

Design and Implementation of a cloud based FPGA accelerator for phylogeny reconstruction



ANASTASIOS BOKALIDIS

Electrical & Computer
Engineering School Technical
University of Crete

Table Of Contents

- ▶ Introduction
- ▶ Theoretical Background
- ▶ Related Work
- ▶ Maximum Likelihood Method
- ▶ Architecture Design
- ▶ FPGA Implementation
- ▶ Results
- ▶ Conclusions

Introduction

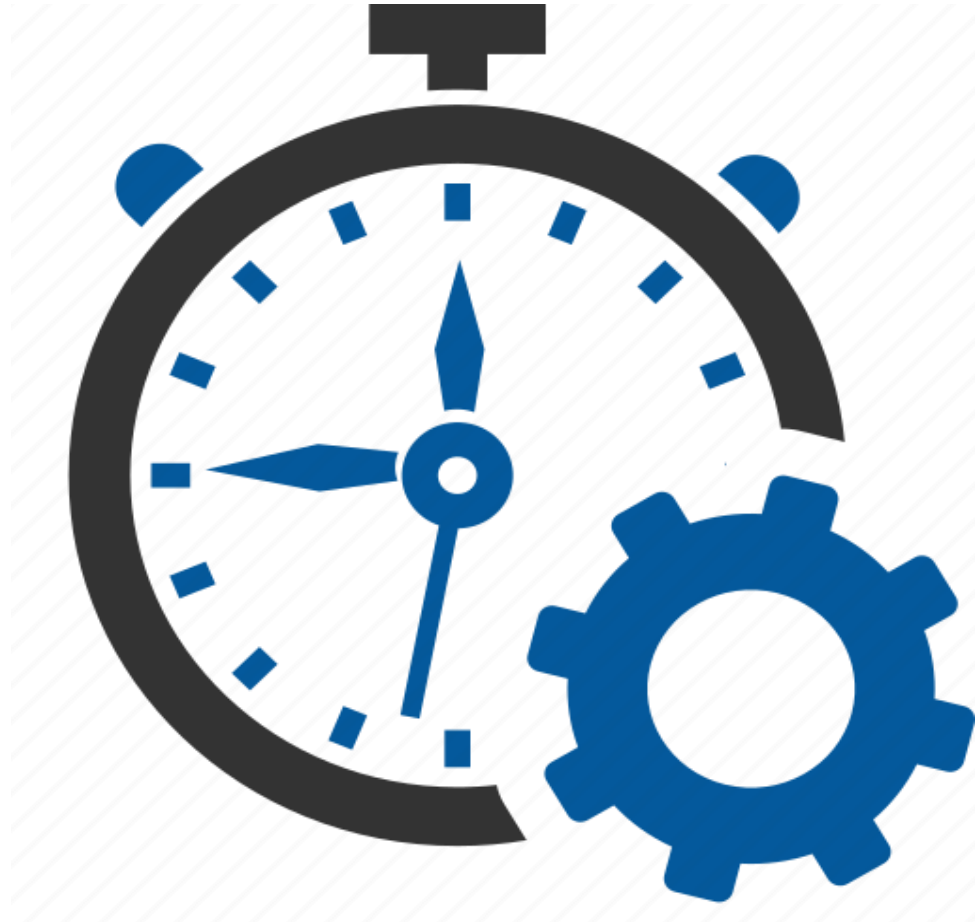
Introduction



Introduction

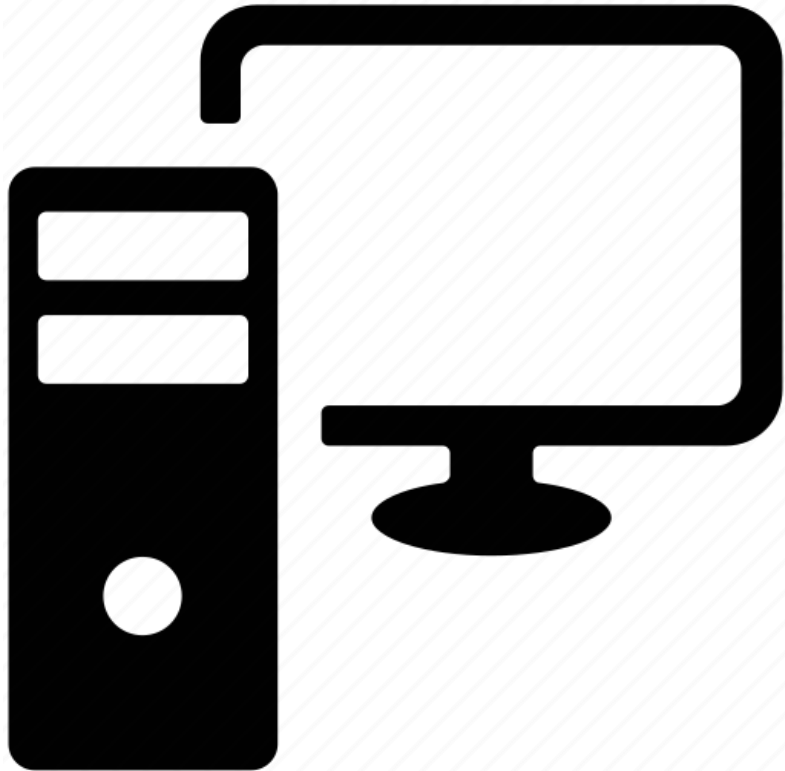


Introduction



Introduction

PERSONAL COMPUTERS

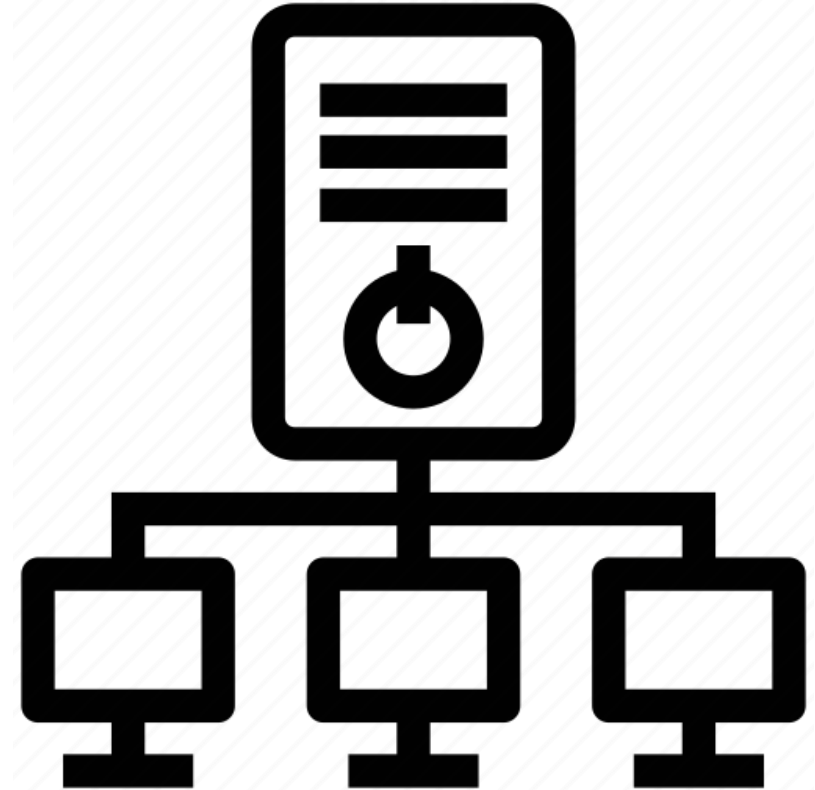


Introduction

PERSONAL COMPUTERS



SUPER COMPUTERS

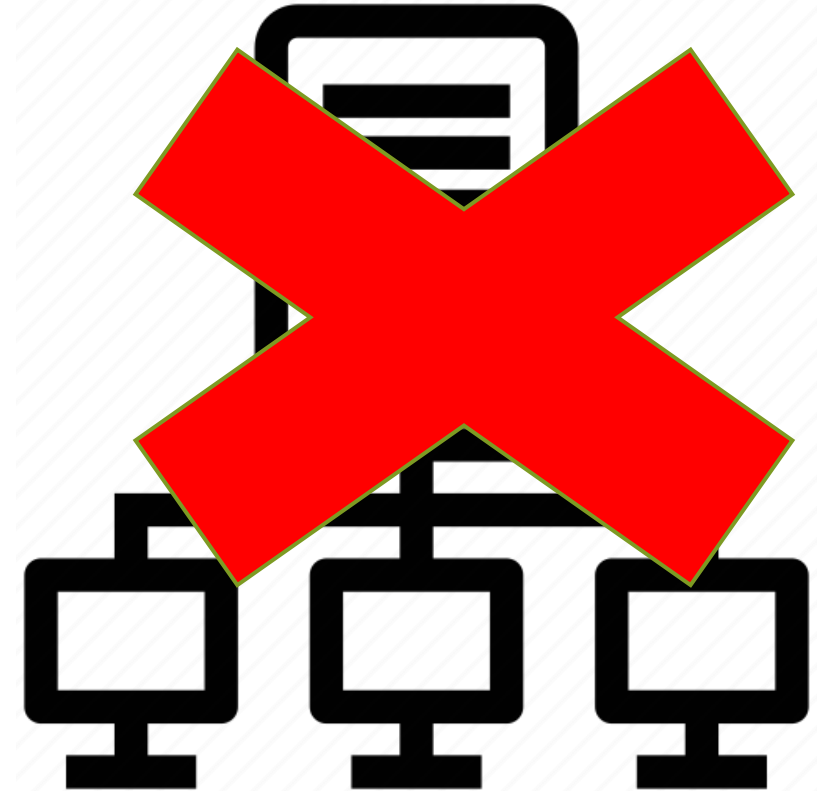


Introduction

PERSONAL COMPUTERS

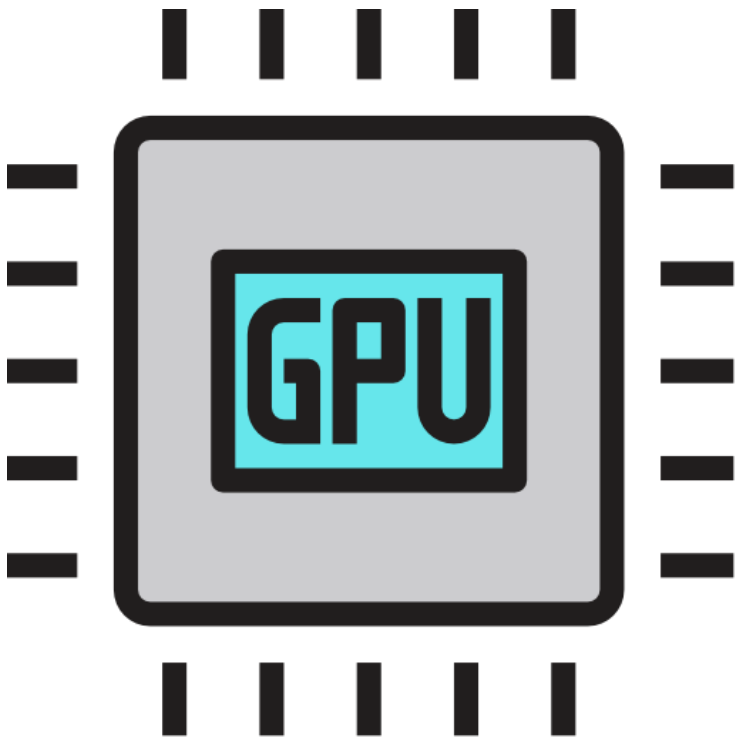


SUPER COMPUTERS



Introduction

GPUs



FPGAs



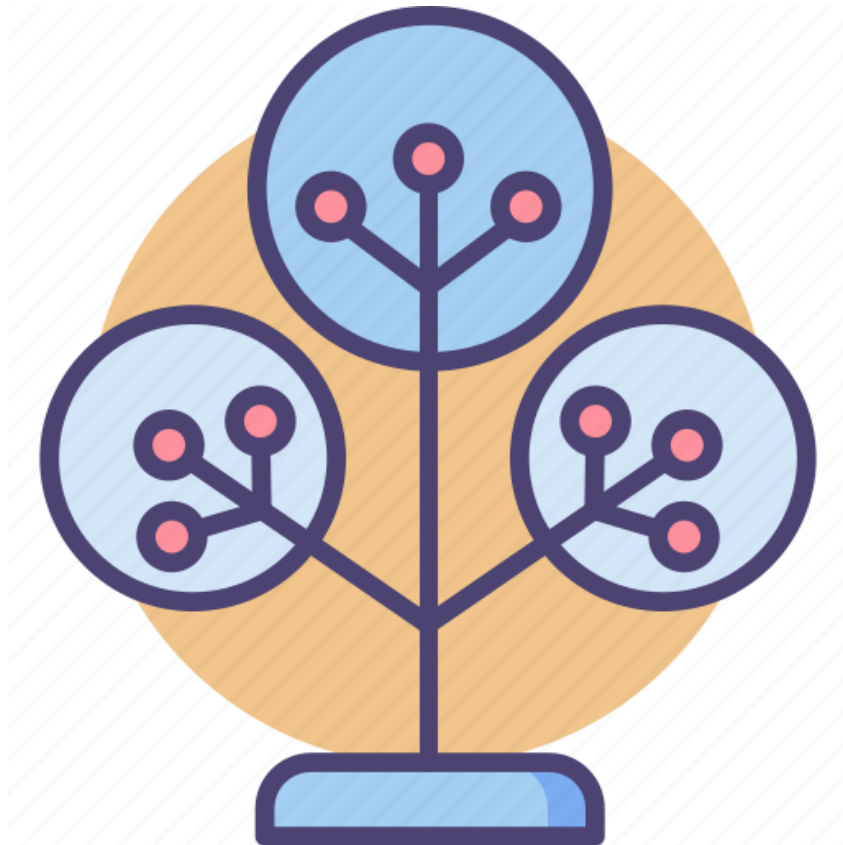
Contribution

- Find a solution to a problem that burdens the surveys on bioinformatics and specific on phylogeny.
- Describe some possible accelerators of time-consuming functions.
- Design and construct units that implement the requisite computations of the accelerator.
- Distribute and manage the available resource in order to surpass a simple CPU's speed.
- Propose hardware platforms that can load and execute effectively the demands of the accelerator.

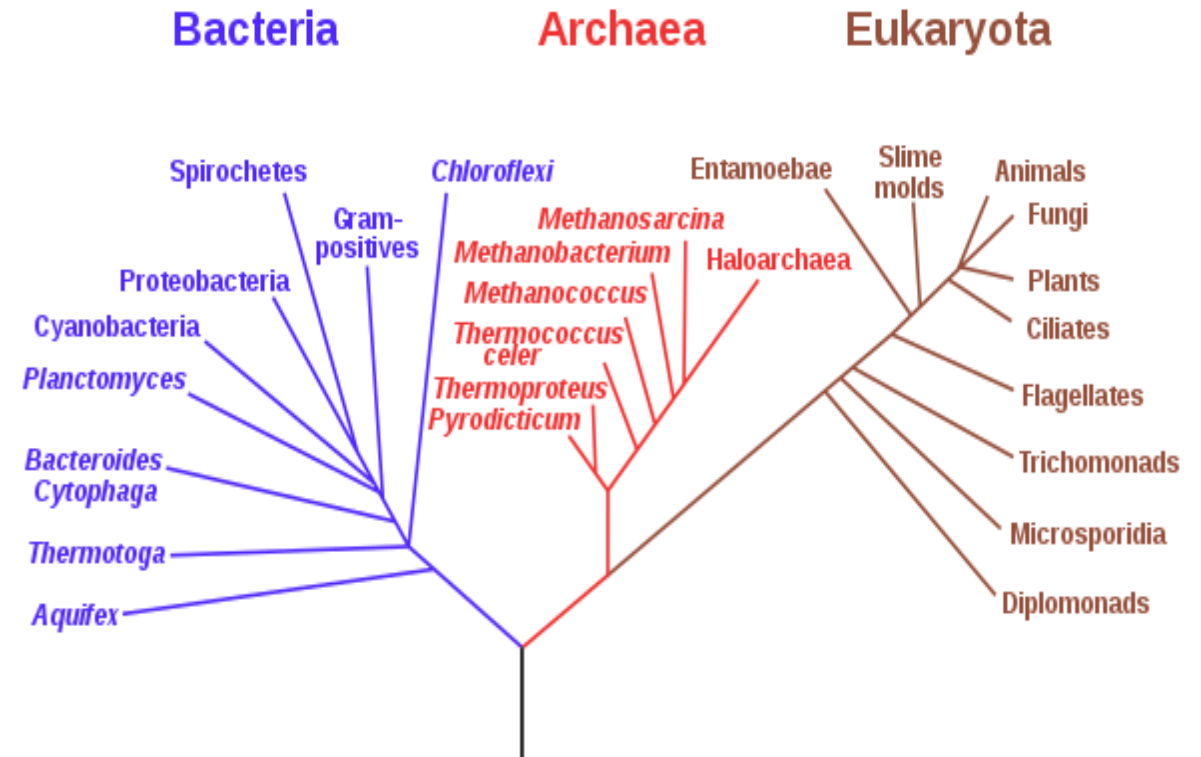
Theoretical Background

Phylogenetics-Phylogenetic Tree

Phylogenetics

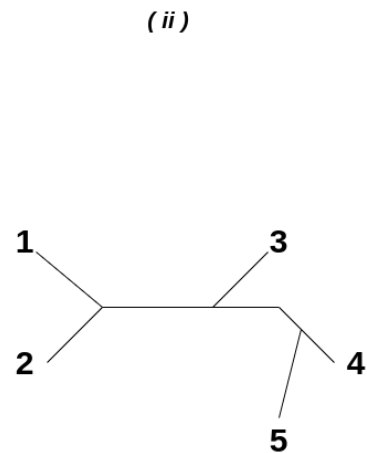
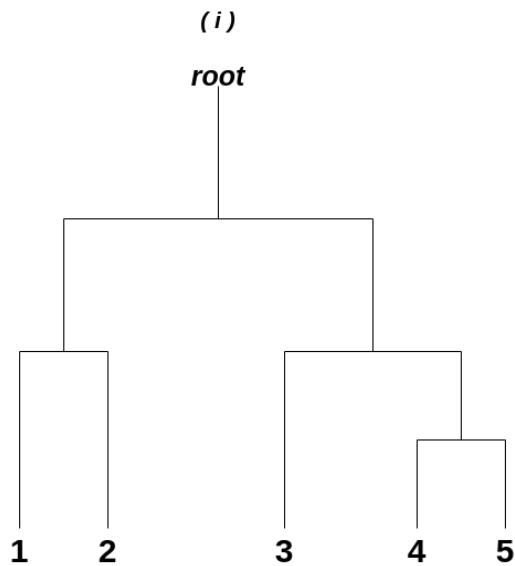


Phylogenetic Tree



Phylogenetics-Phylogenetic Tree

Phylogenetic Trees are classified as **rooted (i)** or **unrooted (ii)** trees.



Both rooted and unrooted trees can be either **bifurcating** or **multifurcating**.

Enumerating Trees

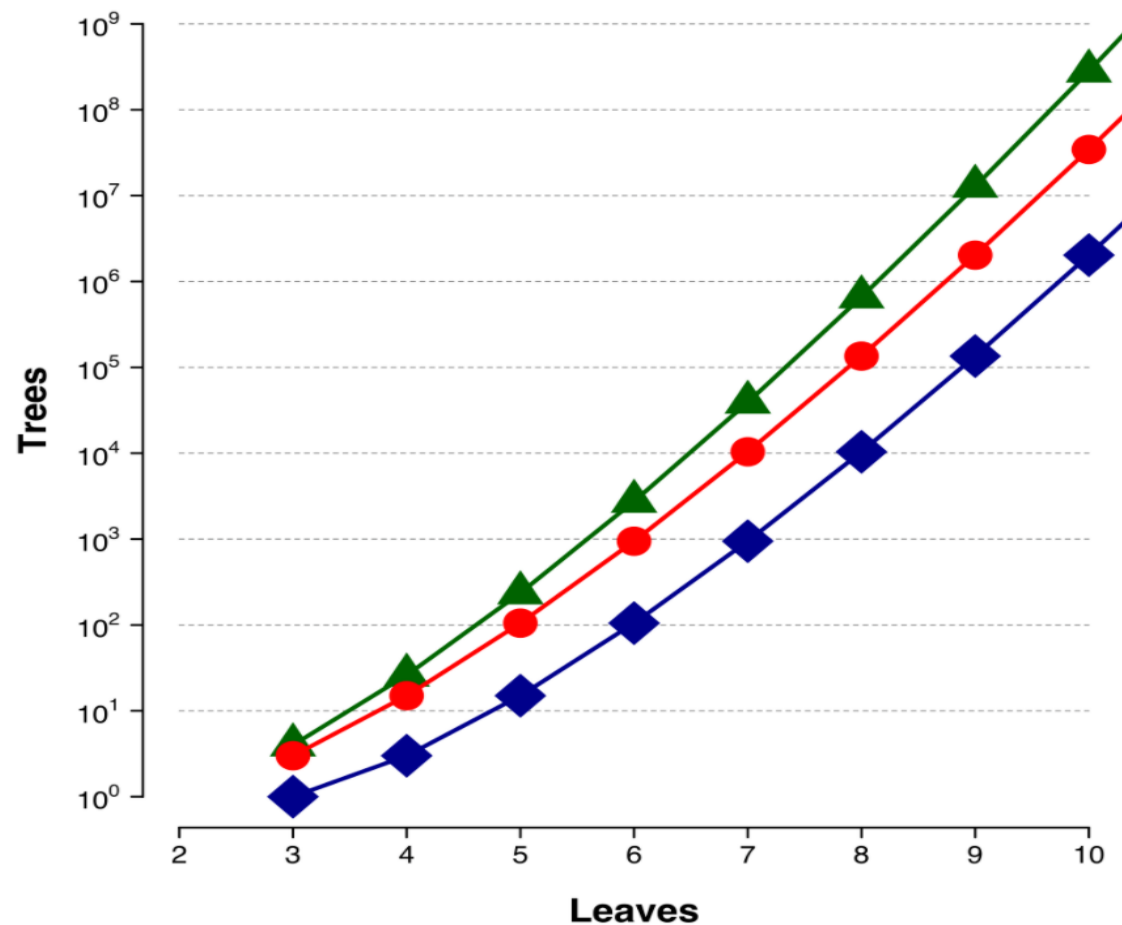
Given a bifurcating tree we can have $N_{\text{rooted}} = \frac{(2n-3)!}{2^{n-2}(n-2)!}$ rooted subtrees

and $N_{\text{unrooted}} = \frac{(2n-5)!}{2^{n-3}(n-3)!}$ unrooted subtrees

Among bifurcating trees, the number of unrooted trees with **n** leaves is equal to the number of rooted trees with **n-1** leaves.

Enumerating Trees

As the number of species (node leaves) increases, the possible trees increase significantly, making it difficult to identify the authentic phylogenetic tree that contains all known species so far.



Related Work

Related Work

Technology	Section	Algorithms	Data
FPGA	Sequence Comparison	BLAST CAST	DNA PROTEINS
	Multiple Sequence Alignment	MAAFT T-Coffee	DNA DNA
	Prediction of RNA and protein secondary structure	Zuker Predator	PROTEINS,DNA PROTEINS,DNA
	Gene identification	Glimmer	GENES
	Phylogenetic Trees	Bayesian RAxML	DNA PROTEINS,DNA
Intel MIC	Phylogenetic Trees	RAxML-Light EXaML	DNA DNA
GPU	Phylogenetic Trees	RAxML-Light EXaML	DNA DNA

Related Work

Technology	Section	Algorithms	Data
FPGA	Sequence Comparison	BLAST CAST	DNA PROTEINS
	Multiple Sequence Alignment	MAAFT T-Coffee	DNA DNA
	Prediction of RNA and protein secondary structure	Zuker Predator	PROTEINS,DNA PROTEINS,DNA
	Gene identification	Glimmer	GENES
	Phylogenetic Trees	Bayesian RxML	DNA PROTEINS ,DNA
Intel MIC	Phylogenetic Trees	RAxML-Light EXaML	DNA DNA
GPU	Phylogenetic Trees	RAxML-Light EXaML	DNA DNA

Maximum Likelihood Method

Maximum Likelihood

- **Maximum likelihood** is a general statistical method for estimating unknown parameters of a probability model.
- The **Likelihood** is defined to be a quantity proportional to the probability of observing the data given the model, $P(D | M)$.
- In **phylogenetics**, there are many parameters, including rates, differential transformation costs, and, most importantly, the tree itself.
- **In this case**, Maximum Likelihood can be used as an optimality measure for choosing a preferred tree or set of trees.

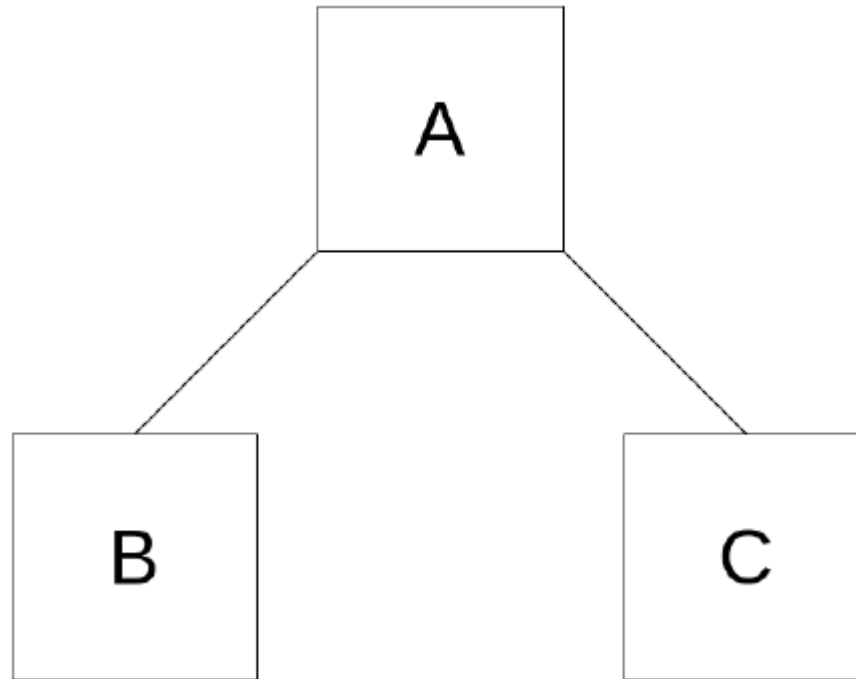
Models of Amino Acid Replacement

- All models used for the study of ML, assume that all amino acid sites in an alignment **evolve independently**. They have the property to be **time-reversible**.
- The probability of amino acid i being replaced by amino acid j over time T is $P_{ij}(T)$, called **replacement matrices**.
- In the **maximum likelihood (ML)** method, they are used to compute substitution probabilities along tree branches and hence the likelihood of the data. These matrices are calculated as $P(T) = \exp(TQ)$.

$$\begin{pmatrix} — & s_{1,2} & s_{1,3} & \cdots & s_{1,20} \\ s_{1,2} & — & s_{2,3} & \cdots & s_{2,20} \\ s_{1,3} & s_{2,3} & — & \cdots & s_{3,20} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{1,20} & s_{2,20} & s_{3,20} & \cdots & — \end{pmatrix} \cdot \text{diag}(\pi_1, \dots, \pi_{20})$$

Calculating Tree's Probability

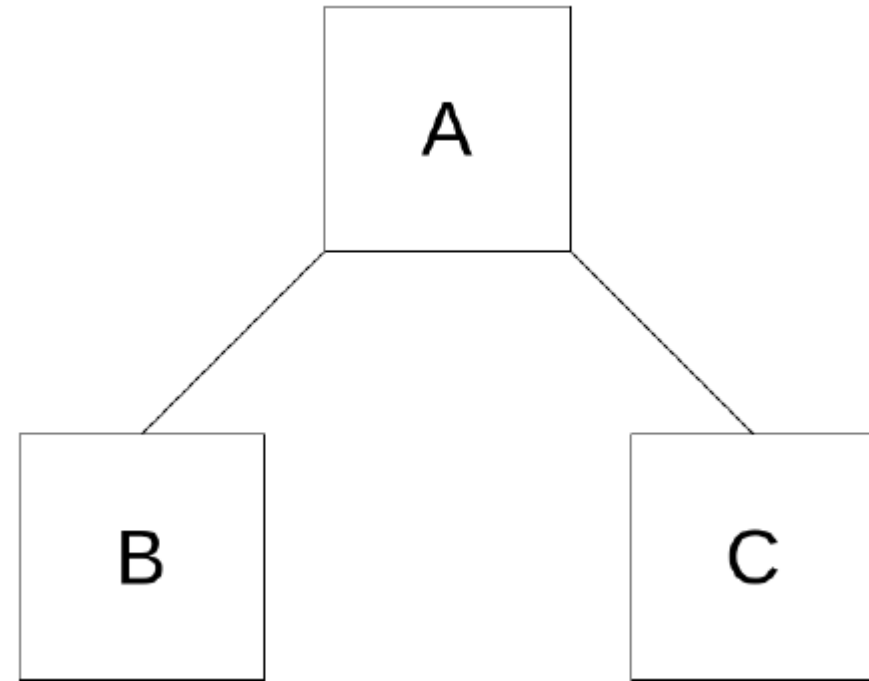
The **probability of the tree** can be estimated by calculating a score for each assumption made for all data, following **ML** and **FPA**.



Calculating Tree's Probability

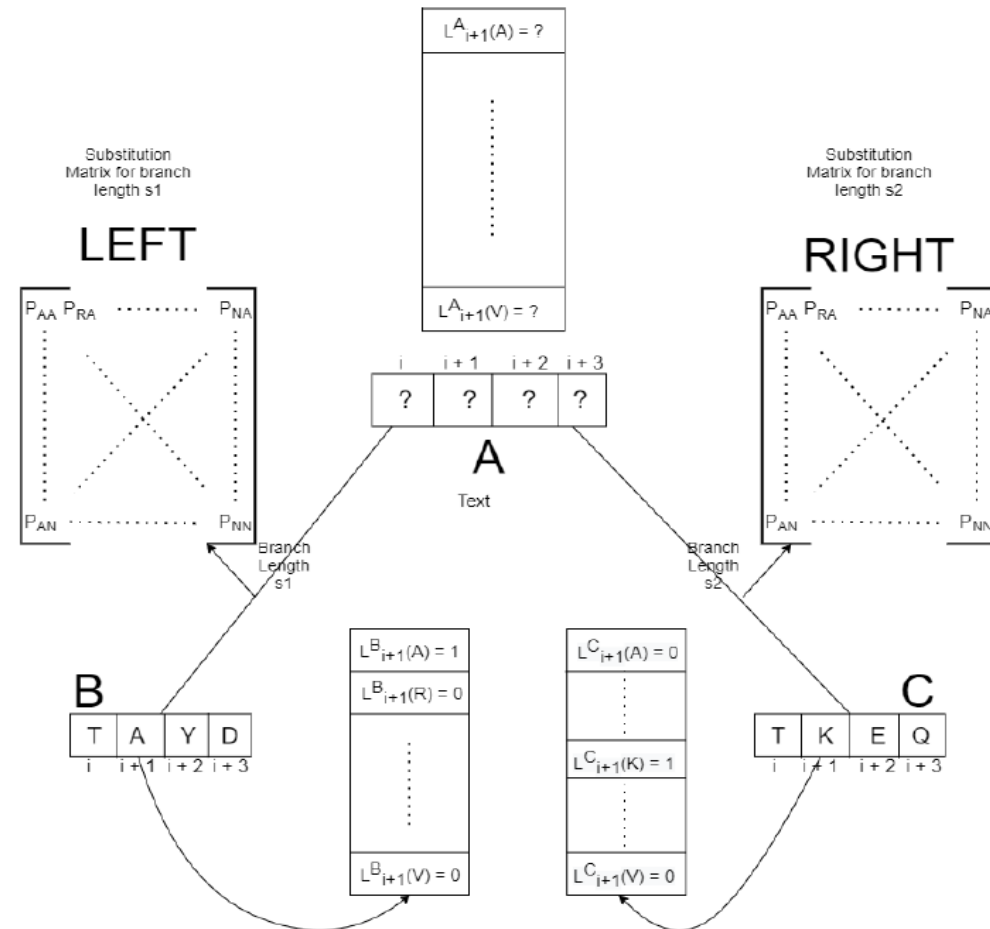
Assuming that B and C have A as common ancestor, the likelihood of root A must be calculated for each site N of B and C.

$$L(X_{Aj} = i) = \left[\sum_z P_{iz}(S_{AB}) L(X_{Bj} = z) \right] \left[\sum_z P_{iz}(S_{AC}) L(X_{Cj} = z) \right]$$



Calculating Tree's Probability

- Each **sequence** split into $N * 80$ vectors for each site N of amino acid sequence.
- Each separate **vector** consists of 4 sites
- Each **site** contains 20 probabilities
- The **probabilities** of N vectors frames a 4×20 matrix for each site of N sequences.
- Substitution Matrices** are called LEFT & RIGHT. Left for B child and Right for C child



Calculating Tree's Probability

The likelihood of site $i+1$ is calculated as :

$$L_{i+1}^A(A) = \left[P_{AA}(s1)L_{i+1}^B(A) + P_{AR}(s1)L_{i+1}^B(R) + \dots + P_{AY}(s1)L_{i+1}^B(Y) + P_{AV}(s1)L_{i+1}^B(V) \right] \\ \times \left[P_{AA}(s2)L_{i+1}^C(A) + P_{AR}(s2)L_{i+1}^C(R) + \dots + P_{AY}(s2)L_{i+1}^C(Y) + P_{AV}(s2)L_{i+1}^C(V) \right]$$

After that, the likelihoods of the common ancestor A are multiplied with the base frequencies p of the 20 different amino acids :

$$L(i) = \sum_j \pi_j * L(X_i^A = j), \text{ where } j = A, R, N, \dots Y, V$$

Total probability of the tree is computed by multiplying the probabilities of its i sites of common ancestor A between them :

$$L = \prod_i L(i)$$

Calculating Tree's Probability

The likelihood of site $i+1$ is calculated as :

$$L_{i+1}^A(A) = \left[P_{AA}(s1)L_{i+1}^B(A) + P_{AR}(s1)L_{i+1}^B(R) + \dots + P_{AY}(s1)L_{i+1}^B(Y) + P_{AV}(s1)L_{i+1}^B(V) \right] \\ \times \left[P_{AA}(s2)L_{i+1}^C(A) + P_{AR}(s2)L_{i+1}^C(R) + \dots + P_{AY}(s2)L_{i+1}^C(Y) + P_{AV}(s2)L_{i+1}^C(V) \right]$$

After that, the likelihoods of the common ancestor A are multiplied with the base frequencies p of the 20 different amino acids :

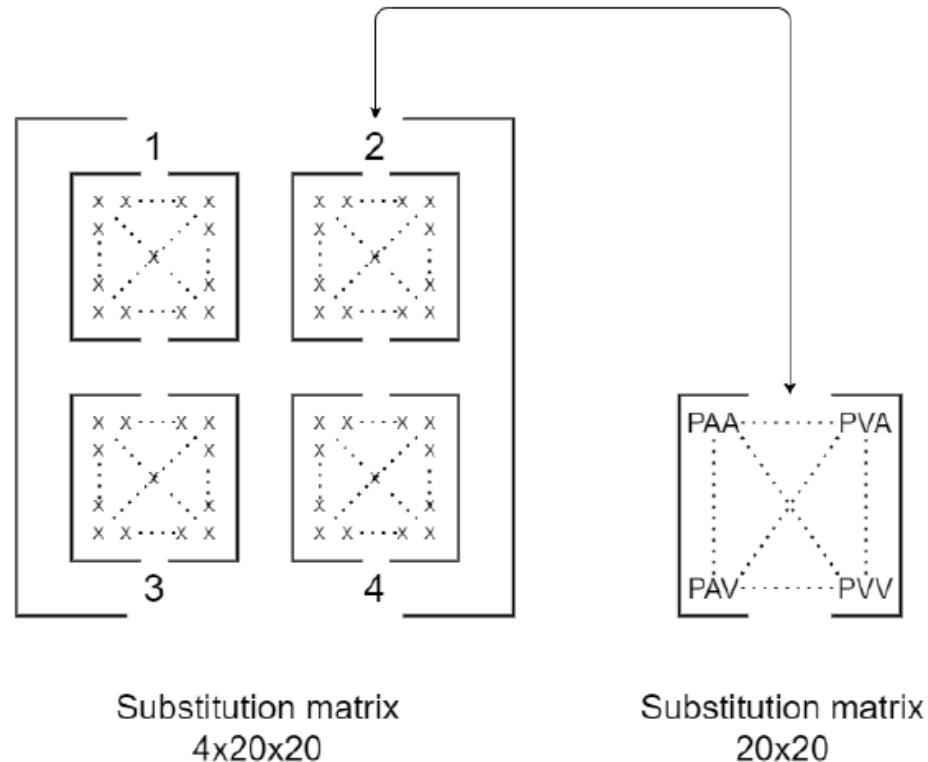
$$L(i) = \sum_j \pi_j * L(X_i^A = j), \text{ where } j = A, R, N, \dots Y, V$$

Total probability of the tree is computed by multiplying the probabilities of its i sites of common ancestor A between them :

$$L = \prod_i L(i) \longrightarrow L = \sum_i \log(L(i))$$

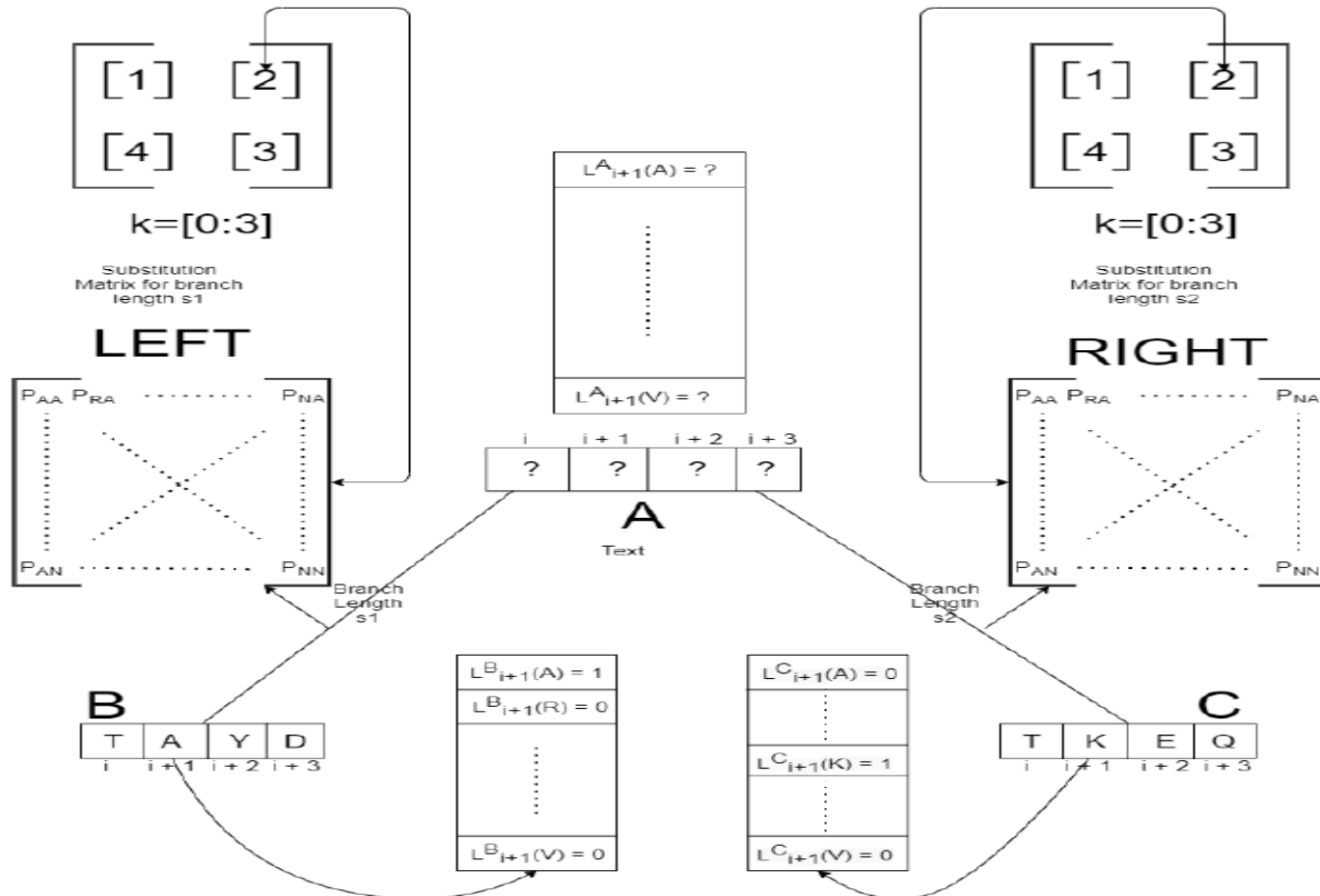
Adjustments on ML for RAxML

Discrete Gamma Model



Accumulate uniformly all different substitution matrices of all i sites of N vectors.

Adjustments on ML for RAxML

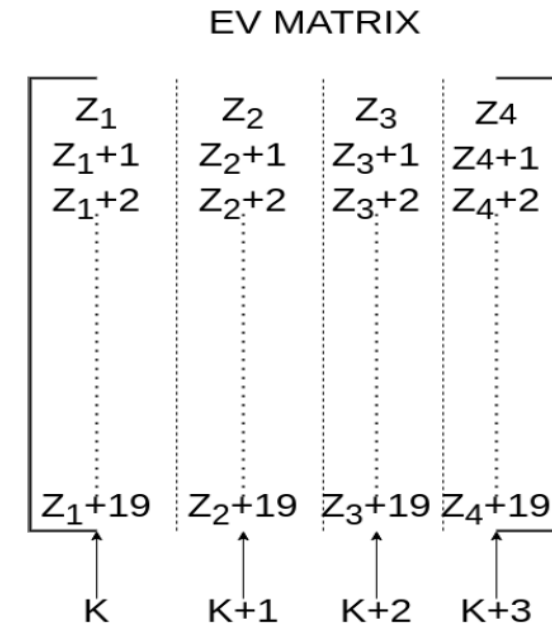


Adjustments on ML for RAxML

Eigen Vector (EV) is the eigenvectors of the relationship $P(s)=e^{\Lambda}Qs$.

$$L(X_{Aj} = i) = \sum_K \sum_z EV_K(z) L(X_{Aj} = K)$$

K = 20 likelihoods of one i site of the vector A and corresponds to 20 eigenvectors of matrix EV



Architecture Design

Phylogenetic Likelihood Library

- NewView() or PLF (Phylogenetic Likelihood Function)
- Evaluate()
- coreDerivative()
- sumGAMMA()

Phylogenetic Likelihood Library

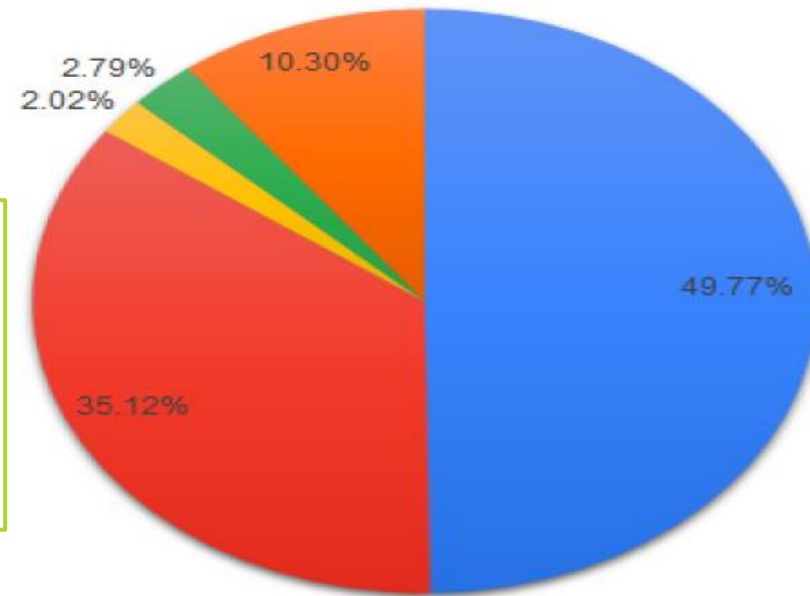
Profiling of PLL

- PLF() \geq 85% of total time
- Evaluate() $<$ 2% of total time
- coreDerivative() $<$ 2% of total time
- sumGAMMA() \approx 5% of total time

Phylogenetic Likelihood Library

Profiling of PLL

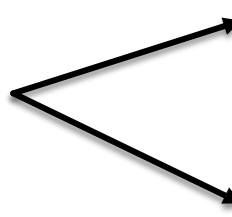
- PLF() $\geq 85\%$ of total time
- Evaluate() $< 2\%$ of total time
- coreDerivative() $< 2\%$ of total time
- sumGAMMA() $\sim 5\%$ of total time



■ NewViewGTRGAMMAPROT Tip - Tip & Tip - Inner
■ NewViewGTRGAMMAPROT Inner - Inner
■ SumGAMMAPROT Tip - Tip & Tip - Inner
■ SumGAMMAPROT Inner - Inner
■ Other

Estimated Speedup

Amdahl's law



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

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

Estimated Speedup

Amdahl's law

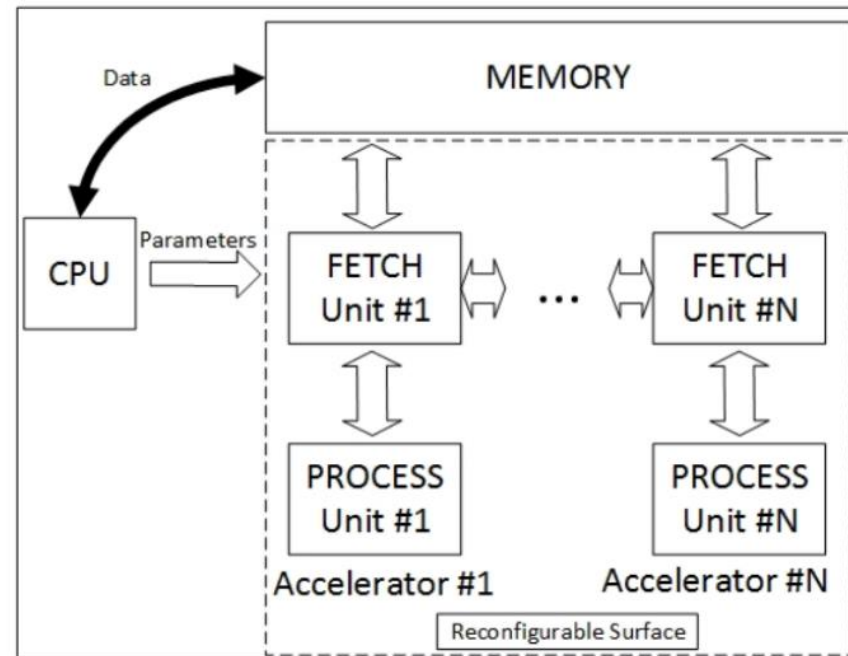
$$\begin{aligned}
 & \rightarrow 1) S_{\text{latency}}(s) = \frac{1}{1-p} \\
 & \rightarrow 2) S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}
 \end{aligned}$$

	PLF	RAxML
Inner - Inner	X1.82	X1.63
All cases		X7.7

	SumGAMMA	RAxML
Inner - Inner	X1.92	X1.02
All cases		X1.05

DAER ARCHITECTURE

Using DAER architecture, the application is split into two parts, the part of **data processing** and the **data fetching**.



FETCH UNITS

- ▶ Use Fetch Units for the **two inputs** and **one output**.
- ▶ Each Fetch Unit consists of 2 functions, **BurstMem** and **BurstConv**.
- ▶ **BurstMem** passes input values to streams. Input values and streams are `ap_uint<512>` type.
- ▶ **BurstConv** converts input streams into `DataType` streams.
- ▶ Knowing the available **bandwidth** of platforms and the **size of data**, we can calculate the optimal **II** .
- ▶ We can adjust the fetch units in order to achieve $II \leq \text{optimal II}$.

PROCESSING UNITS

39

Algorithm 1 PLF , based on a three vector tree with root A

```
for  $i = 0 : N$  do
   $X_1[80] \leftarrow B(i)vector$ 
   $X_2[80] \leftarrow C(i)vector$ 
   $X_3[80] \leftarrow A(i)vector$ 
  for  $j = 0 : 3$  do
    for  $k = 0 : 19$  do
      for  $l = 0 : 19$  do
        //Calculation of left distance ( $B \rightarrow A$ )
         $Tmp_{x=x_1} \leftarrow X_1[j * 4 + l] * Left[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_1} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_1}$ 
        //Calculation of right distance ( $C \rightarrow A$ )
         $Tmp_{x=x_2} \leftarrow X_2[j * 4 + l] * Right[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_2} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_2}$ 
        //Multiplication of summarized  $ump_{x_1}$  and  $ump_{x_2}$ 
         $x1px2_j[k] \leftarrow Ump_{x=x_1} * Ump_{x=x_2}$ 
      for  $k = 0 : 19$  do
        for  $l = 0 : 19$  do
          //Multiplication of EV and  $x1px2 \rightarrow X_3$ 
           $X_3[j * 4 + l] += x1px2_j[k] * EV[k * 20 + l]$ 
        //Continue with the scaling
         $scale \leftarrow 0$ 
         $addscale \leftarrow 0$ 
        for  $l = 0 : 79$  do
           $scale += (ABS(X_3[l] < minlikelihood))$ 
        if  $scale \neq 0$  then
           $X_3 \leftarrow X_3 * factorM$ 
           $addscale += wgt[i]$ 
        else
           $X_3 \leftarrow X_3$ 
```

PROCESSING UNITS

40

Algorithm 1 PLF , based on a three vector tree with root A

```

for  $j = 0 : N$  do
   $X_1[80] \leftarrow B(i)vector$ 
   $X_2[80] \leftarrow C(i)vector$ 
   $X_3[80] \leftarrow A(i)vector$ 
  for  $j = 0 : 3$  do
    for  $k = 0 : 19$  do
      for  $l = 0 : 19$  do
        //Calculation of left distance ( $B \rightarrow A$ )
         $Tmp_{x=x_1} \leftarrow X_1[j * 4 + l] * Left[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_1} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_1}$ 
        //Calculation of right distance ( $C \rightarrow A$ )
         $Tmp_{x=x_2} \leftarrow X_2[j * 4 + l] * Right[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_2} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_2}$ 
        //Multiplication of summarized  $ump_{x_1}$  and  $ump_{x_2}$ 
         $x1px2_j[k] \leftarrow Ump_{x=x_1} * Ump_{x=x_2}$ 
      for  $k = 0 : 19$  do
        for  $l = 0 : 19$  do
          //Multiplication of EV and  $x1px2 \rightarrow X_3$ 
           $X_3[j * 4 + l] += x1px2_j[k] * EV[k * 20 + l]$ 
        //Continue with the scaling
         $scale \leftarrow 0$ 
         $addscale \leftarrow 0$ 
        for  $l = 0 : 79$  do
           $scale += (ABS(X_3[l] < minlikelihood))$ 
        if  $scale \neq 0$  then
           $X_3 \leftarrow X_3 * factorM$ 
           $addscale += wgt[i]$ 
        else
           $X_3 \leftarrow X_3$ 

```

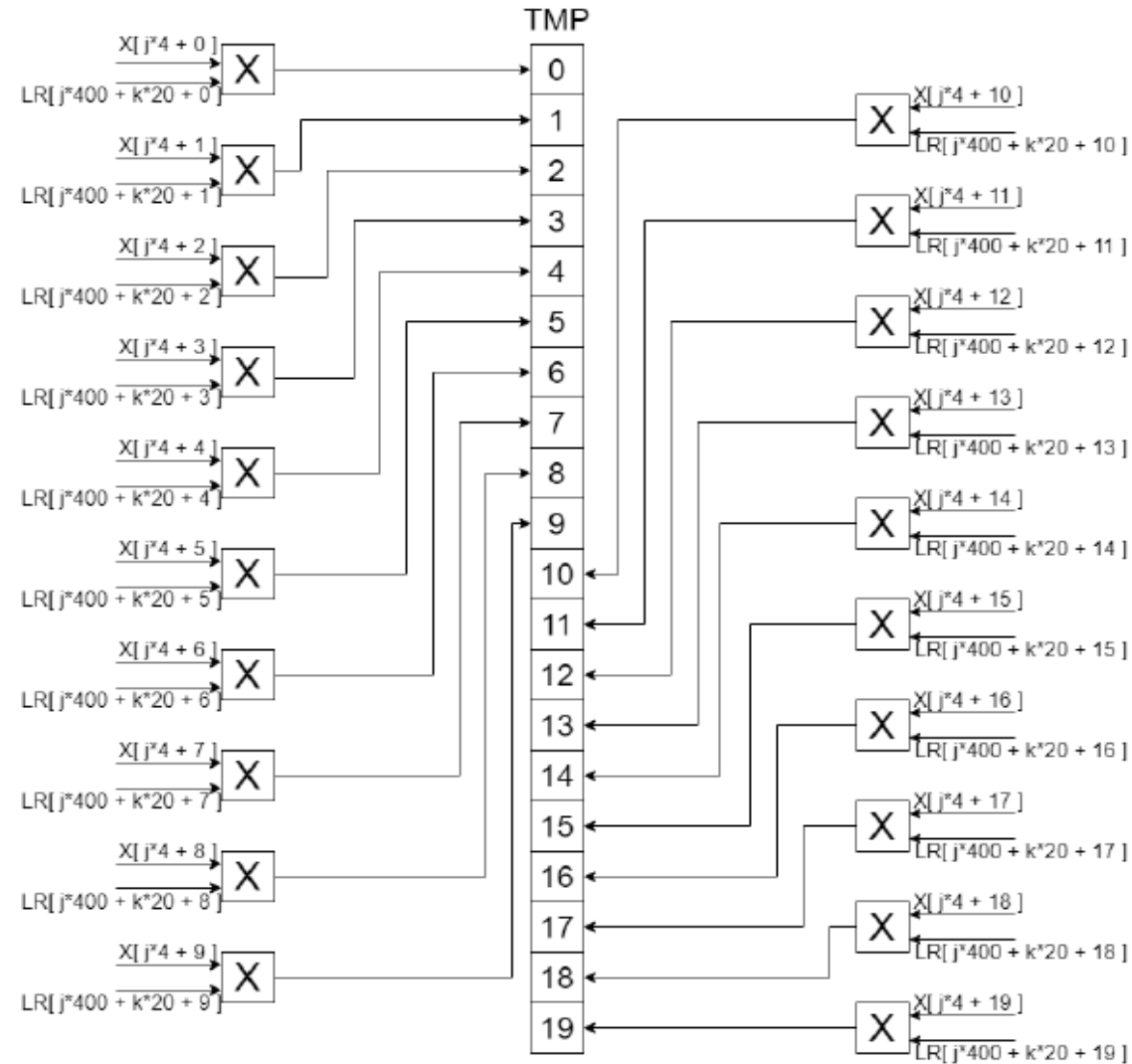

PROCESSING UNITS

41

Algorithm 1 PLF , based on a three vector tree with root A

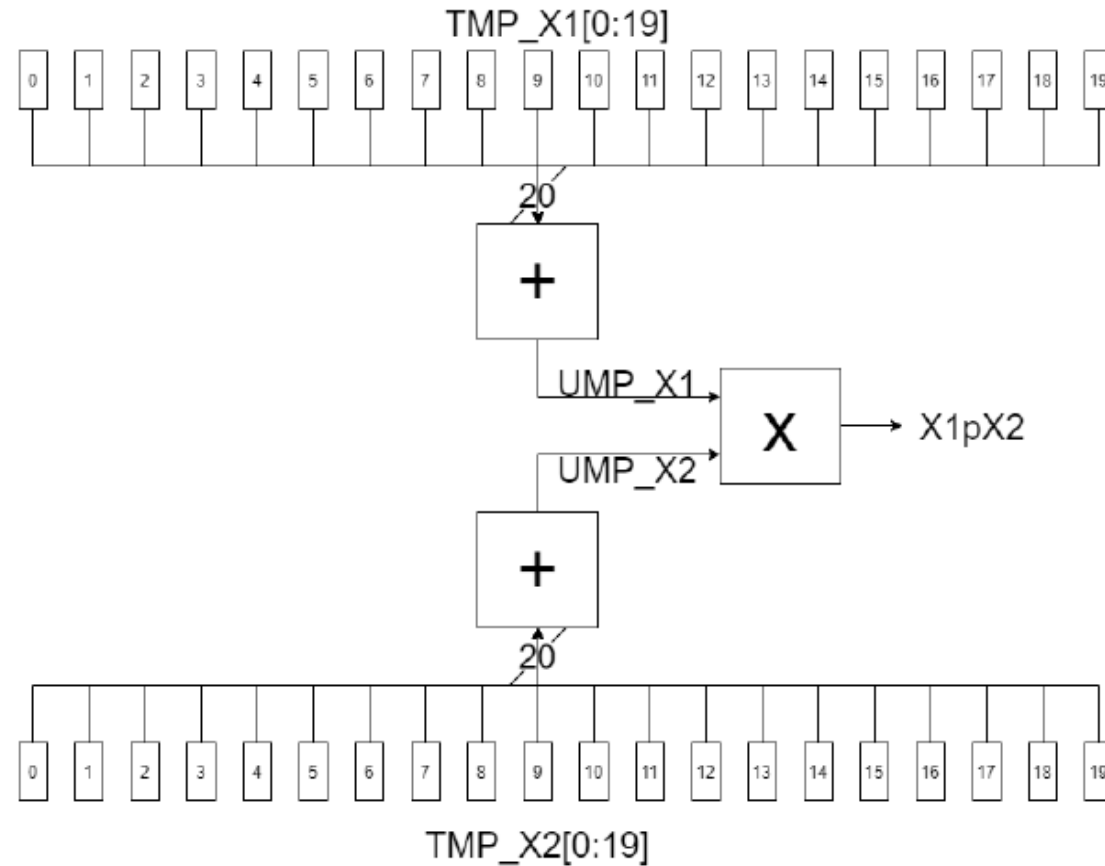
```
for  $i = 0 : N$  do
   $X_1[80] \leftarrow B(i)vector$ 
   $X_2[80] \leftarrow C(i)vector$ 
   $X_3[80] \leftarrow A(i)vector$ 
  for  $j = 0 : 3$  do
    for  $k = 0 : 19$  do
      for  $l = 0 : 19$  do
        //Calculation of left distance ( $B \rightarrow A$ )
         $Tmp_{x=x_1} \leftarrow X_1[j * 4 + l] * Left[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_1} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_1}$ 
        //Calculation of right distance ( $C \rightarrow A$ )
         $Tmp_{x=x_2} \leftarrow X_2[j * 4 + l] * Right[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_2} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_2}$ 
        //Multiplication of summarized  $ump_{x_1}$  and  $ump_{x_2}$ 
         $x1px2_j[k] \leftarrow Ump_{x=x_1} * Ump_{x=x_2}$ 
      for  $k = 0 : 19$  do
        for  $l = 0 : 19$  do
          //Multiplication of EV and  $x1px2 \rightarrow X_3$ 
           $X_3[j * 4 + l] += x1px2_j[k] * EV[k * 20 + l]$ 
        //Continue with the scaling
         $scale \leftarrow 0$ 
         $addscale \leftarrow 0$ 
        for  $l = 0 : 79$  do
           $scale += (ABS(X_3[l] < minlikelihood))$ 
        if  $scale \neq 0$  then
           $X_3 \leftarrow X_3 * factorM$ 
           $addscale += wgt[i]$ 
        else
           $X_3 \leftarrow X_3$ 
```

PROCESSING UNITS



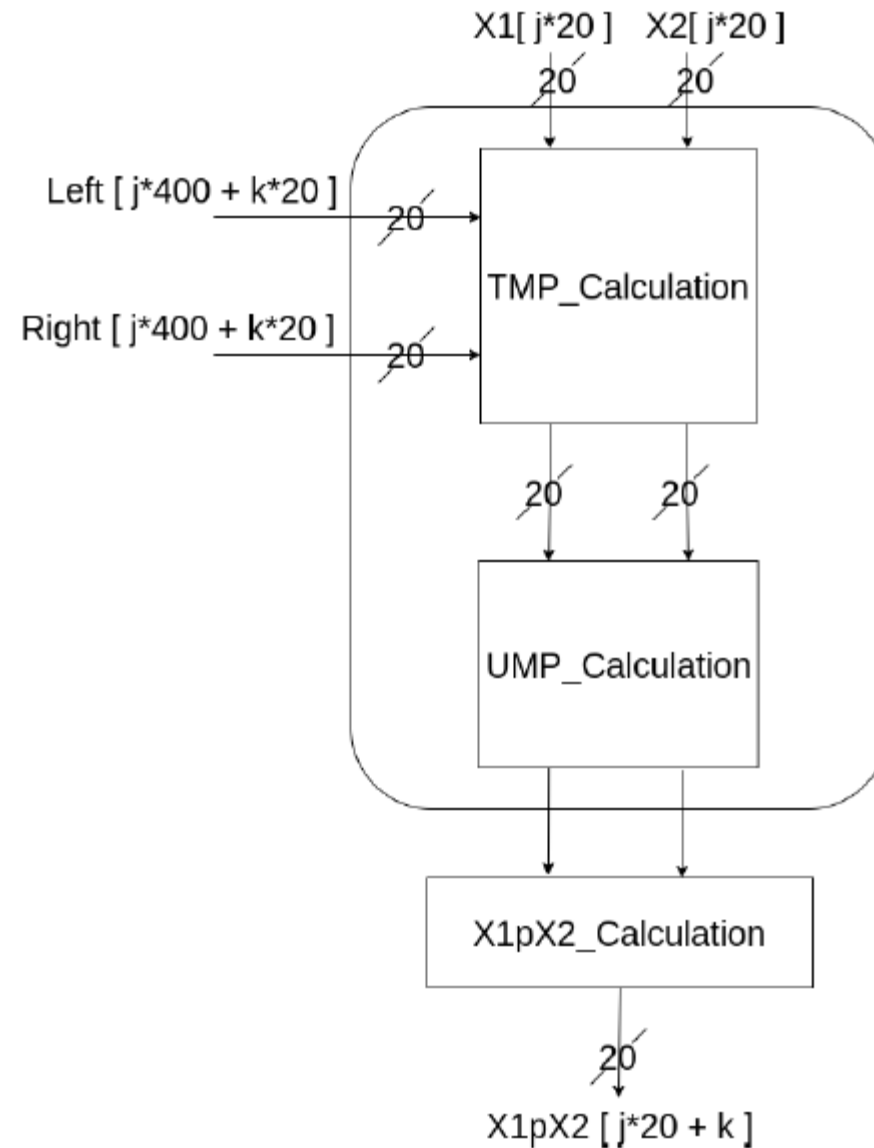
Multiplications For The Calculation of UMP for X1,X2 input vectors

PROCESSING UNITS



Summation of TMP vector to produce UMP_{X1} and UMP_{X2} and their multiplication for $X1pX2$

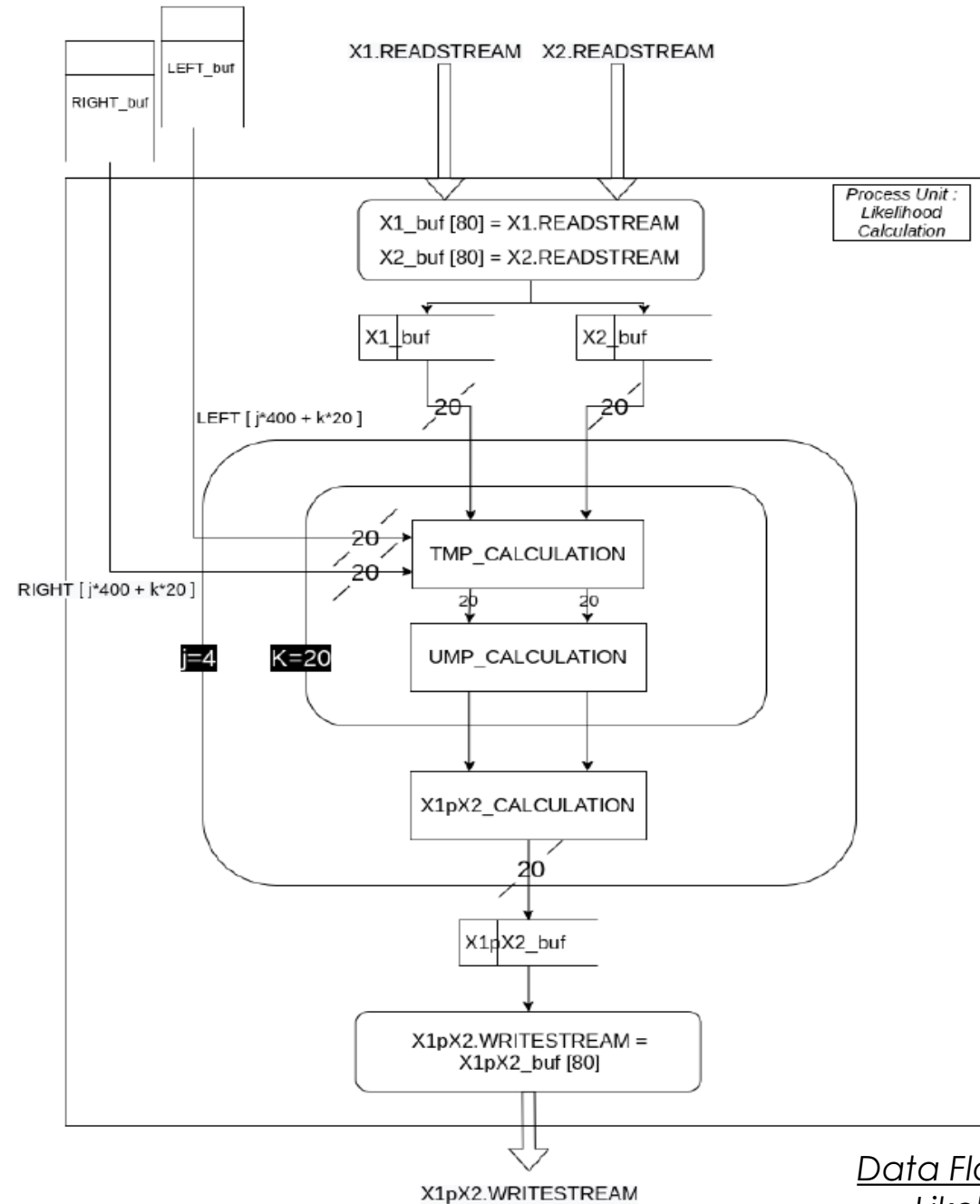
PROCESSING UNITS



Calculation of likelihoods for a j site

PROCESSING UNITS

45



*Data Flow of Processing Unit :
Likelihood Calculation*

Processing Units

46

Algorithm 1 PLF , based on a three vector tree with root A

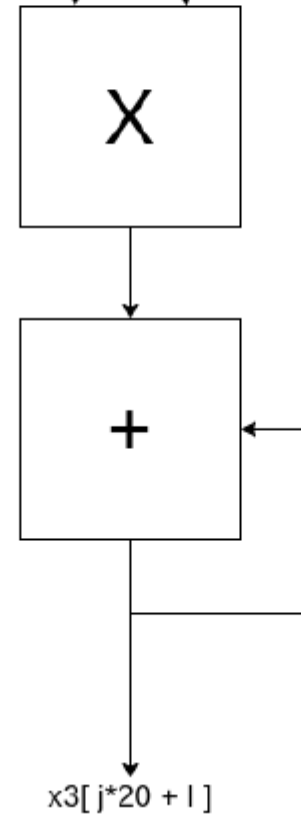
```
for  $i = 0 : N$  do
   $X_1[80] \leftarrow B(i)vector$ 
   $X_2[80] \leftarrow C(i)vector$ 
   $X_3[80] \leftarrow A(i)vector$ 
  for  $j = 0 : 3$  do
    for  $k = 0 : 19$  do
      for  $l = 0 : 19$  do
        //Calculation of left distance ( $B \rightarrow A$ )
         $Tmp_{x=x_1} \leftarrow X_1[j * 4 + l] * Left[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_1} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_1}$ 
        //Calculation of right distance ( $C \rightarrow A$ )
         $Tmp_{x=x_2} \leftarrow X_2[j * 4 + l] * Right[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_2} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_2}$ 
        //Multiplication of summarized  $ump_{x_1}$  and  $ump_{x_2}$ 
         $x1px2_j[k] \leftarrow Ump_{x=x_1} * Ump_{x=x_2}$ 
      for  $k = 0 : 19$  do
        for  $l = 0 : 19$  do
          //Multiplication of EV and  $x1px2 \rightarrow X_3$ 
           $X_3[j * 4 + l] += x1px2_j[k] * EV[k * 20 + l]$ 
        //Continue with the scaling
         $scale \leftarrow 0$ 
         $addscale \leftarrow 0$ 
        for  $l = 0 : 79$  do
           $scale += (ABS(X_3[l] < minlikelihood))$ 
        if  $scale \neq 0$  then
           $X_3 \leftarrow X_3 * factorM$ 
           $addscale += wgt[i]$ 
        else
           $X_3 \leftarrow X_3$ 
```

Processing Units

$EV[k * 20 + l]$ $X1pX2[j * 20 + k]$

$j = 0 : 3$
 $k = 0 : 19$
 $l = 0 : 19$

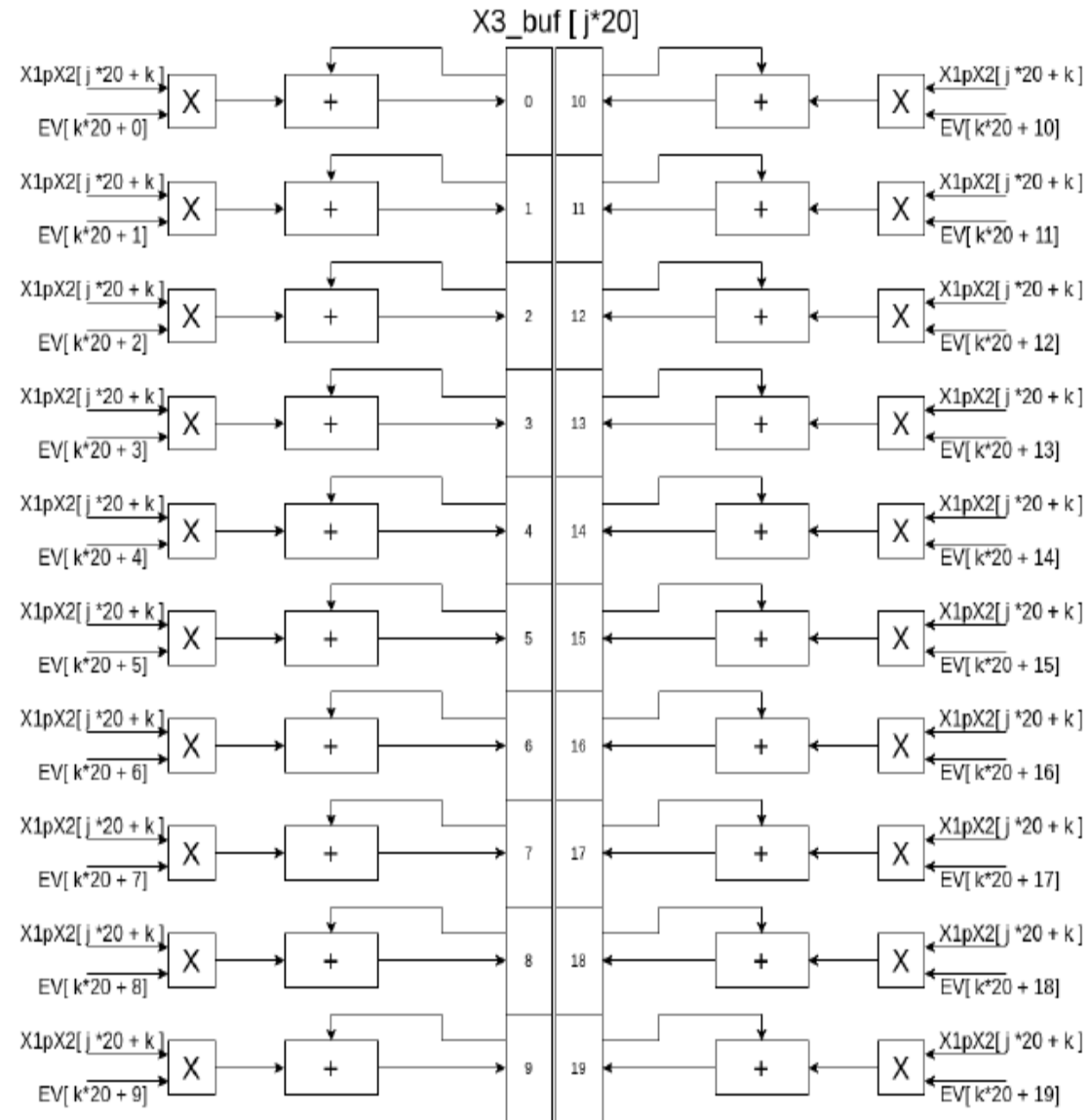
47



Multiplication Of EV with X1pX2 and a recursive addition for X3

Processing Units

48



Final Calculation of a j site of $X3$ vector

Processing Units

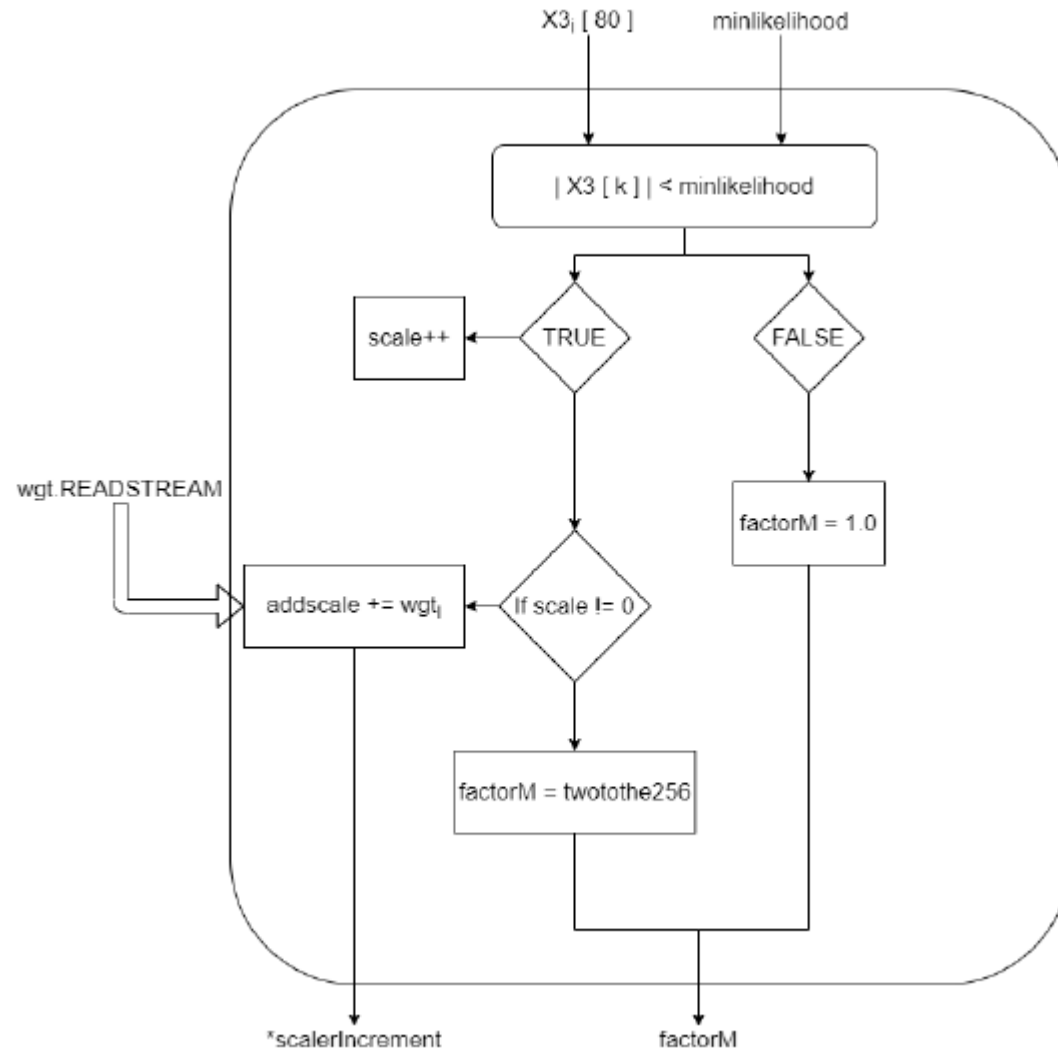
49

Algorithm 1 PLF , based on a three vector tree with root A

```
for  $i = 0 : N$  do
   $X_1[80] \leftarrow B(i)vector$ 
   $X_2[80] \leftarrow C(i)vector$ 
   $X_3[80] \leftarrow A(i)vector$ 
  for  $j = 0 : 3$  do
    for  $k = 0 : 19$  do
      for  $l = 0 : 19$  do
        //Calculation of left distance ( $B \rightarrow A$ )
         $Tmp_{x=x_1} \leftarrow X_1[j * 4 + l] * Left[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_1} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_1}$ 
        //Calculation of right distance ( $C \rightarrow A$ )
         $Tmp_{x=x_2} \leftarrow X_2[j * 4 + l] * Right[j * 400 + k * 20 + l]$ 
         $Ump_{x=x_2} \leftarrow \sum_{i=0}^{19} Tmp_{x=x_2}$ 
        //Multiplication of summarized  $ump_{x_1}$  and  $ump_{x_2}$ 
         $x1px2_j[k] \leftarrow Ump_{x=x_1} * Ump_{x=x_2}$ 
      for  $k = 0 : 19$  do
        for  $l = 0 : 19$  do
          //Multiplication of EV and  $x1px2 \rightarrow X_3$ 
           $X_3[j * 4 + l] += x1px2_j[k] * EV[k * 20 + l]$ 
        //Continue with the scaling
         $scale \leftarrow 0$ 
         $addscale \leftarrow 0$ 
        for  $l = 0 : 79$  do
           $scale += (ABS(X_3[l] < minlikelihood))$ 
        if  $scale \neq 0$  then
           $X_3 \leftarrow X_3 * factorM$ 
           $addscale += wgt[i]$ 
        else
           $X_3 \leftarrow X_3$ 
```

Processing Units

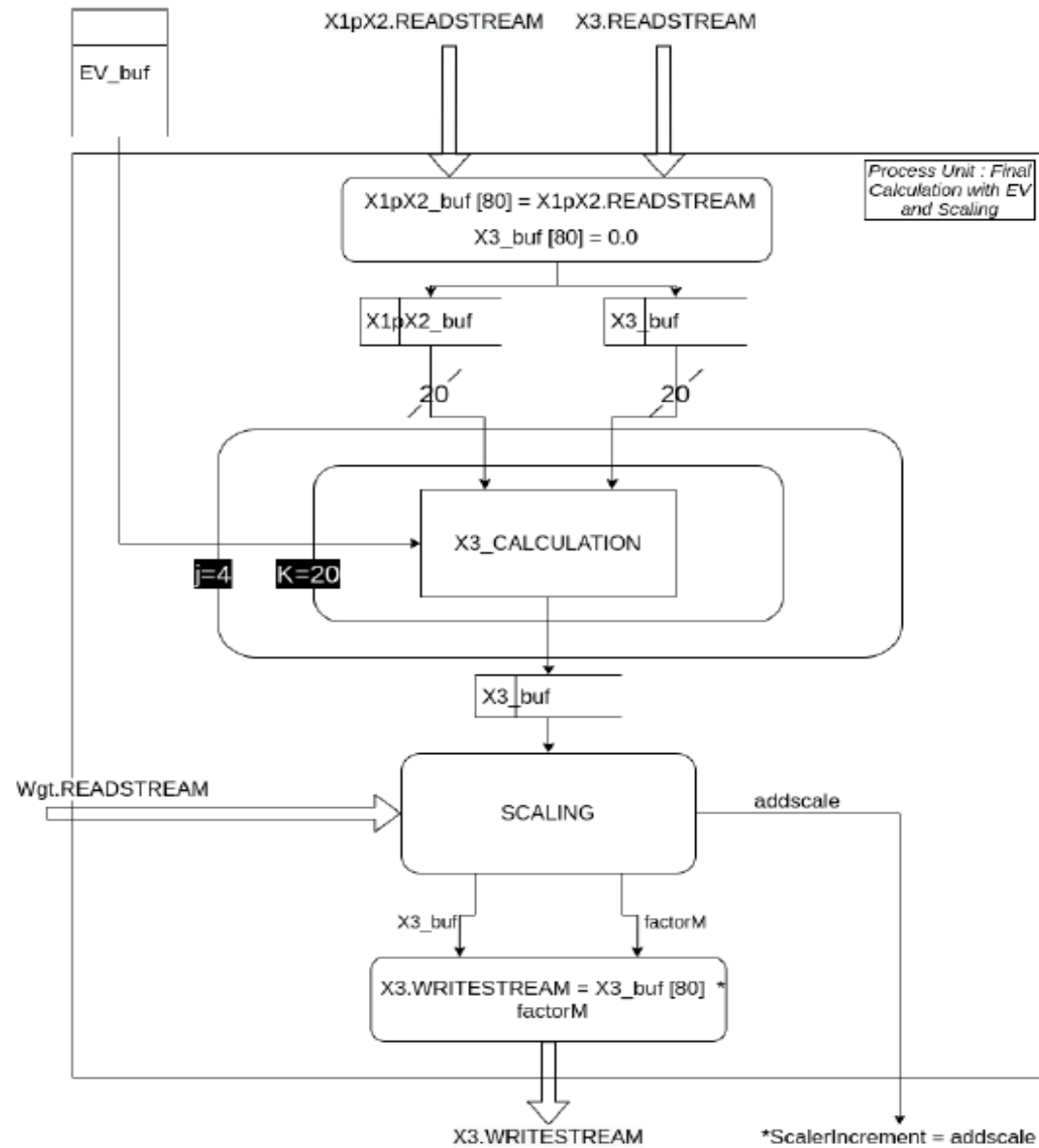
50



Scaling process of X3 vector

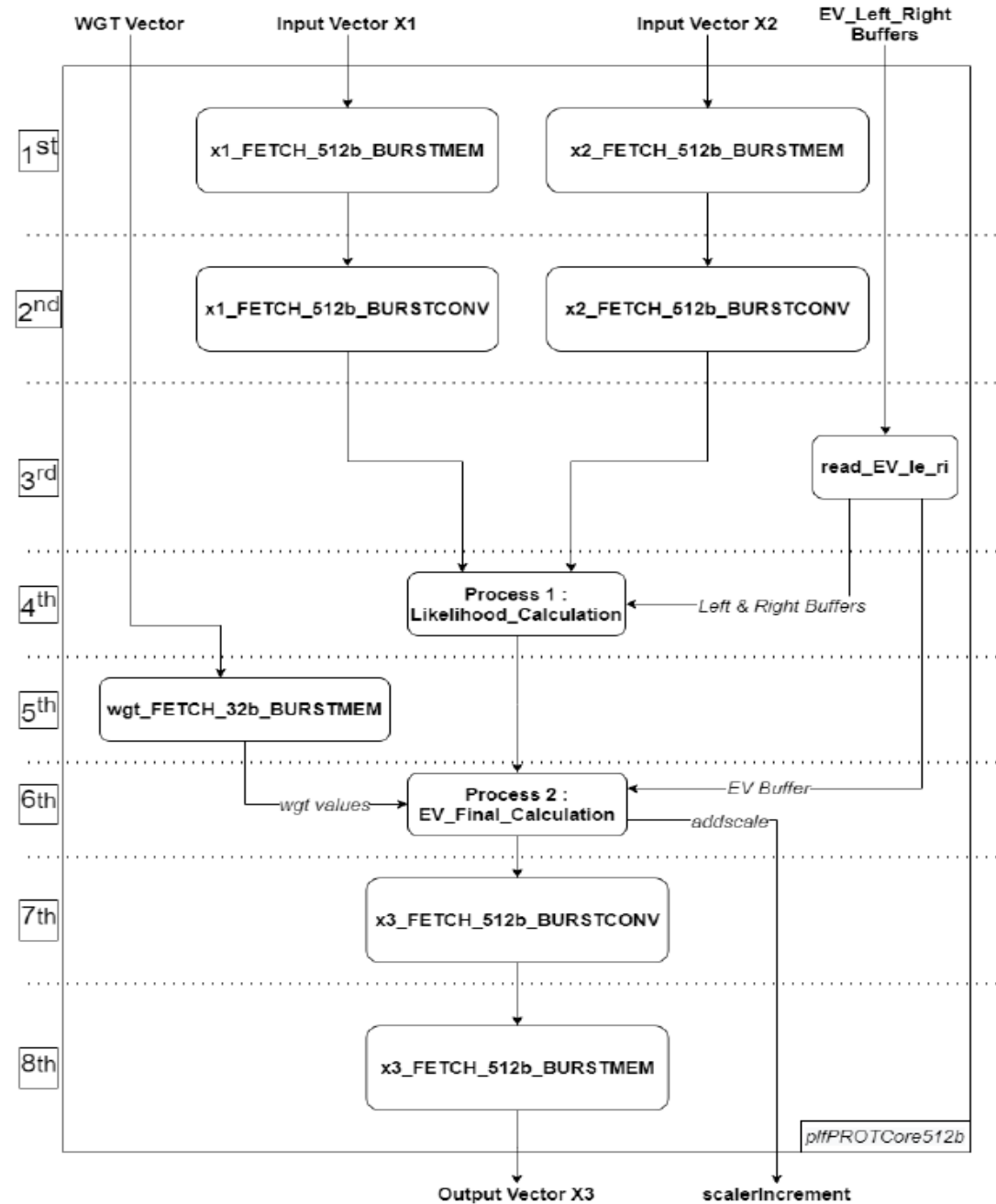
Processing Units

51



Data Flow of Processing Unit : EV Final Calculation

Processing Units

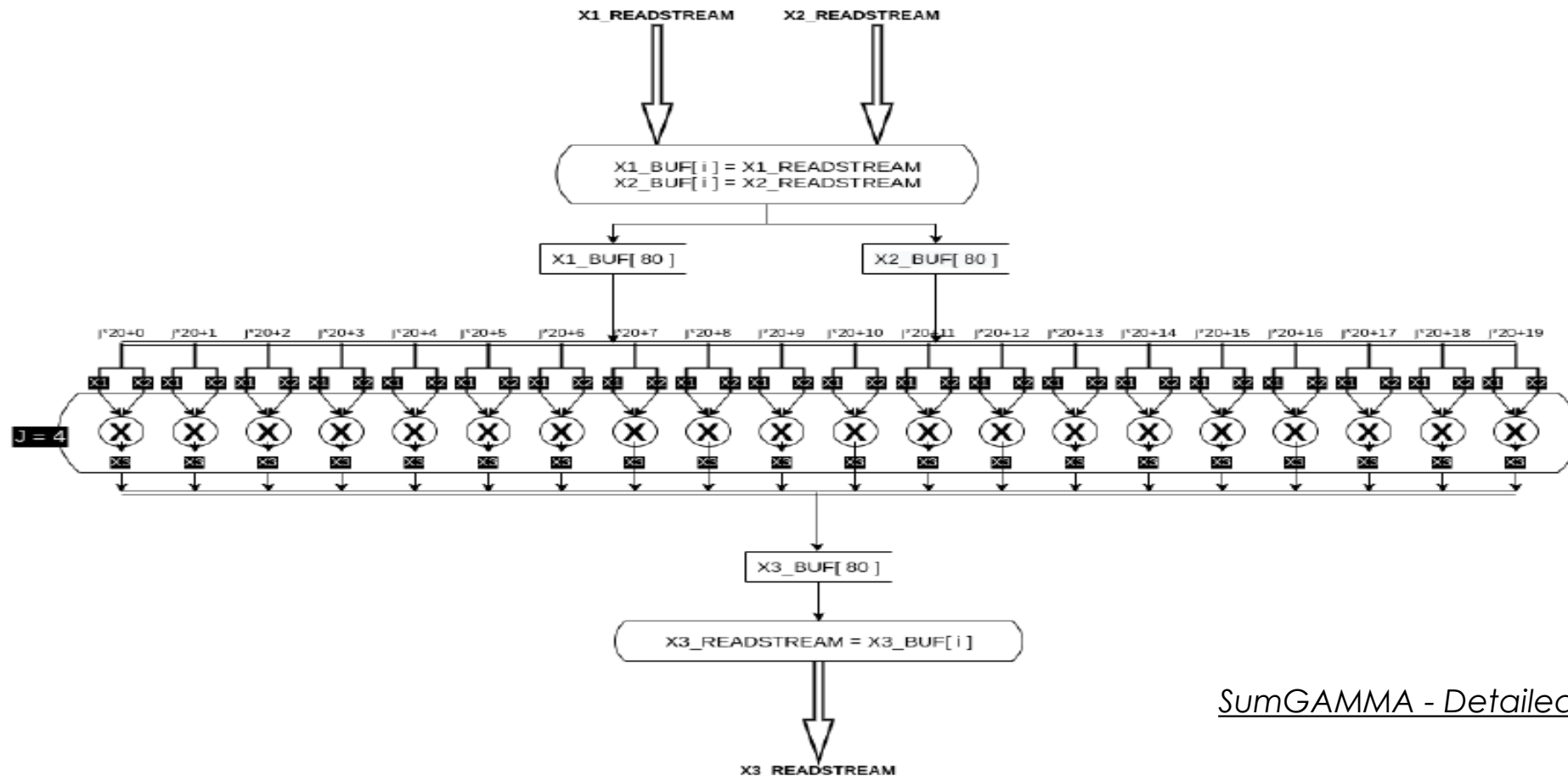


Processing Units

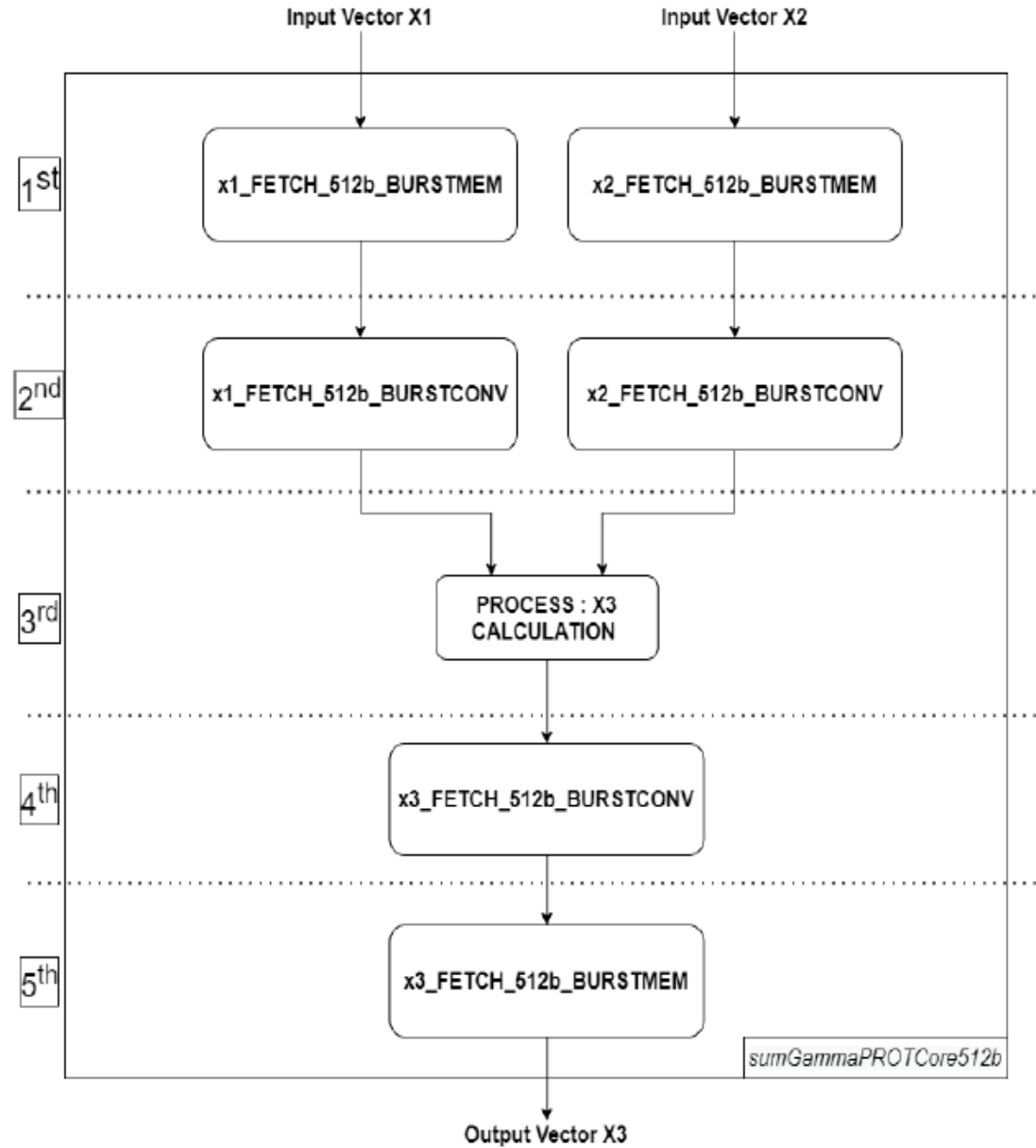
Algorithm 2 SumGamma Calculation

```
for  $i = 0 : N$  do
     $X_1[80] \leftarrow \text{Left}(i)\text{vector}$ 
     $X_2[80] \leftarrow \text{Right}(i)\text{vector}$ 
     $X_3[80] \leftarrow \text{Sum}(i)\text{vector}$ 
    for  $j = 0 : 3$  do
        for  $k = 0 : 19$  do
            //Calculation of Sum
             $X_3[j * 4 + k] \leftarrow X_1[j * 4 + k] * X_2[j * 4 + k]$ 
```

Processing Units



Processing Units



FPGA Implementation

Tools

- ▶ Vivado High Level Synthesis (HLS)
- ▶ Vivado SDx (SDSoc & SDAccel)
- ▶ Vivado IDE

Vivado High Level Synthesis (HLS)

Synthesis Report : Latency, Iteration Latency, Initiation Interval, Pipelined, Area (BRAM, URAM, FF, DSP, LUT)

Optimization Directives :

- **Interface** : Maps the top-level function's arguments to RTL ports to configure the IP block's functionality. The interface directive specifies each argument's port type.
- **Dataflow** : Enables parallel execution of functions and loops, increasing throughput and latency. It is used on our top level function.
- **Pipeline** : Reduces the number of clock cycles a function or loop can accept new inputs, by allowing the concurrent execution of operations.
- **Inline** : Removes a function as a separate entity in the hierarchy.
- **Array Partition** : Partitions an array into multiple smaller arrays or assigns each array's element to its register.
- **Resource** : Specifies the resource (core) is used to implement a variable.

SDSoc & SDAccel

SDSoc Directives :

- **Data access pattern** : Specifies the data access pattern in the hardware function so as to determine the hardware interface to synthesize.
- **Data copy** : Means that data are explicitly copied between the host processor memory and the hardware function.
- **Data zero_copy** : Means that the hardware function accesses the data directly from shared memory through an AXI master bus interface.
- **Data mem_attribute** : Tells the compiler whether the arguments have been allocated in physically contiguous memory.
- **Data sys_port** : Used in order to assign the arguments to specific memory ports.

In **SDAccel** environment we use OpenCL to build the kernel and the communication with the host on devices

Vivado IDE

Strategies & Directives :

- **OPT_DESIGN = ExploreArea** : Runs multiple passes of optimization with an emphasis on reducing combinational logic.
- **PLACE_DESIGN = SSI_BalanceSLRs** : Partitions across Super Logic Regions (SLRs) while attempting to balance Super Logic Lines (SLLs) between SLRs. In other words, trying to balance resources on all SLRs of the platform.
- **PHYS_OPT_DESIGN = AggressiveFanoutOpt** : Uses different algorithms for fanout-related optimizations with more aggressive goals.
- **ROUTE_DESIGN = AlternateCLBRouting** : Chooses alternate routing algorithms that require extra runtime but may help resolve routing congestion.
- **POST_ROUTE_DESIGN = AggressiveExplore** : Higher and aggressive placer effort in detail placement and post-placement optimization goals.

FPGA Platforms

- ▶ **ZCU102** : includes 4GB of DDR4 for the Processing System, 512MB of DDR4 for the Programmable Logic, 264MB Quad-SPI Flash, and an SDIO card interface. Its CPU frequency is 1200(MHz).
- ▶ **AWS EC2 F1 Instance** : includes four channels of DDR4-2400 DIMMs (64GB), the expanded partial reconfiguration flow for high fabric resource availability, and Xilinx DMA Subsystem for PCI Express with PCIe Gen3 x16 connectivity. Its runtime is OpenCL. Moreover, its CPU frequency is 2.3GHz on basic mode and it can reach the peak of 2.7GHz on turbo mode.

FPGA Platforms

ZCU102

	II	Frequency	BRAMs	DSPs	LUTs	REGs
TOTAL			4,320	6,840	2,364,480	1,182,240
SumGAMMAPROT	80	200 MHz	1.81%	3.49%	6.86%	5.91%
PLF	160	100 MHz	10.64%	17.54%	61.59%	25.67%

AWS F1

	II	Frequency	BRAMs	DSPs	LUTs	REGs
TOTAL			4,320	6,840	2,364,480	1,182,240
SumGAMMAPROT	10	350 MHz	1.26%	1.17%	1.78%	1.71%
PLF	20	100 MHz	19.25%	50.48%	85.35%	53.05%
	40	100 MHz	14.79%	25.24%	49.57%	36.28%
	80	150 MHz	8.09%	12.62%	39.79%	25.62%
	160	150 MHz	4.74%	6.47%	19.31%	8.87%

Results

Software Performance

RAxML

CPU :	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20 GHz
Architecture :	x86_64
Thread(s) / Core :	2
Core(s) / Socket :	10
Socket(s) :	2
RAM :	251GB

Software Performance

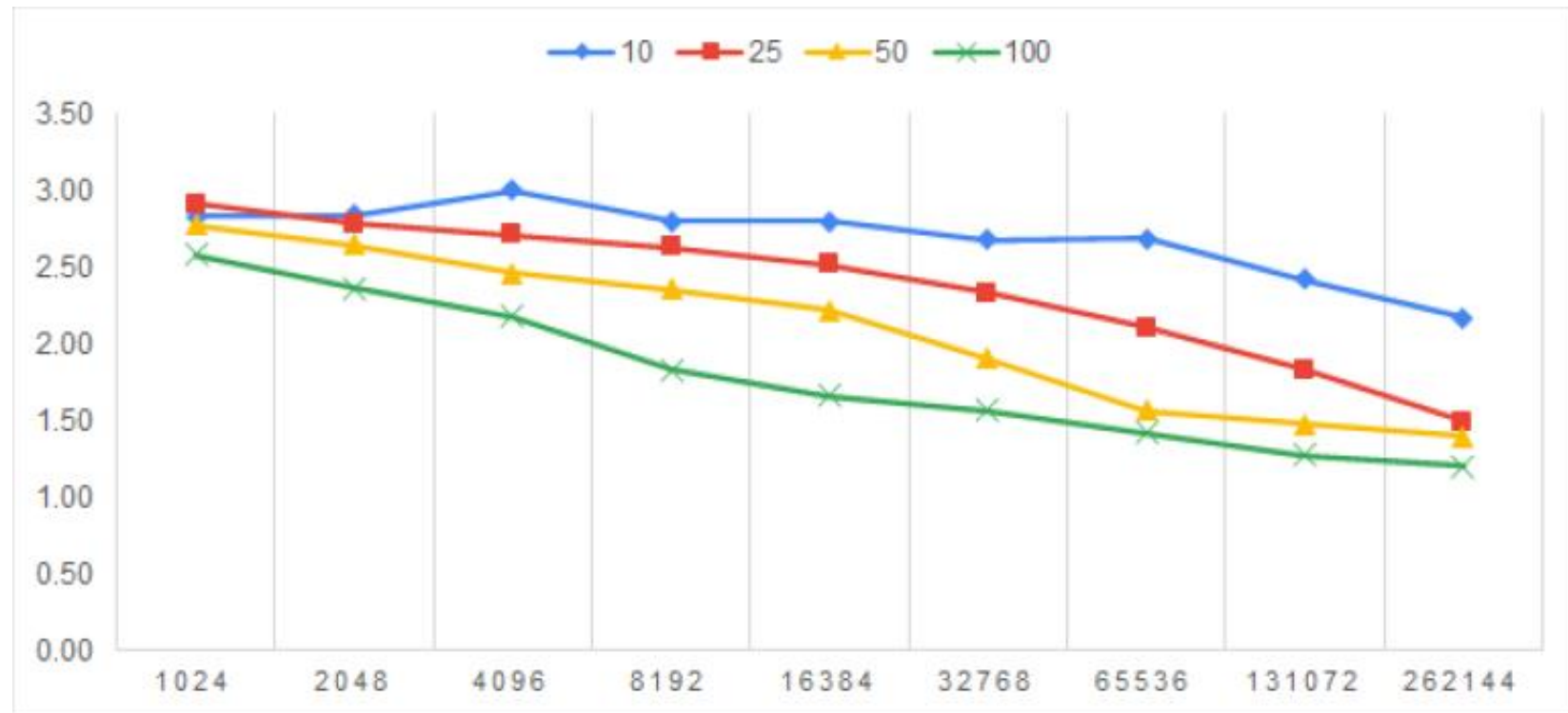
Observations

- ▶ Both **total execution time** and the **functions' time** increase proportionally with the increase of the number of alignment patterns & with the increase of the number of taxa.
- ▶ Increased **Alignment Patterns** => Increased Function's Execution Time.
- ▶ Increased **Taxa** => Too many recalls of functions => Increased Total Execution Time.
- ▶ Using AVX instructions => Good Acceleration of the Sequential version.

Hardware Performance

ZCU102 - PLF

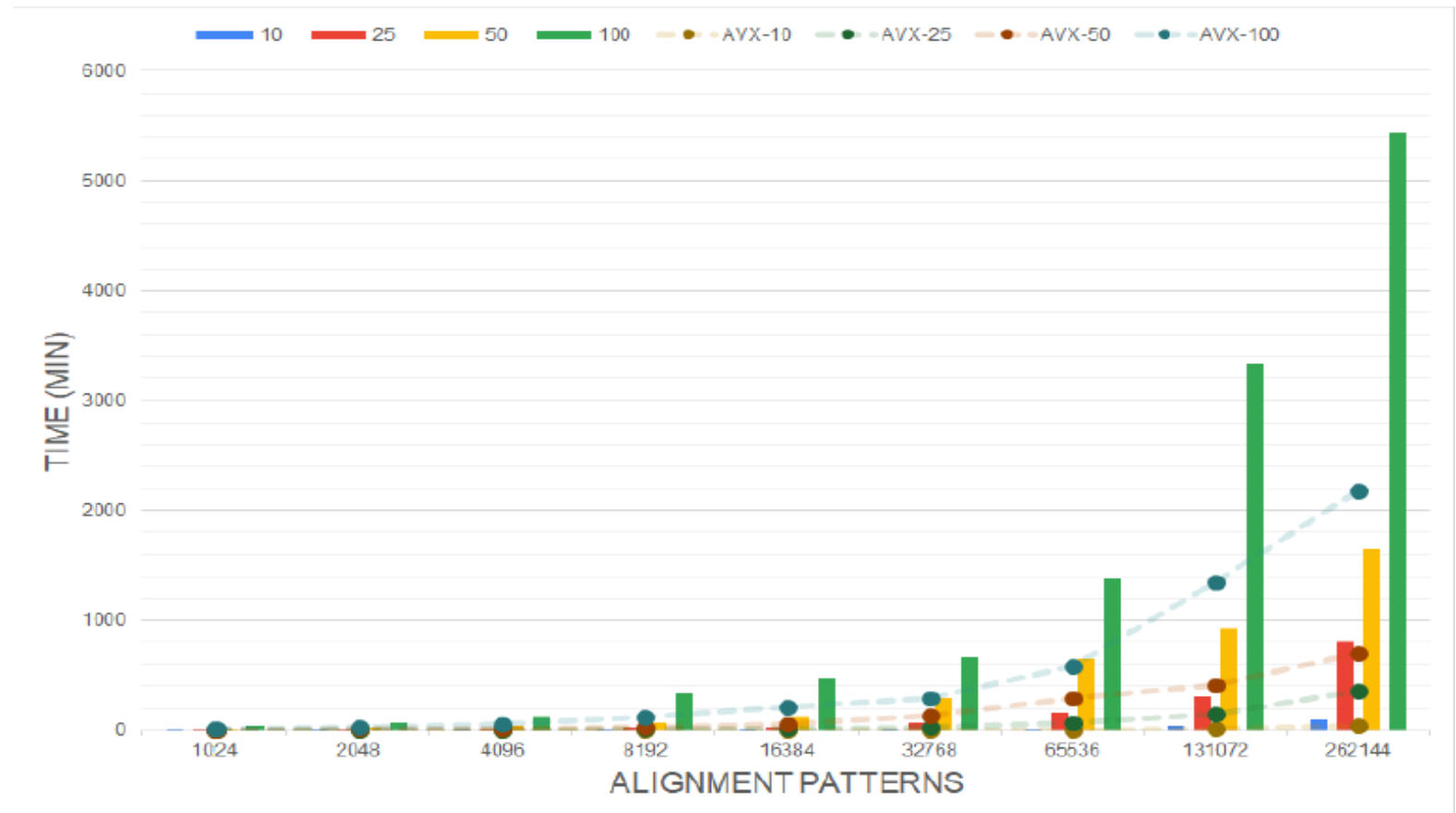
- $II = 160$
- $Clk = 100\text{ MHz}$



Hardware Performance

ZCU102 - PLF

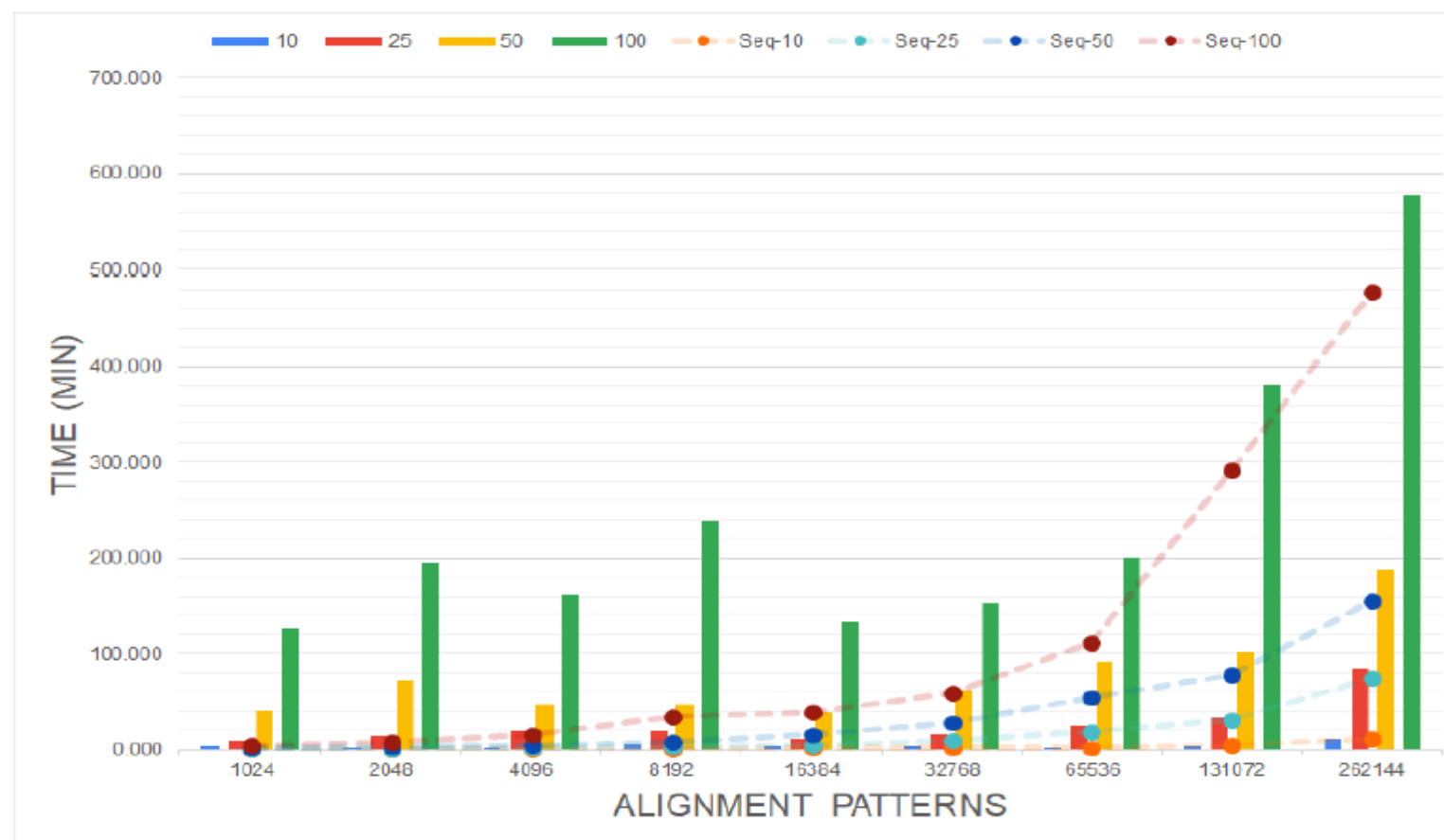
- $ll = 160$
- $Clk = 100\text{ MHz}$



Hardware Performance

AWS F1 - SumGAMMAPROT

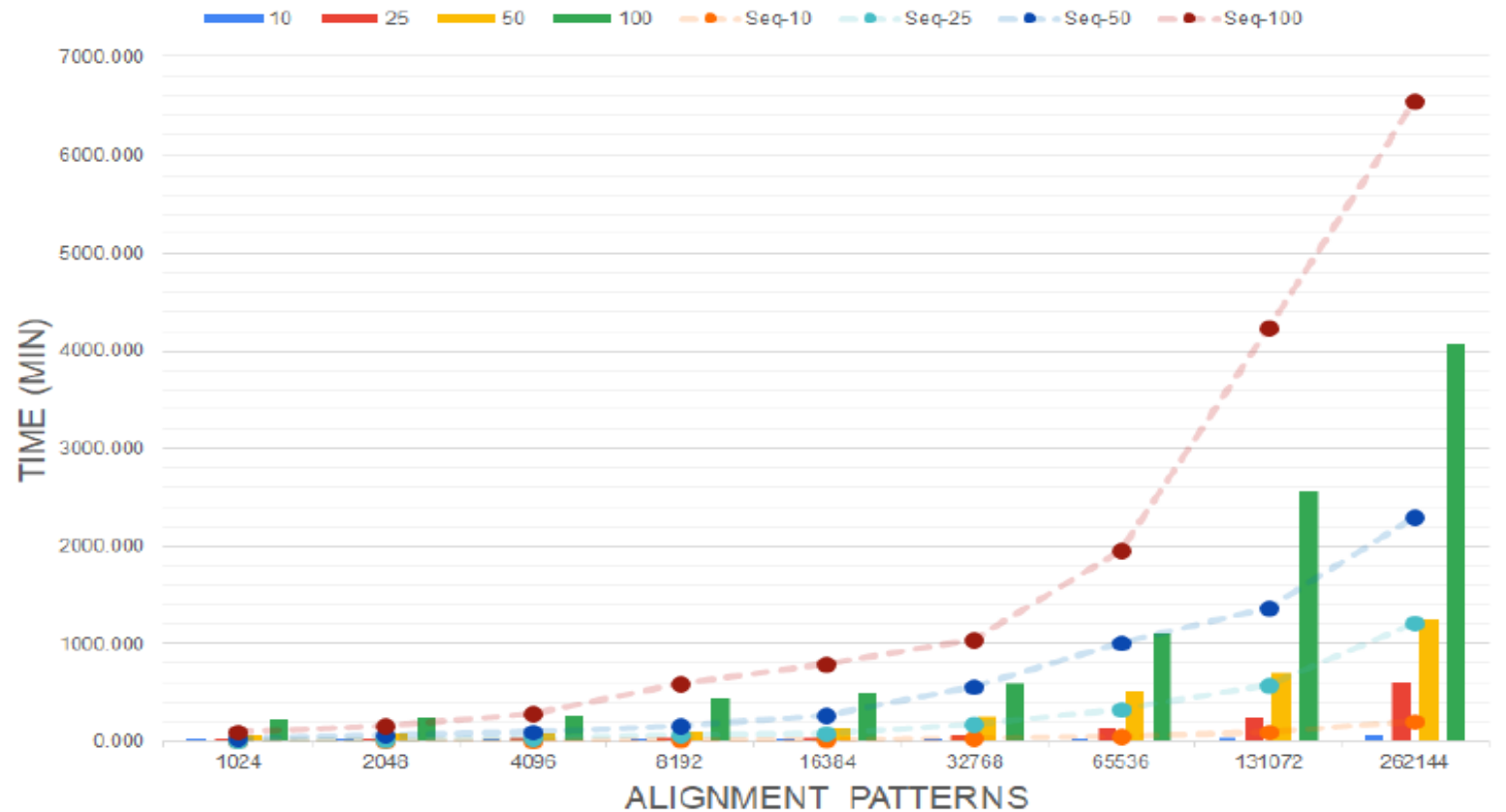
- $l_l = 10$
- $Clk = 350$ MHz



Hardware Performance

AWS F1 - PLF

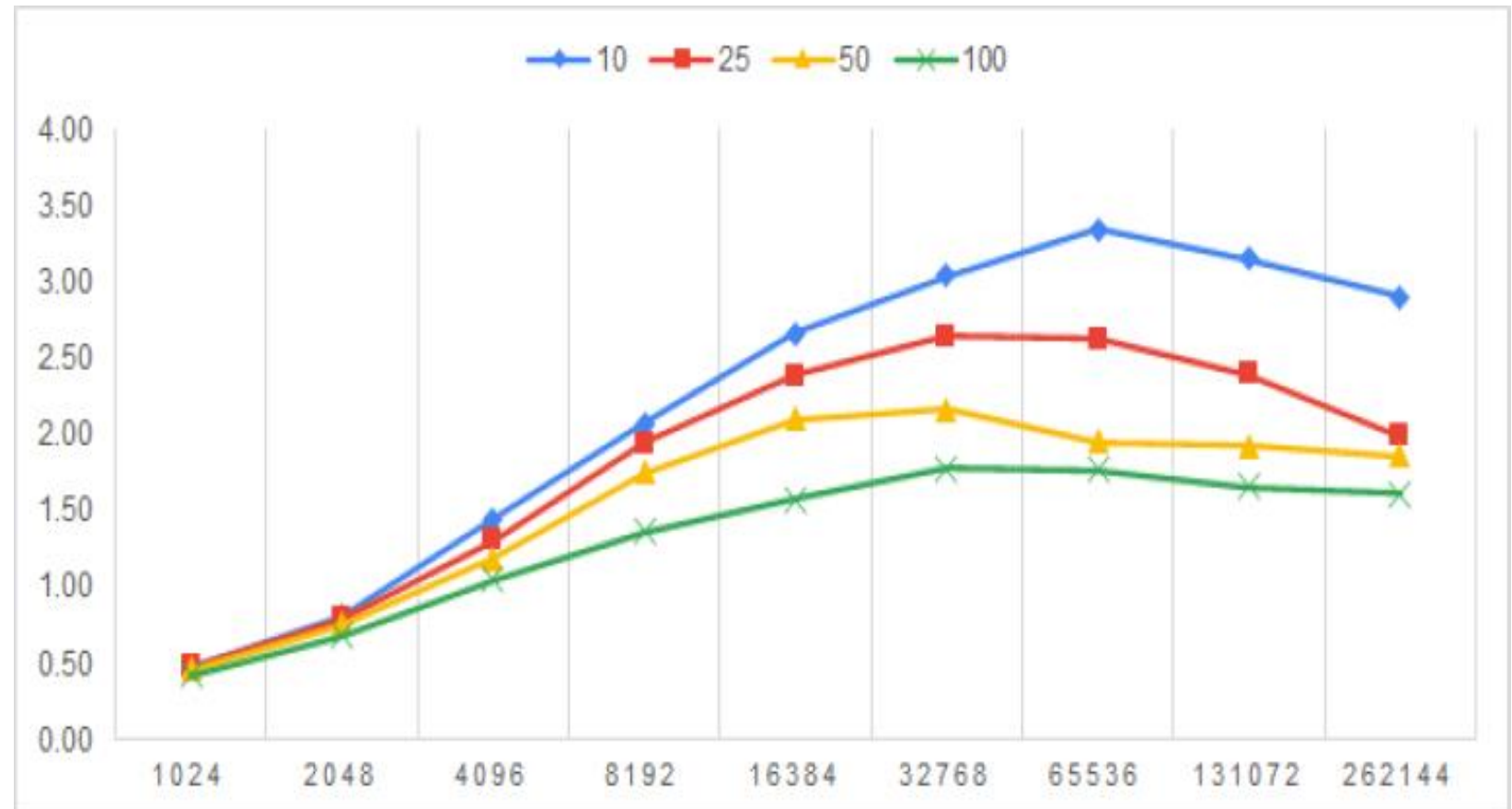
- $II = 160$
- $Clk = 150 \text{ MHz}$



Hardware Performance

AWS F1 - PLF

- $II = 160$
- $Clk = 150\text{ MHz}$

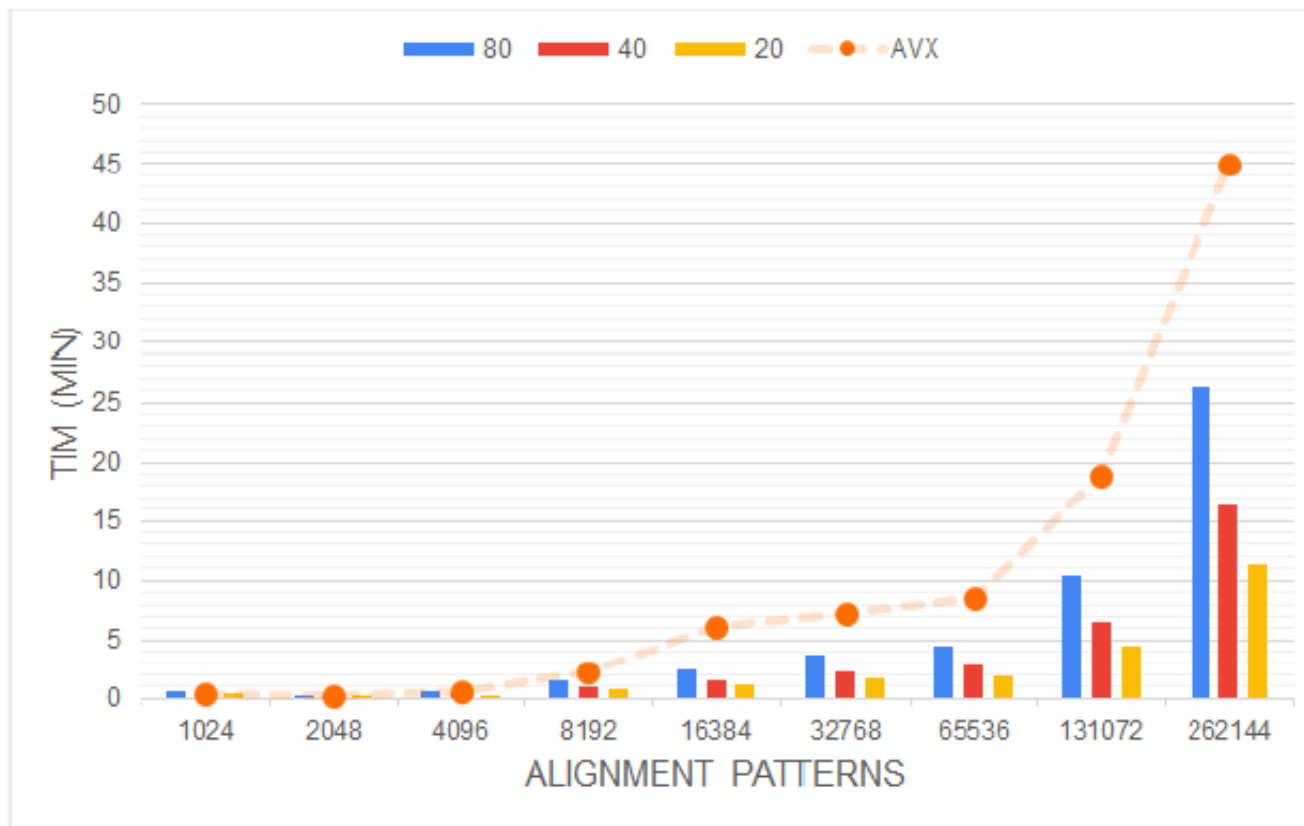


Performance Model

- We achieved to pass completely the synthesis process of our PLF design with $II = 80, 40, 20$.
- We designed a performance model that could simulate an approximate performance framework.
- This model counts in the time of data transfer and setup of kernel arguments.
- There is a comparison among the new performance and the AVX one which was our threshold.

PLF using 10 Taxa

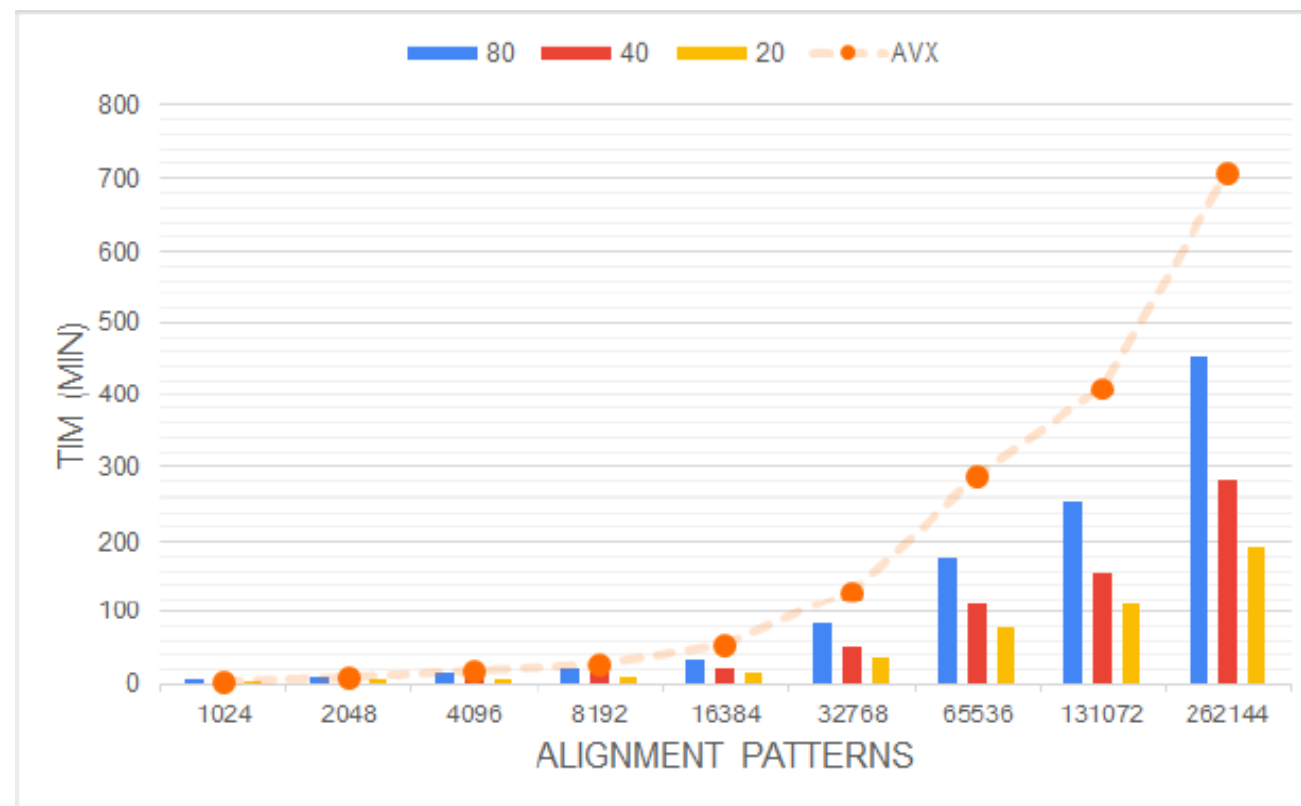
Performance Model



	1024	2048	4096	8192	16384	32768	65536	131072	262144
80	x0.70	x0.98	x1.24	x1.41	x2.54	x1.93	x1.92	x1.82	x1.71
40	x0.84	x1.28	x1.75	x2.13	x3.94	x3.06	x3.09	x2.93	x2.77
20	x0.93	x1.51	x2.21	x2.84	x5.45	x4.33	x4.43	x4.23	x4.00

PLF using 25 Taxa

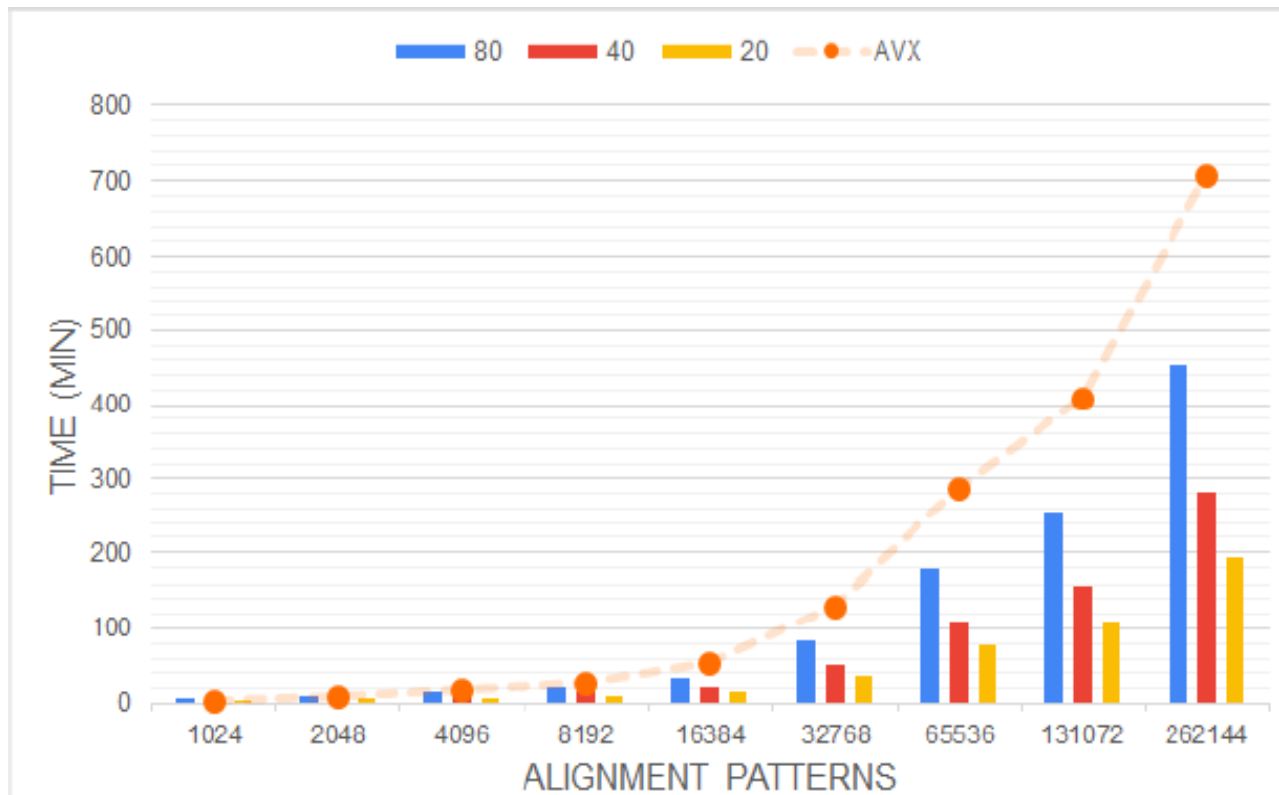
Performance Model



	1024	2048	4096	8192	16384	32768	65536	131072	262144
80	x0.70	x0.98	x1.22	x1.36	x1.70	x1.67	x1.66	x1.67	x1.59
40	x0.84	x1.28	x1.73	x2.05	x2.64	x2.65	x2.66	x2.69	x2.56
20	x0.93	x1.51	x2.18	x2.74	x3.65	x3.75	x3.81	x3.88	x3.71

PLF using 50 Taxa

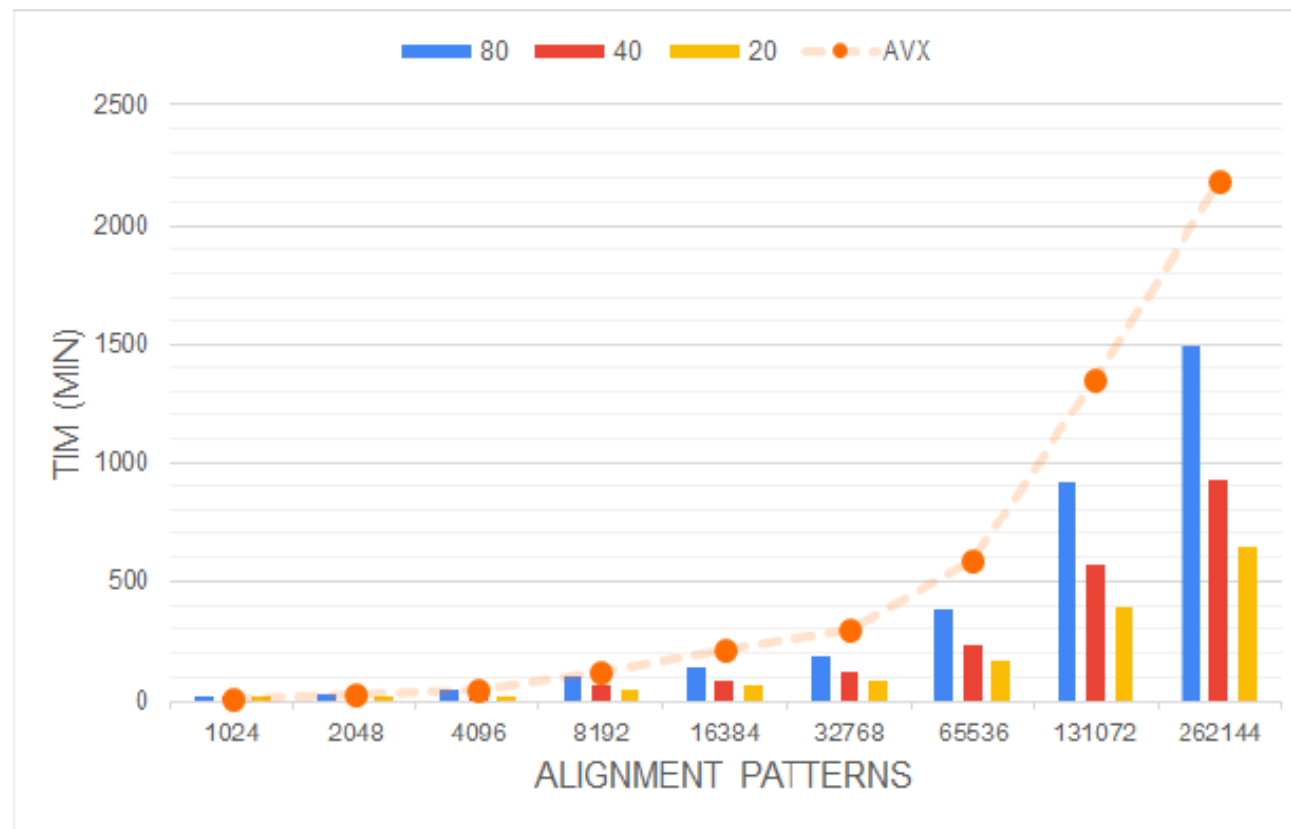
Performance Model



	1024	2048	4096	8192	16384	32768	65536	131072	262144
80	x0.69	x0.95	x1.18	x1.31	x1.60	x1.60	x1.61	x1.61	x1.56
40	x0.83	x1.24	x1.67	x1.96	x2.49	x2.54	x2.58	x2.59	x2.52
20	x0.93	x1.47	x2.11	x2.63	x3.44	x3.60	x3.70	x3.74	x3.65

PLF using 100 Taxa

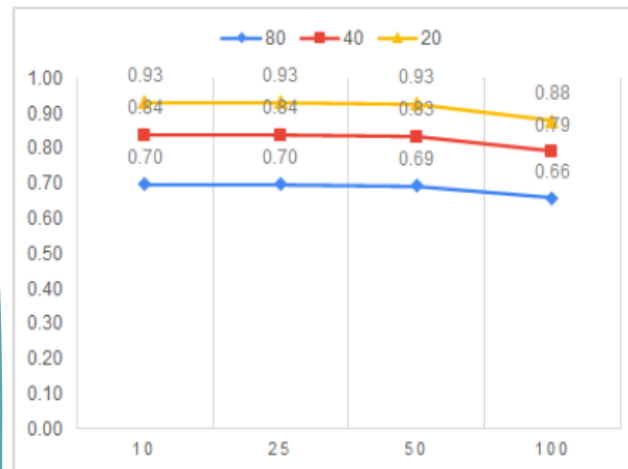
Performance Model



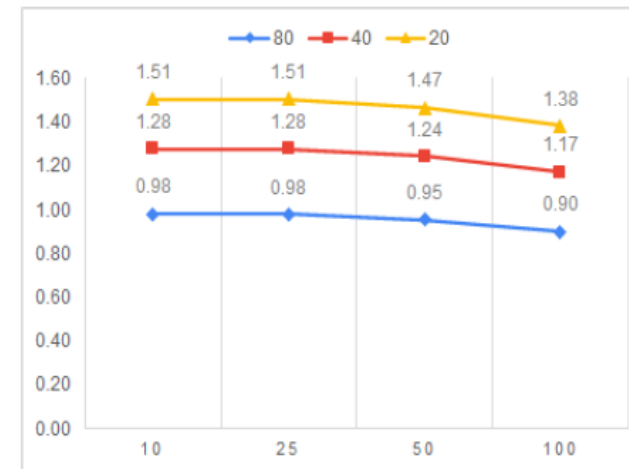
	1024	2048	4096	8192	16384	32768	65536	131072	262144
80	x0.66	x0.90	x1.10	x1.20	x1.55	x1.57	x1.53	x1.47	x1.47
40	x0.79	x1.17	x1.56	x1.80	x2.40	x2.49	x2.45	x2.38	x2.37
20	x0.88	x1.38	x1.97	x2.41	x3.32	x3.52	x3.51	x3.43	x3.43

SPEEDUP PERFORMANCE

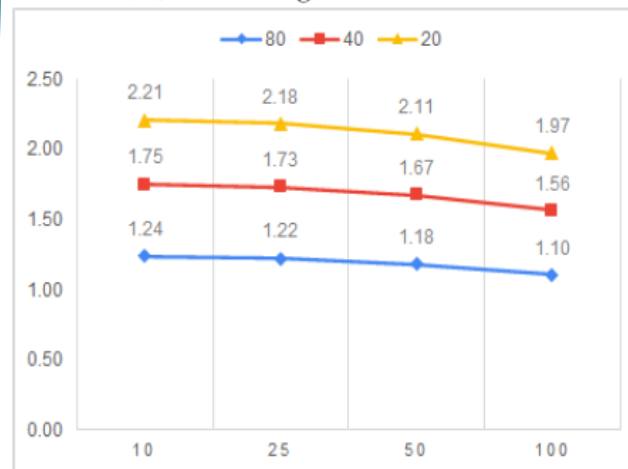
Performance Model



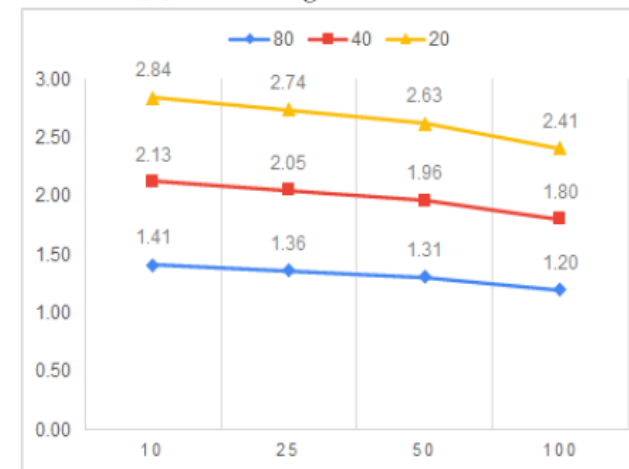
(A) 1024 Alignment Patterns



(B) 2048 Alignment Patterns



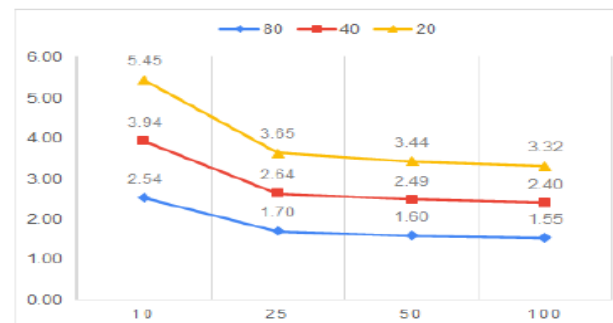
(C) 4096 Alignment Patterns



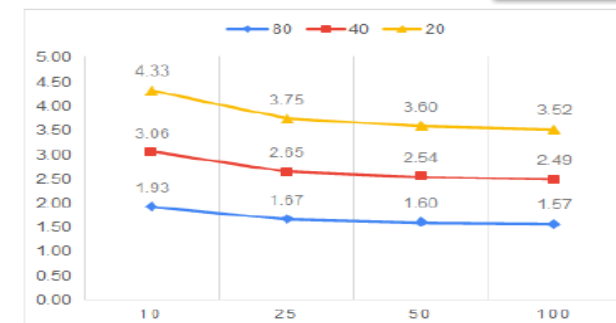
(D) 8192 Alignment Patterns

Performance Model

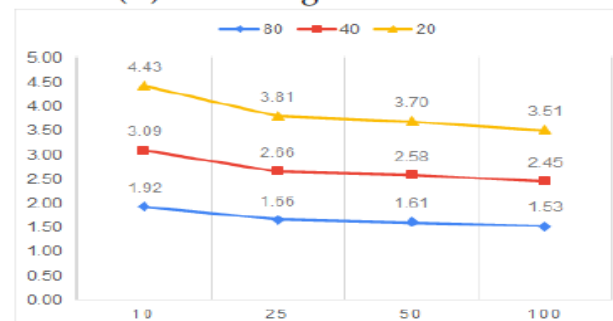
SPEEDUP PERFORMANCE



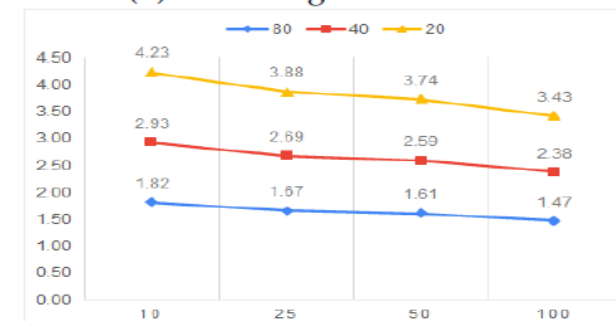
(A) 16384 Alignment Patterns



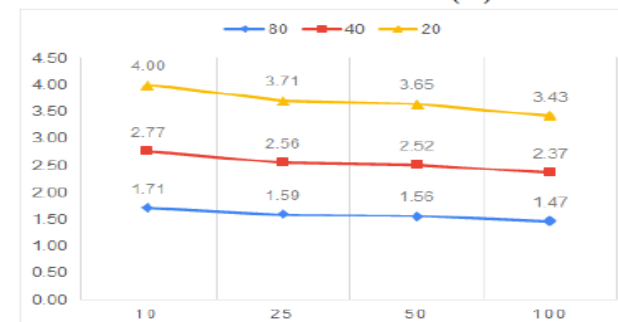
(B) 32768 Alignment Patterns



(C) 65536 Alignment Patterns



(D) 131072 Alignment Patterns



(E) 262144 Alignment Patterns

Analysis and Final Performance

- ✓ Both on **ZCU102** and on **AWS F1 Instance**, there is acceleration of the sequential version of **PLF** function using our totally completed kernels.
 - Still can not surpass threshold the AVX performance
- ✓ On **AWS F1 Instance** there is acceleration of the **SumGAMMAPROT** function, using our totally completed kernel.
 - Restriction is the big number of Alignment Patterns.
- ✓ On **AWS F1 instance**, there is acceleration of the **PLF** function surpassing the performance of AVX version using an implemented theoretical performance model.
 - It could not be placed & routed on any available platform.
- ✓ Subsequently to the above success, whole **RAxML** can be accelerated by an approximate **x1.47** factor.
 - This fact comes from the theoretical performance model.

Conclusions

Conclusions

In this thesis we **achieved** to :

- ▶ Examine and analyze the optimal performances of our systems, taking advantage of the specifications of the given platforms such as the clock frequencies, memory access patterns, and bandwidths.
- ▶ Design hardware kernels with as far as possible minimum resources of the targeted platforms.
- ▶ Accelerate the initial functions (both sequential and AVX versions), using these kernels, by a significant factor.
- ▶ Propose one of the least implemented accelerators of an algorithm that targets the usage and processing of amino acids data.

Future Work

Optimizations :

- ▶ A Better and sufficient technique for the transfer of the data from Host to Device and vice versa.
- ▶ A better way of recalling the kernel and transfer data without time losses during the total run of RAxML.
- ▶ Use a larger platform to achieve the optimal II and export the conclusion for the accelerators.
- ▶ Gather real data sets from scientists to run them and show the performance of the accelerators.
- ▶ Integration of the other two cases (tip-tip & tip-inner) of these functions into the kernels.

Thank You!
Any Questions?