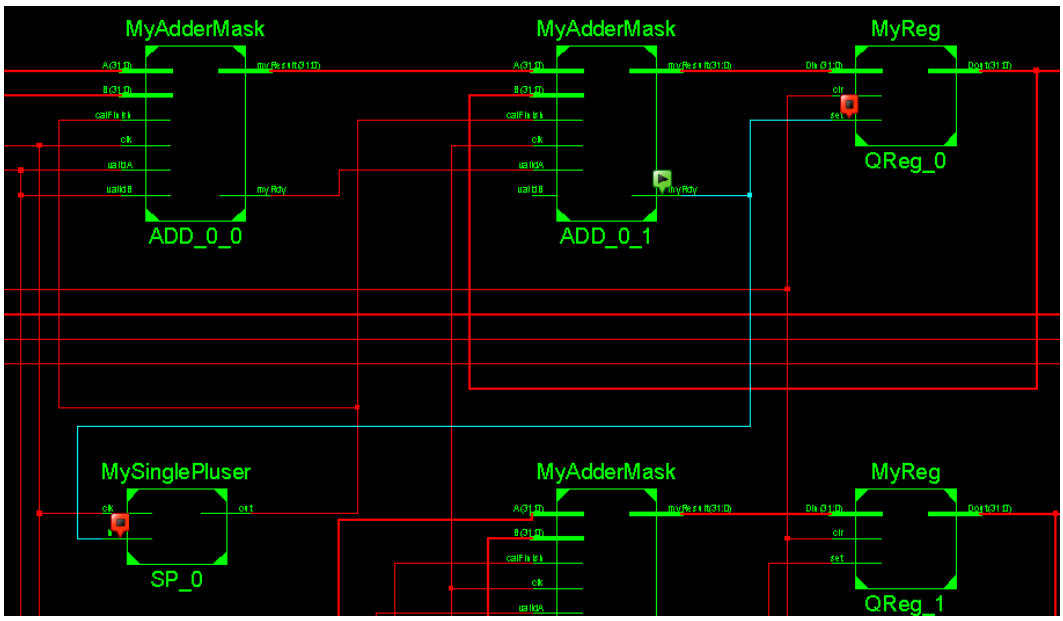

Figure 30 The connection of calFinish within the calculation block

3.4.5 The sub-calculation unit of updateAB

The circuit updates the latent variable in every locus of the individual. The input of calculation is the previous individual admixture and the ancestral allele frequency as Eq 11 and Eq 12. The updated latent variables are the outputs. The nextCol signals from the control unit are input control signal to start the calculation of each locus of an individual. The calFinish signal was also applied to the circuit from the control unit. Finally, myRdy signal is the output to the control unit. This signal was also used to acknowledge the status of the calculation circuit. The circuit connection is the same as in Figure 18.

3.4.6 Byte-Float Multiplication block (Mul012)

The operation of float-byte was used in the circuit of updateP as the Eq 9. The circuit was designed for multiply the floating point of latent variables by only two bit of genotype data. The value is either 0, 1 or 2. The Mul012 circuit reduces the size of all hardware and time of operation. The operation separates the IEEE754 floating point 32-bit format of latent variables into three parts as Figure 7. The sign part is the same for all operations. The part of mantissa is 0 if genotype is 0. The part of exponent will be increased if the genotype is 2, the same is true when the genotype is 1 and 0 for genotype 0. The interface of the Mul012 block is the same as the others operators.

CHAPTER4

SUMMARIZATION AND FUTURE WORKS

4.1 Dissertation Summarization

Recently in the research literature most of floating point operations has been transferred from software calculation into hardware calculation to gain more speed. This dissertation proposed to generate the resizable hardware circuit, aimed for gain up the calculation performance via FPGA. One of the advantages of FPGA is the fully parallel processing unit, especially floating calculation. The performance of the calculation can be increased in two directions. The first one from the direct use of FPGA floating-point operators. The second one is the use of the fully parallel architecture that each computing unit can be independently operated. The design of floating point and byte operation can reduce both operation time and hardware resource usage.

4.2 Design Summarization of Hardware Utilization

The implementation and synthesis tool in this dissertation is ISE Project Navigators version 13.4. ISim is a simulation tool in this dissertation. ISim[48] is integrated with ISE design suite. The Xilinx libraries as IP-cores were built-in. The simulation provides four different levels of timing precision. The first level is “Behavioral” simulation. This level, the simulator only parses the source code and directly generates the timing from the source code. Simulation at this level is fast. Source Code understanding is also a goal of this simulation level. The second level is “Post-Translate”. This level of simulation was used to verify the functionality of the source code after it is translated into the hardware. The hardware profile of the translation of Verilog/VHDL code to logic gate such as Verilog code “case” command may be translated to Mux circuit according to the technology of each device. The available devices are Spartan, Vertex5, Vertex6, Vertex7 etc. The next level is “Post-Map”. This level gains more precision of the hardware profile by adding the timing delay of each hardware profile from the previous level. The last level is the level of “Post-Place” or “Post-Place & Route”. In this level of simulation,

every profile, timing delay profile of each hardware devices and placement profiles of the wires from the hardware connections as net-list profile, are included for the simulation result. This level of the simulation is much closer to the timing of circuit simulation. The results from different series of technologies cause the different of net-list profiles and hardware timing delay profiles such as; the net-list profile of Vertex-7 series was fast because the 3D routing technology.

The simulation starts from testing every sub-calculation block such as floating point adder, multiplier, divider and byte float multiplier. The simulator randomly generates the numbers in IEEE-754 floating point format and feeds them into the circuits to verify the result of each operation. The next summary is a hardware resource usage of each calculation block. This summary was generated for two hardware series, Spartan and Vertex, to illustrate the trend of hardware usage. The original operators and the wrapped operators were compared. Table 4 - Table 6 show the hardware resource usage for the Spartan3 xc3s400, which has 8064 logic cells. Table 7 - shows the hardware resource usage for the Virtex6 xc6vlx75t, it has 74496 logic cells. The Table 7 - Table 9 shows only some part of the synthesized result because there are many different resources between the series of Xilinx chip.

Table 4 The hardware resource usage of floating point adder for Spartan-3

Adder Utilization Summary (Spartan3)	Wrapped			Original		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Flip Flops	**582**	7,168	8%	**581**	7,168	8%
Number of 4 input LUTs	590	7,168	8%	589	7,168	8%
Number of occupied Slices	460	3,584	12%	470	3,584	13%
Number of Slices containing only related logic	460	460	100%	470	470	100%
Number of Slices containing unrelated logic	0	460	0%	0	470	0%
Total Number of 4 input LUTs	594	7,168	8%	593	7,168	8%
Number used as logic	545			544		
Number used as a route-thru	4			4		
Number used as Shift registers	45			45		
Number of bonded IOBs	101	173	58%	99	173	57%
IOB Flip Flops	**32**					
Number of BUFGMUXs	2	8	25%	1	8	12%
Average Fanout of Non-Clock Nets	2.38			2.39		
Maximum Frequency:	206.303MHz			206.303MHz		

Table 5 The hardware resource usage of floating point multiplier for Spartan-3

Multiplier Utilization Summary (Spartan3)	Wrapped			Original		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Flip Flops	**690**	7,168	9%	**696**	7,168	9%
Number of 4 input LUTs	608	7,168	8%	627	7,168	8%
Number of occupied Slices	410	3,584	11%	436	3,584	12%
Number of Slices containing only related logic	410	410	100%	436	436	100%
Number of Slices containing unrelated logic	0	410	0%	0	436	0%
Total Number of 4 input LUTs	643	7,168	8%	662	7,168	9%
Number used as logic	565			584		
Number used as a route-thru	35			35		
Number used as Shift registers	43			43		
Number of bonded IOBs	101	173	58%	101	173	58%
IOB Flip Flops	**32**					
Number of BUFGMUXs	2	8	25%	1	8	12%
Average Fanout of Non-Clock Nets	2.51			3.22		
Maximum Frequency:	165.358MHz			165.358MHz		

Table 6 The hardware resource usage of floating point divider for Spartan-3

Divider Utilization Summary (Spartan3)	Wrapped			Original		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Flip Flops	**1,350**	7,168	18%	**1,349**	7,168	18%
Number of 4 input LUTs	812	7,168	11%	811	7,168	11%
Number of occupied Slices	859	3,584	23%	868	3,584	24%
Number of Slices containing only related logic	859	859	100%	868	868	100%
Number of Slices containing unrelated logic	0	859	0%	0	868	0%
Total Number of 4 input LUTs	814	7,168	11%	813	7,168	11%
Number used as logic	751			750		
Number used as a route-thru	2			2		
Number used as Shift registers	61			61		
Number of bonded IOBs	102	173	58%	100	173	57%
IOB Flip Flops	**32**					
Number of BUFGMUXs	2	8	25%	1	8	12%
Average Fanout of Non-Clock Nets	1.82			1.82		
Maximum Frequency:	180.183MHz			180.183MHz		

Table 7 The hardware resource usage of floating point adder for Virtex-6

Adder Utilization Summary	Original			Wrapped		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Registers	**591**	93120	0%	**592**	93120	0%
Number of Slice LUTs	592	46560	1%	593	46560	1%
Number of fully used LUT-FF pairs	374	809	46%	374	811	46%
Number of bonded IOBs	99	240	41%	101	240	42%
Number of BUFG/BUFGCTRLs	1	32	3%	2	32	6%
Maximum Frequency	650.745MHz			650.745MHz		

Table 8 The hardware resource usage of floating point multiplier for Virtex-6

Multiplier Utilization Summary	Original			Wrapped		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Registers	**703**	93120	0%	**704**	93120	0%
Number of Slice LUTs	672	46560	1%	673	46560	1%
Number of fully used LUT-FF pairs	565	810	69%	565	812	69%
Number of bonded IOBs	101	240	42%	101	240	42%
Number of BUFG/BUFGCTRLs	1	32	3%	2	32	6%
Maximum Frequency	596.908MHz			596.908MHz		

Table 9 The hardware resource usage of floating point divider for Virtex-6

Divider Utilization Summary	Original			Wrapped		
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization
Number of Slice Registers	**1358**	93120	1%	**1359**	93120	1%
Number of Slice LUTs	832	46560	1%	833	46560	1%
Number of fully used LUT-FF pairs	696	1494	46%	696	1496	46%
Number of bonded IOBs	100	240	41%	102	240	42%
Number of BUFG/BUFGCTRLs	1	32	3%	2	32	6%
Maximum Frequency	625.900MHz			625.900MHz		

Table 4 - Table 9 shows the correlation of hardware resource usage from different technologies of Spartan-3 and Virtex-6. The usage also correlates with the original floating point operators and the wrapped operators. This result can be used to predict the hardware resource usage of each FPGA board. It indicates how many calculation blocks can be placed in the FPGA board. The operation speed of different technologies are also a concern. The routing of Virtex-6 has a special technique called "*Number-used-exclusively-as-route-thrus*". It produces a shorter routing path than the Spartan. The number of Slice-Register of Virtex-6 and the number of Slice-Flip-Flop were used to represent the overall hardware resource usage. There are some relationship of Slice and LUT dependent on the hardware technology. There were 32-bit between the original floating point operator block and output as shown in Table 4 - Table 6 but it does not show Virtex-6 table as index of "IOB Flip Flops". This number was used to represent the circuit that has output registers. The last calculation block is byte-float operator as block of Mul012 the synthesized result for Virtex-6 show as Table 10.

Table 10 The hardware utilization of a block of Mul012 operator for Virtex-6

Mul012 Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	34	93120	0%
Number of Slice LUTs	52	46560	0%
Number of fully used LUT-FF pairs	34	52	65%
Number of bonded IOBs	72	360	20%
Number of BUFG/BUFGCTRLs	1	32	3%
Maximum Frequency	778.028MHz		

The sub-calculation circuit such as updataP updateQ and updateAB composed of the calculation block from Table 1. The circuit for updateQ was composed from K blocks of Adder and Divider, K blocks of temporary registers and K blocks of Single Pulsers. From the synthesized result, the Adder block uses 590 slices of FPGA register and the Divider block uses 1360 slices of FPGA register. Therefore the circuit of updateQ approximately uses 2K×(590)+K×(1360) and K×32 of I/O Flip-Flops as a temporary registers and K blocks of Single Pulser. This is very small amount of FPGA resource. Assuming K = 3 so the hardware resource is 7620 +xx slice registers (from 3×2×590 + 33×1360+xx). xx is from 3 blocks of Single Pulser and I/O Flip-Flops. The synthesized result was shown in Table 11 that is corresponded with the relationship of Table 1.

Table 11 The resource utilization of updateQ circuit for Virtex-6

UpdateQ Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	7,926	93,120	8%
Number used as Flip Flops	7,926		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of bonded IOBs	330	360	91%
Maximum Frequency	625.900MHz		

There were three types of updateP. The aim of different type was for resource saving. The first type was designed for the first individual. The operation of this type is to multiply the value of latent variables by value of genetic of each ancestry groups and transmits the result to updateP. This circuit was called “headUpdateP”. This circuit composed of 2K blocks of mul012. The resource usage of this circuit is 2K×(35) slice registers. The K value was set to 3, as above,

so the resource usage was 210 slices registers. The second type of updateP was called “bodyUpdateP”. This circuit was designed to sum the result of its own genetic value and latent variables with the result from headUpdateQ. The circuit uses 2K blocks of Mul012, for multiplying of genetic value and the value of latent variables, and 2K blocks of adder, to add the mul012 result with the above calculation circuit. The resource consumption is 2K×(590 + 35) slices registers. The setting was K = 3 so the resource usage was 800 slices registers. The final circuit of updateP is circuit “tailUpdateP”. The operation of this circuit is the same as the bodyUpdateP circuit but this circuit divides the result of G_{im}×A_{imk} by (G_{im}×A_{imk}+ G_{im}×B_{imk}). K blocks of adder and divider were added to the circuit. The circuit uses 3930 slices registers, from 800+3 × (590+1360). The third circuit of updateP (buttUpdateP) was placed in the end of the calculation circuit. The adder block and divider block were added into the circuit to normalize the cumulative sum. The K blocks of adder and divider were used. The hardware resource was obtained from the circuit of bodyUpdateP + K(Adder+Divider). The hardware resource usage of three ancestral population was 9780, 3930 + 3(590+1360) of slices registers. The synthesized result of hardware utilization was shown in Table 12

The last circuit is updateAB. Every calculation units can directly access to the register unit of admixture ratio and the ancestral allele frequency. The circuit uses 2K blocks of multiplier to multiply the individual admix ratio by the ancestry allele frequency and 2(K-1) blocks of serial adder for the denominator of the divider block and the added result of Q_{ik}×P_{km} be a numerator of the upper latent variables and the added result of Q_{ik}×(1-P_{km}) is a numerator of the lower latent variables as Eq 11 and Eq 12. The hardware resource usage of the circuit is 2K×(700+1360) + 2(K-1)×(590). The number of ancestral groups was set to 3, so the resource utilization is 14720 slices of registers. The synthesized result for circuit of updateAB was shown in Table 13.

Table 12 The hardware resource utilization for each type of updateP circuit for Virtex-6

headerUpdateP Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	204	93120	0%
Number of Slice LUTs	126	46560	0%
Number of fully used LUT-FF pairs	60	270	22%
Number of bonded IOBs	396	360	110%
Number of BUFG/BUFGCTRLs	1	32	3%
Maximum Frequency	625.900MHz		
bodyUpdateP Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	3948	93120	4%
Number of Slice LUTs	3654	46560	7%
Number of fully used LUT-FF pairs	2358	5244	44%
Number of bonded IOBs	594	360	165%
Number of BUFG/BUFGCTRLs	7	32	21%
Maximum Frequency	415.870MHz		
buttUpdateP Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	9993	93120	10%
Number of Slice LUTs	7932	46560	17%
Number of fully used LUT-FF pairs	5568	12357	45%
Number of bonded IOBs	495	360	137%
Number of BUFG/BUFGCTRLs	13	32	40%
Maximum Frequency	415.870MHz		

Table 13 The hardware resource utilization of updateAB circuit for Virtex-6

updateAB Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	15258	93120	16%
Number of Slice LUTs	11408	46560	24%
Number of fully used LUT-FF pairs	9062	17604	51%
Number of bonded IOBs	489	360	135%
Number of BUFG/BUFGCTRLs	16	32	50%
Maximum Frequency	650.745MHz		

The synthesized results of Table 11 - Table 13 are correlated with the calculation blocks usage of Table 1 for the number of calculation unit and Table 4 - Table 6 for hardware resource usage. The size of hardware circuit can be calculated to determine the sufficient size of FPGA.

The experimental result is the synthesized circuit of 5 individual, 10 markers, and 3 ancestral groups. The circuit consists of 1 circuit of updatePQ header, 2 circuits of updatePQ body, 1 circuit of CalPQ last, 5 circuits of updateAB and controlAll circuit. The first 5 circuits are composed of rotate-right registers for latent variables for 10 markers and 3 ancestral groups. The second part is 5 circuits for genotype rotate-right registers. The third part is 5 circuits of 3 groups admixture units. The last part is 1 circuit of 10 markers and 3 groups of ancestral allele frequencies. There are also miscellaneous components such as several blocks of single pluser and AND gates. The synthesized result can be calculated by sum up the hardware utilization result of Table 4 - Table 13.

Table 14 The hardware resource utilization of Frappe calculation circuit

Frappe I5M10K3 Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	143031	-	-
Number of Slice LUTs	110927	-	-
Number of fully used LUT-FF pairs	84279	-	-
Number of bonded IOBs	-	-	-
Number of BUFG/BUFGCTRLs	16	-	-

The hardware utilization resource can be calculated by directly add up the number of each usage as Table 15 and Table 16

Table 15 The hardware usage for the calculation circuit that composes of 5 individuals 10 markers and 3 ancestral groups

	updatePL	updatePB	updatePH	updateQ	updateAB	Total
#slices	9993	3948	204	7,926	15258	
#block	1	2	1	5	5	
	9993	7896	204	39630	76290	124020

Table 16 The hardware usages for the register circuit that composes of 5 individuals 10 markers and 3 ancestral groups

	AB	G	P	Q	total
#bit	32	2	32	32	
Marker	10	10	10	1	
individual	10	5	2	5	
ancestral	3	3	3	3	
	9600	300	1920	480	12300

4.3 Calculation result

The calculation result of the circuit depends on the executed environment. The software version was re-implemented from the original FRAPPE's application. The floating point single precision and floating point double precision were re-implemented and compared to the original version. Please note that the original version implements the floating point single precision format. The results show the slightly different accuracy of single precision and double precision. The result is compared with to the re-implemented software version. Figure 31-Figure 33 show the collocation of single precision result and the result from original version. The results were plotted in the 100% stack to represent the composite of each ancestry group in an individual. The Y-axis denotes an individual. The X-axis denotes the mixing ratio. The results were compared by root mean squared error (rmse) as $\sqrt[2]{\frac{1}{I}\sum_{i=0}^{I}(qo_i - qr_i)}$. qo_i means the admix ratio of the original application of the i^{th} individual. qr_i means the admix ratio of the re-implemented application of the i^{th} individual. The rmse is small around .05 compared to floating point single precision between the original FRAPPE application and re-implemented application.

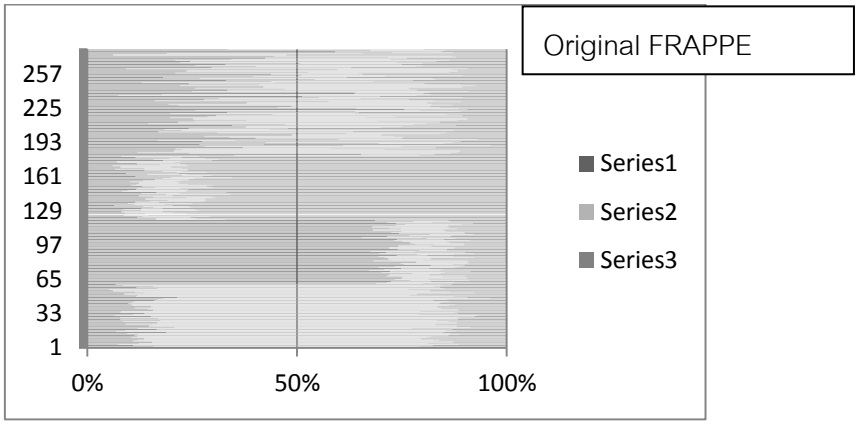

Figure 31 The result of admix ratio from the simulation data with original FRAPPE application

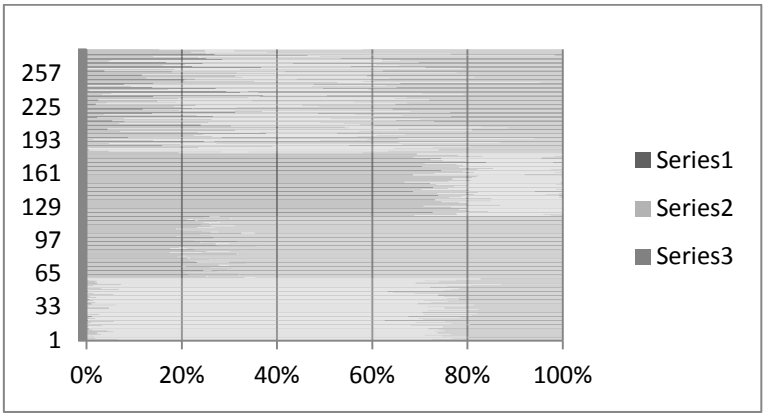

Figure 32 The result of admix ration from the simulation data with double precision re-implemented application

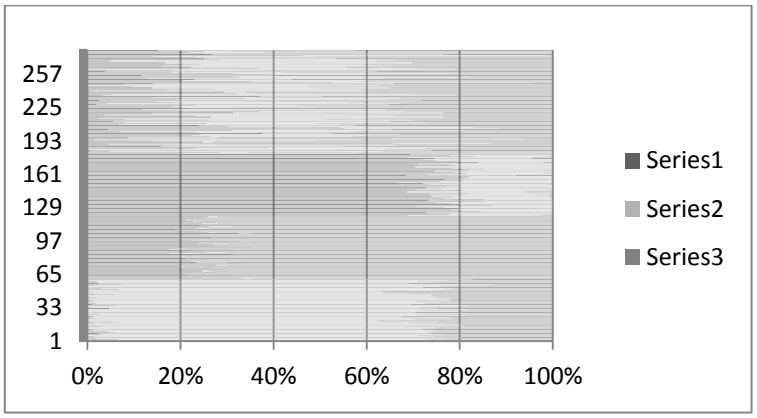

Figure 33 The result of admix ration from the simulation data with single precision re-implemented application

The large size of dataset cannot be implemented in the real circuit using Spartan3 xcs4000. The size of data set was reduced into 5 individuals, 10 markers and 3 ancestral groups.

4.4 Results

The synthesis result was obtained from Xilinx Design Suite [46] and the calculation result was obtained from ISim[48]. There were many levels of simulation, Behavioral, Post-Translate, Post-Map and Post-Route. The Post-Route simulation was selected with this dissertation because the gate delay profile with routing profile were included in the simulation result that makes the timing result reliable. And the table and graph above denote that the hardware circuit can gain more efficient way to apply to this problem especially for Multiple Instructions Multiple Data (MIMD) platform. Figure 32 shows the result precision from the hardware calculation has no significantly difference from the software version. The result of the calculation speed up was

4.5 times for spartan3 architecture compare to single core processing unit and 2.2 times for spartan3 compare to multi-cores processor. The detail of speed up for overall circuit calculation can be described as 4.5

4.5 Hardware and Software performance

The performance comparisons were compared between FPGA VS single core processor FPGA vs multi-core processor, GPU was also included to multi-core processor.

4.5.1 Single Core

The proposed method of calculation was implemented via single core using one thread of calculation. There were three loops of calculation, the first loop performs the calculation of the admix ratio from each the first row through the last row as Figure 34

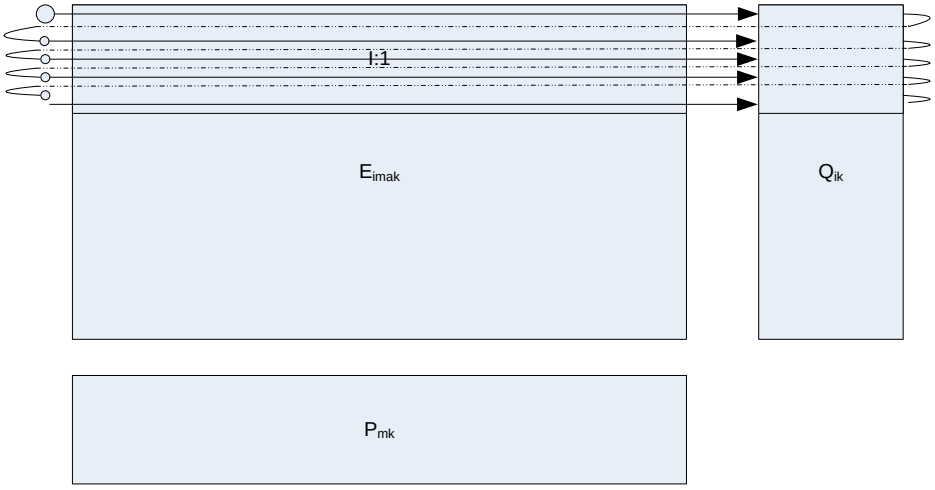

Figure 34 the calculation of updateQ for single core processor.

The second loop performs the calculation of the ancestral allele frequency as Figure 35. The loop starts from the first column through the last column. The last loop updates the latent variables. The loop can be calculated either by row-wise or column- wise, because there are no dependency from the calculation the processing unit reads values from the specified group of admix ratio and ancestry allele frequency to update the specified latent variables as Figure 36. The row-wise or column-wise are used to specify the direction of calculation. The single core implementation takes I*M*K memory accesses for every step.

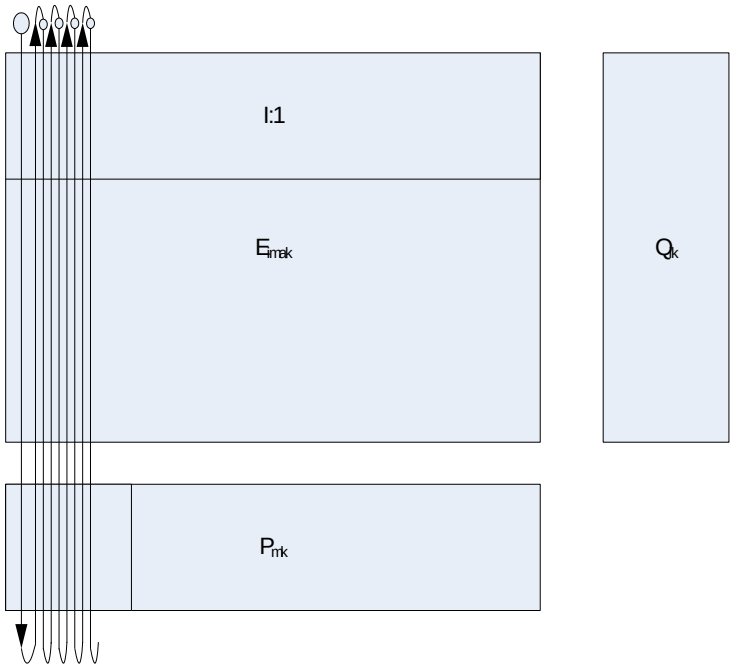

Figure 35 the column-wise architecture, the calculation loop of updateP for a single core processor

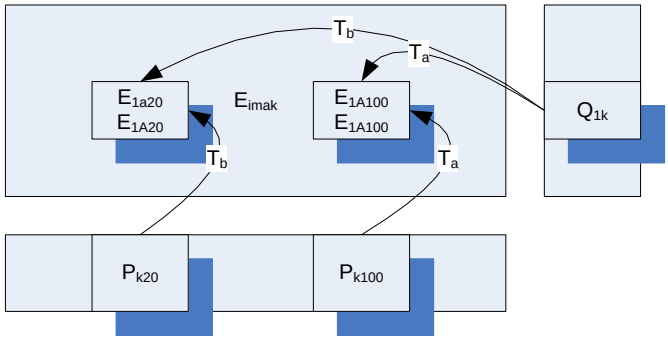

Figure 36 the calculation of updateAB for a single cores processor

4.5.2 Multi-Core

The next implementation was transferred from single thread to be multiple threads (number of threads denoted as T). To calculate the admix ratio can simultaneous calculates T different rows at the same time as The speed up gains from loop unrolling technique.

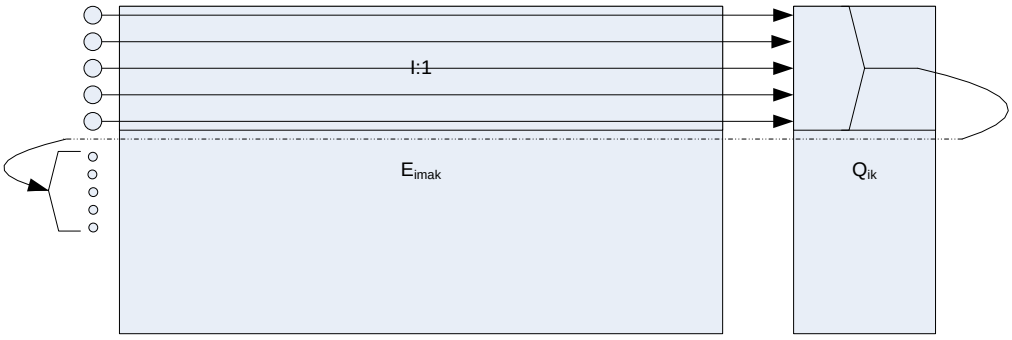

Figure 37 the operation of updateQ for multi-processing unit.

In Figure 37, the big circle denotes the operating threads and the small circle denotes the waiting job in the queue. The method was the same as multi-processing on the single core. The processing time of each thread takes M steps of calculation, there are I*K works to be done and T calculation units. The work for calculation of admix ratio is I*K/T. The overall time is I*M*K/T. Figure1115 represents the method of the ancestral allele frequency calculation. Row-wise was applied with the ancestry allele frequency calculation. The calculation time of each thread is I*K, there are M/T works in a waiting queue. The overall of calculation times is I*M*K/T same as updateQ. Finally the Expectation calculation to update the latent variable is updateAB. Each latent variable was calculated by directly access from P and Q at the specified k as Figure1116. Each unit of calculation takes K steps of calculation, there are I*M/T blocks of calculation. The overall of calculation time of updateAB is I*M*K/T, same as the two methods above. The major obstruction of speed up is not as I*M*K/T. There are /three major causes. The first is memory overhead. The second cause is insufficient floating point operation unit. Memory overhead came from accessing to memory unit especially the array data-type. In the data unit (Figure in Chapter 2) memory unit was declared as array. The loop of individual admixture can be implemented with no temporary register in the cumulative summation of AiK+Bik. The result is written to array Qik M times that can slow down the index calculation of array. The loop unrolling method assigns each row of data to each thread of calculation as Figure 37. The method can reduce the number of memory accessing of Qik from M times to only one. The task of updateP is very similar to the updateQ but the access is row-wise. The updateAB calculation assigned each thread of calculation to each cell as Figure 36. The calculation causes large amount of memory accessing both Qik and Pkm, M times of memory Qik accessing per a row and I*K times of array accessing of Pkm. Column-wise access was applied to updateAB

calculation. It takes time of calculation to I*M*K/T but the number of memory of Qik access can be reduced from M to be 1 while Pmk remained the same. Row-wise access can be applied for the calculation. This method also takes time of calculation to I*M*K/T. It reduces the memory accessing of Pmk from I to be 1 but the number of acces of Qik is M. Most of SNPs data has more number of markers than the number of individual, so column-wise access was applied for the calculation of updateAB. The floating point units of multi-cores processors are special processing units for general processors. It requires special data format. The special hardware is not scalable by the number of thread in a CPU. Finally the last task has different data type of the calculation of update, Gim*Pmk. Gim can either be 0, 1 or 2 bytes data. It was implemented for genotype data to save resource. The calculation of Pmk is a single precision floating point data. It has more precision than fixed point data format. The operator of Gim*Pkm was slowed down by data casting from byte into float.

The proposed technique above were implemented to compare the speed of GPU, single-core CPU and multi-core CPU and GPU, three versions of the calculation were implemented. The implementation of multi-core version gained highest performance on quad core when compared to the implementation of GPU (Nvidia 8600GT. The major barrier of GPU may come from memory synchronization between CPU memory and GPU memory. The thread of calculation of GPU is not directly uncontrollable by the programmer.

4.5.3 FPGA

Using FPGA can overcome these limitations. The proposed circuit was motivated from software implementation. The limitation of number threads can be avoid using the systolic architecture because of the systolic architecture can be scalable using FPGA. The calculation was designed for scalability by using sub-common blocks of calculation to simultaneously compute each row of each calculation unit together or each column of each unit or each cell of each unit. The calculation unit can be placed in several ways such as: row-wise, column-wise or strip-wise and bundle-wise. The calculation of updateP and updateQ cause a problem of ripple carry of floating point data format. The ripple carry in the cumulative summation came from the first individual through the last individual, for updateP calculation, and in the cumulative summary from the first marker through the last marker for calculation of updateQ. The bundle-wise access cannot gain much speed up. The calculation of updateAB can gain speed up from

I×M×K to K for the bundle-wise access. The calculation of updateAB has no dependency between units of calculation; every unit of calculation can directly read from memory unit of P and memory unit of Q as Figure 36. Therefore, the calculation of expectation reduced the time from I×M×K to K for bundle-wise, but the hardware becomes too large. The row-wise and column-wise accesses are preferred choice. The ripple carry in each column was obtained from a cumulative sum of Eq 9. The ripple carry from each row was also obtained from Eq 10. Row-wise architecture for calculation updateP was shown in Figure 38.

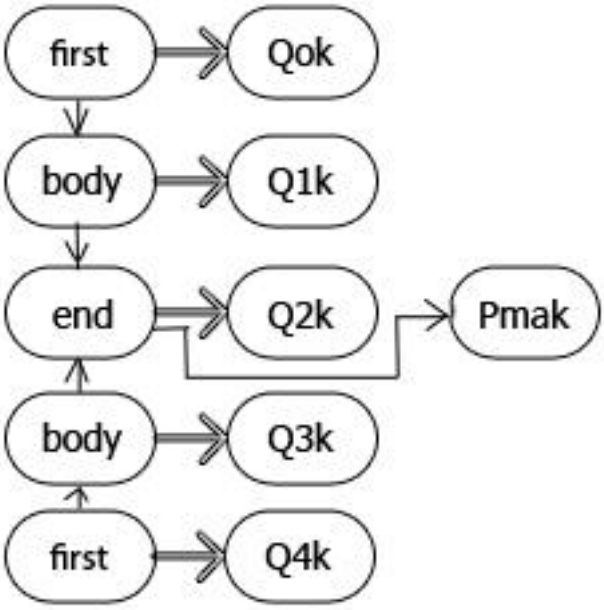

Figure 38 the placed calculation circuits and register units of row-wise access

Figure 38 shows the direction of calculation. The architecture reduces the time of calculation from I×K to I×K/2 for one marker. The data were fed into the calculation unit M times. The total time of calculation is M×I×K/2. Figure 39 shows the column-wise access. The architecture takes time of calculation M/2 for a row. There are I*K rows of data. Total time of calculation for column-wise access is the same as row-wise. Data type casting also causes a slow down. The updateP has I*M*K times of byte-float multiplier, to multiply Gim with the latent variables. The byte operand will automatically be casted to float before multiply. The operator of float-float multiplication is too slow. This dissertation proposed a byte-float multiplier circuit that takes only 3 clocks.

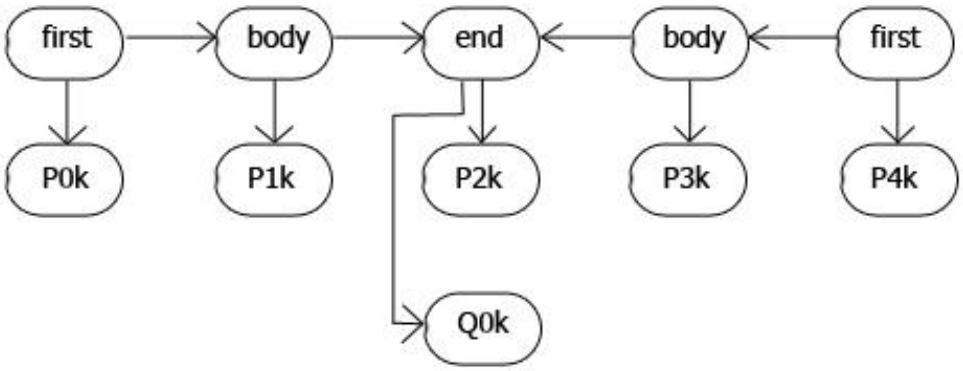

Figure 39 the calculation block connection for column-wise access

In theory, the calculation takes the same time for both row-wise and column-wise access. The dependencies of the calculation are different for row-wise and column-wise from the direction of each calculation, but the overall calculation time is the same. The major difference is the number of sub-calculation in the circuit. Most of genomic data has a lot more marker than the number of individual. The row-wise access results in smaller circuits. The calculation of updateAB for bundle-wise or full placement takes just K times of calculation. The time complexity is the same for software calculation. The main benefit of hardware calculation circuit is that every calculation unit can access directly to the memory unit for reading a data while the software can not read a common memory at the same time. The index calculation was also eliminated because the placement of register unit and calculation unit as Figure 15. Every calculation unit can access to their connected memory unit.

The circuit has no memory overhead, it can simultaneously start the process of calculation that every calculation threads is controllable. There were some specialist said that “For CUDA, you have to twist and turn your algorithm in very specific ways to enjoy the speed up. With FPGA you can do whatever you want i.e. implement specialize computation routines tailored just for your algorithm. CUDA -> SIMD, memory coalesced. Reducing a communication of main memory and devices memory is the main paradigm of the speed up.”[50]. The sentence is true for multi-cores memory system. Tuning up the speed of calculation in multi-cores is very difficult because assignment of multithreads to the cores are not directly controllable by programmers.

4.6 Future works

The admixture calculation was developed in several ways. The model of calculation has been developed for a long time from 1991. The calculation technique was proposed. The calculation based on iteration methods required large memory and long calculation time. There are several approaches of Newton-Raphson method such as, Quasi-Newton-Rahpson, Jacobian free Newton-Method. The algorithms of Newton-Raphson methods guarantee the quadratic of convergence but the assumption was based on an appropriate starting point. The speed of convergence was very good. The approach of Newton-Raphson method cannot be applied directly to the admixture model because there are both equality and inequality constraints in the problem. The newton's method has a good rate of convergence but there are no known solution for global maximum of the admixture calculation. The evolutionary computing may be a choice to solve the global maximum but the cost of fitness evaluation is high.

For the point of view of circuit design, the proposed circuit can speed up the calculation by streaming the input to sub-unit (systolic architecture) but the new buffer is needed for a queue in the wrapping circuit. The sub-unit was designed for scalability. The system can contain more than one FPGA board and used handshaking signals to communicate. The current design allocates data in the registers. Using an external memory to hold variable size of data is also possible. The design is flexible because it is independent of the data format. The floating point data format can be change from 32-bit to another format (but 32-bit). For higher precision the size of mantissa can be increased and the size of exponent part reduced. FPGA is applicable to speed up floating point calculation in several ways. There are several dissertations that used FPGA for CPU co-processing which gain speed up from this approach.

There are numerous options in the configuration of ISE design suite to optimize a synthesized circuit to suite different user requirements such as, optimize for area or optimize for speed. The different options may result in different types of circuit.

REFERENCE

[1] Jeffery C. Long: The Genetic Structure of Admixed Population. Genetics1991, 127:417-428

[2] Nick Patterson, Neil Hattangadi, Barton Lane, Kirk E. Lohmueller, David A. Hafler, Jorge R. Oksenberg,Stephen L. Hauser, Michael W. Smith, Stephen J. O'Brien, David Altshuler, Mark J. Daly, and David Reich1,(2004)" Methods for High-Density Admixture Mapping of Disease Genes", 2004 by The American Society of Human Genetics

[3] C. L. Hanis, R. Chakraborty, R. E. Ferrell, W. J. Schull (1986)," Individual admixture estimates: disease associations and individual risk of diabetes and gallbladder disease among Mexican-Americans in Starr County, Texas.", Am J PhysAnthropol, Vol. 70, No. 4. (August 1986), pp. 433-441

[4] Smith MW, Patterson N, Lautenberger JA, Truelove AL, McDonald GJ, Waliszewska A, Kessing BD, Malasky MJ, Scafe C, Le E, De Jager PL, Mignault AA, Yi Z, De The G, Essex M, Sankale JL, Moore JH, Poku K, Phair JP, Goedert JJ, Vlahov D, Williams SM, Tishkoff SA, Winkler CA, De La Vega FM, Woodage T, Sninsky JJ, Hafler DA, Altshuler D, Gilbert DA, O'Brien SJ, Reich D.,(2004)," A high-density admixture map for disease gene discovery in african americans.", Am J Hum Genet. 2004 May;74(5):1001-13. Epub 2004 Apr 14.

[5] JacekMajewski, PiotrZawadzki, Paul Pickerill, Frederick M. Cohan and Christopher G. Dowson. Barriers to Genetic Exchange between Bacterial Species: Streptococcus pneumonia Transformation, J. Bacteriol. 2000, 182(4):1016

[6] Hua Tang, JiePeng, Pei Wang and Neil J. Risch :Estimation of Individual Admixture: Analytical and Study Design Considerations. Genetic Epidemiology 2005, 28: 289–301

[7] Alkes L. Price, Nick Patterson, Fuli Yu, David R. Cox, AlicjaWaliszewska, Gavin J. McDonald, ArtiTandon, Christine Schirmer, Julie Neubauer, Gabriel Bedoya, Constanza Duque, Alberto Villegas, Maria CatiraBortolini, Francisco M. Salzano, Carla Gallo, Guido Mazzotti, Marcela Tello-Ruiz, Laura Riba, Carlos A. Aguilar-Salinas, Samuel Canizales-Quinteros, Marta Menjivar, William Klitz, Brian

Henderson, Christopher A. Haiman, Cheryl Winkler, Teresa Tusie-Luna, Andrea Ruiz-Linares, and David Reich,(2007)," A Genomewide Admixture Map for Latino Populations", The American Journal of Human Genetics Volume 80 June 2007

[8] R. et al. Sachidanandam. "A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms". Nature, 2001

[9] Carolina Bonilla, Mark D. Shriver, Esteban J. Parra, Alfredo Jones, Jose R. Fernandez (2004) "Ancestral proportions and their association with skin pigmentation and bone mineral density in Puerto Rican women from New York city", 30 April 2004, Hum Genet 57-68

[10] Hanis CL, Chakraborty R, Ferrell RE, Schull WJ: Individual admixture estimates: disease associations and individual risk of diabetes and gallbladder disease among Mexican-Americans in Starr County, Texas. Am J PhysAnthropol. 1986 Aug;70(4):433-41.

[11] A. P. Dempster, N. M. Laird, D. B. Rubin: Maximum Likelihood from Incomplete

[12] Data via the EM algorithm. JSTOR Series B 1977, 39: 1-38

[13] Christopher M. Bishop ," PATTERN RECOGNITION AND MACHINE LEARNING: Mixture model and EM", p.423-454

[14] http://en.wikipedia.org/wiki/Newton's_method

David H. Alexander, John Novembre, Kenneth Lange: Fast Model-Based Estimation of Ancestry in Unrelated Individuals. Genome Res. 2009, 19(9): 1655-1664

[15] Stephanie Zierke and Jason D Bakos: "FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods", BMC Bioinformatics 2010, 11:184

[16] Peter J. Green and Desmond P. Taylor: FPGA Implementation of a Real Time Maximum Likelihood Space-Time Decoder on a MIMO Software Radio Test Platform. IEEE Symposium on Electronic Design, Test & Application 2010.

[17] Keith Underwood: FPGA vs. CPUs: Trends in Peak Floating-Point Performance. FPGA'04 2004 Feb; 22-24

[18] Florent de Dinechin. The arithmetic operators you will never see in a microprocessor. IEEE Symposium on Computer Arithmetic 20th 2011

[19] Cristian Grozea, Zorana Bankovic, Pavel Laskov. FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application. Lecture Notes in Computer Science Volume 6310, 2010, pp 105-117.

[20] Satoshi Sukuki, Hirotsugu Kajisaki, Keisuke Iwai, Takakazu Kurokawa. CAM based hardware implementation of IDS. 2nd European Conference on Information Warefare and Security. 2003, PP317-323

[21] FPGA: http://www.xilinx.com/products/silicon-devices/fpga/index.htm, July 26, 2012

[22] CUDA: http://developer.nvidia.com/category/zone/cuda-zone. 2012

[23] ATI Stream: http://developer.amd.com/archive/gpu/ATIStreamSDKv1.4Beta/Pages/default.aspx. 2012

[24] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2. 2012

[25] http://en.wikipedia.org/wiki/F-statistics

[26] Carrie Lynn Pfaff 1, Jill Barnholtz-Sloan 2, Jennifer K. Wagner 1, Jeffrey C. Long ,(1 December 2003), "Information on ancestry from genetic markers", Genetic Epidemiology Volume 26 Issue 4, Pages 305 – 315

[27] Pritchard JK, Stephens M, Donnelly P: Inference of population structure using multilocus genotype data. Genetics 2000, 155:945-959

[28] LounesChikhi,* Michael W.Bruford* and Mark A.Beaumont," Estimation of Admixture Proportions: A Likelihood-Based Approach Using Markov Chain Monte Carlo", April24,2001

[29] Daniel Falush, Matthew Stephens, Jonathan K. Pritchard: Inference of Population Structure Using Multilocus Genotype Data: Linked Loci and Correlated Allele Frequencies. Genetics 2003, 164:1567-1587

[30] Christopher M. Bishop: Mixture Models and The EM Algorithm, Advance Tutorial Lecture Series, CUED, 2006

[31] Crow, Jf (Jul 1999). "Hardy, Weinberg and language impediments." Genetics 152 (3)pp. 821-825

[32] Guo, Sw; Thompson, Ea (Jun 1992). "Performing the exact test of Hardy-Weinberg proportion for multiple alleles.". Biometrics 48 (2): 361-372.

[33] Hardy, Gh (Jul 1908). "MENDELIAN PROPORTIONS IN A MIXED POPULATION.". Science (New York, N.Y.) 28 (706): 49--50.

[34] Allele, http://en.wikipedia.org/wiki/Allele

[35] SNPs, http://en.wikipedia.org/wiki/Single-nucleotide_polymorphism

[36] Laurent ExcofieP and Montgomery Slatkin,(1995)," Maximum-Likelihood Estimation of Molecular Haplotype Frequencies in a Diploid Population", Mol. Biol. Evol. 12(5):921-927. 1995.

[37] Binomial Distribution Model, http://en.wikipedia.org/wiki/Binomial_distribution

[38] In Jae Myung: Tutorial on maximum likelihood estimation. Journal of Mathematical Psychology, October 16, 2002

[39] C.T. Kelly. Solving Nonlinear Equations with Newton's Method, SIAM 2003

[40] Jason HandUber: Systolic Arrays, Feb 12, 2003

[41] http://en.wikipedia.org/wiki/Systolic_array

[42] Jop Frederik Sibeyn, http://users.informatik.uni-halle.de/~jopsi/dpar03/chap3.shtml

[43] Xilinx: Spartan-3 Libraries Guide for HDL Designs. UG607(v 13.1), March1, 2011

[44] Xilinx: Spartan-3 Libraries Guide for Schematic Design. UG608(v13.4), Jan 18, 2012

[45] IEEE754 Floating-Point format: http://en.wikipedia.org/wiki/IEEE_floating_point, July 26, 2012

[46] ISE Design Suite. http://www.xilinx.com/ise_eval/index.htm

[47] Floating-Point operator: LogiCORE IP Floating-Point Operator v5.0. DS335, March 1, 2011

[48] ISim, ISE Simulator: http://www.xilinx.com/tools/isim.htm. July 26, 2012

[49] Alexander Heinecke, Michael Klemm, Hans-Joachim Bungartz: From GPGUP to Many-Core: Nvidia Fermi and Intel Many Integrated Core Architecture. Computing in Science & Engineering 2012, 78-82

[50] CUDA-vs-FPGA, http://stackoverflow.com/questions/317731/cuda-vs-fpga, Nov 2008

Biography

Alongkot Burutarchanai was born in Bangkok, Thailand, on January, 1982. He received B.Eng. and M.Eng., both in computer engineering from Chulalongkorn University, Thailand, in 2003 and 2006, respectively. He had done his senior project and Master degree with Prof. Prabhas Chongstitvatana. His projects were related to hardware and embedded systems. All of his projects were implemented with FPGA. In the second year of his Ph.D., he joined a biomedical research group with Dr. Anunchai Assawamakin at Biotec, Thailand. Dr. Sissades Tongsima is the head of Biotec laboratory. Dr. Sissades and Dr. Anunchai gave him a problem in biology that became his Ph.D. research topic. Finally he learnt a lot of techniques from Dr. Siroj Sirisup and Evolutionary techniques from his advisor and a lot of FPGA implementation techniques from Prof. Tomohiro Yoneda to complete his Ph.D. research topic.