Acta Universitatis Sapientiae

Informatica

Volume 7, Number 2, 2015

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Contents

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 125–142

DOI: 10.1515/ausi-2015-0015

A strategy for elliptic curve primality proving

Gyöngyvér KISS
Eötvös Loránd University
Budapest
email: kissgyongyver@gmail.com

Abstract. This paper deals with an implementation of the elliptic curve primality proving (ECPP) algorithm of Atkin and Morain. As the ECPP algorithm is not deterministic, we are developing a strategy to avoid certain situations in which the original implementation could get stuck and to get closer to the situation where the probability that the algorithm terminates successfully is 1. We apply heuristics and tricks in order to test the strategy in our implementation in Magma on numbers of up to 7000 decimal digits and collect data to show the advantages over previous implementations in practice.

1 Introduction

The elliptic curve primality proving (ECPP) algorithm of Atkin and Morain [1] starts from an input probable prime n and is called recursively on probable primes of decreasing size in order to reach a probable prime whose primality can be easily verified. In the paper of Atkin and Morain an implementation of the ECPP algorithm is described. Further descriptions can be found in the work of Lenstra, Lenstra [6] and Morain [7]. The aim of one recursive step of these implementations is to find a new probable prime: the input for the next recursive step. As not all numbers are equally suitable, it is useful to produce

Computing Classification System 1998: F.2.1
Mathematics Subject Classification 2010: 11Y11, 11A51, 14H52
Key words and phrases: ECPP, elliptic curves, primality proving

more than one probable prime in one step and select ‘the best one’. In this paper we describe an implementation in the Computational Algebra System Magma [2] that applies a strategy to control the selection of the input for the next step. We refer to the original implementations (see in [7], [2]) as ECPP and our modified implementation as Modified-ECPP in the rest of the paper.

This is the second implementation of Modified-ECPP within the confines of a project that, besides investigating different strategies on the ’Downrun’, deals with the heuristic running time, and questions and assumptions related to this topic. The details of this project can be found in the work of Farkas, Kallós, Kiss [4], Járai, Kiss [5] and Bosma, Cator, Járai and Kiss [3]. One goal is to replace the actual ECPP implementation that is used in Magma.

In the rest of the paper $\ln^k n$ shall denote $(\ln n)^k$, $\ln\ln^k n$ shall denote $(\ln\ln n)^k$, and so on.

2 Implementation of ECPP: details and tricks

The description, analysis, and implementation details of ECPP and the theoretical background of the statements below can be found in our paper [3]. Here we give a brief outline.

The input of the algorithm is a large probable prime n; this is a positive integer that has passed some probabilistic primality tests. The aim is to provide a *proof* for its primality.

Algorithm 1 : ECPP

(P) Starting with $n_0 = n$, build up a sequence of probable primes $n_0, n_1, \ldots, n_l$, such that n_l is small enough for the primality of n_l to be recognized easily.

(F) For each of the integers n_i with $i = 0, 1, \ldots, l-1$, build up the elements of the proof (consisting of pairs (E_i, P_i) of an elliptic curve and a point on it).

(V) Verify that n_l is prime and verify the other steps of the primality proof (by showing that P_i has order n_{i+1} on E_i modulo n_i and that n_{i+1} exceeds $(n_i^{\frac{1}{4}} + 1)^2$, for $i = l-1, l-2, \ldots, 0$).

In this paper we only deal with stage (P). On one hand the other two stages are not a risk in the sense that they always terminate successfully, on the other hand the other stages can be highly parallelized.

Next we describe the stages of algorithm (D), with input set n_{i_j}, $j = 1 \dots t$ briefly; this is part of stage (P), but the actual relation to it is described later on.

Algorithm 2 : Downrun

(D) *For each* n_{i_j} *select a set of negative discriminants* D *that is suitable for* n_{i_j} *such that for integers* u *(determined by* D*),* $n_{i_j} + 1 - u$ *is the product of small primes and a probable prime* q *that exceeds* $(\sqrt[4]{n_{i_j}} + 1)^2$*. This is done as follows:*

- *(1) Find a list of discriminants* D *for each* n_{i_j} *for which the binary quadratic form* $n_{i_j}x^2 + \mathsf{B}xy + \frac{\mathsf{B}^2 - \mathsf{D}}{4n_{i_j}}y^2$*, where* $\mathsf{B}^2 \equiv \mathsf{D} \bmod n_{i_j}$*, provides* ν *with* $\nu \cdot \overline{\nu} = n_{i_j}$ *(cf.* [3, 6]*). Store the pairs* $(\mathsf{D}, \pm u)$ *for each* n_{i_j} *where* $u = \nu + \overline{\nu}$*.*
- *(2) From the list of* $(\mathsf{D}, \pm u)$ *from the previous step, select those for which* $m = n_{i_j} + 1 - u$ *can be factored as* $m = q \cdot f$ *with* q *a probable prime exceeding* $(n_{i_j}^{\frac{1}{4}} + 1)^2$ *and* f *is completely factored.*
- *(3) Store the set of tuples* (D, u, q) *for each* n_{i_j}*.*
- *(4) From the possible choices of* (D, u, q) *select a subset that will be the input for the next iteration.*

An iteration step (D) in our Modified-ECPP differs slightly from an iteration in the original ECPP.

In the original ECPP, in general the i-th iteration of (P) consists of running (D) with only one input n_i and the outcome is just one q. At the first successful q, the i-th iteration returns and q becomes n_{i+1}, the input of the $(i + 1)$-th iteration. If there is no new q after running through a predefined discriminant set, it backtracks and returns n_{i-1} as n_{i+1}. What actually happens after backtracking is that it goes further on the discriminants starting (D_{i+1}) right after the last successful D.

In the case of Modified-ECPP, the i-th iteration of (P) can have one or more inputs n_{i_j}. An initial (D) runs on the input set n_{i_j}. This can result in zero, one or more q-s. If there is at least one q, the iteration returns all of them and they become the inputs of the $(i + 1)$-th iteration. If there is no new q after processing the discriminants up to certain limits, we select the *best* from a list of n_k, the results of the previous iterations. The selected n_k becomes the only input of a new run of (D), running on a new set of discriminants, or factoring the existing m-s harder. Note that this is still the i-th iteration. The iteration only returns when there is at least one new q produced. If not ambiguous, we will denote n_{i_j} with n_k.

We have chosen this generic way above to describe (D), because it is applicable to both Modified-ECPP that runs more (D)-s on a set of numbers and returns a set of q-s and to ECPP that runs one (D) on one input and either returns the first successful q or backtracks.

2.1 Parameters

There are three main parameters that we use in the algorithm to control the Downrun, introduced in the work of Atkin and Morain [1]. As they play a major role in our strategy we describe them here briefly.

Parameter d – In $(D1)$ we select a set of fundamental discriminants D. In order to control the size of this set, we apply an upper bound d on the size of the discriminants. Unlike ECPP, an iteration of Modified-ECPP does not stop at the first successful discriminant, but processes all of them up to d. In stages $(D1)$ and $(D2)$ we need to perform a reduction for essentially every discriminant that is suitable for the current input, as well as a factorization and a primality test on each m and q that were produced processing the suitable discriminants; thus the number of the selected discriminants has a huge impact on the running time.

Parameter s – In stage $(D1)$ we also have to extract the square root of discriminants D modulo n_i, which can be done faster if we extract the modular square roots of all the prime divisors of the D-s instead. An upper bound s on the size of the factors of the discriminants can control the size of the set on which we have to perform the root extraction and the size of s also has an effect on the number of the discriminants, as we throw away everything that is not s-smooth.

Parameter b – One of the bottlenecks is factoring the m-s, performed in stage $(D2)$. There are two ways to control the running time of the factoring. The first one, mentioned above, is to control the size of the discriminant set through d and s; but we can also restrict the set of primes that we use to factor the m-s. The bound on these primes is b.

Most of the ECPP implementations use these parameters as fixed limits. For example in [1], d is taken to be 10^6, for practical purposes.

As we mentioned earlier, Modified-ECPP deals with a set of q-s during the Downrun. In our case, we control the size of this set with the above parameters.

The parameters depend on n_k; by our choice these parameters will all be taken of the form

$$a \ln^{c_1} n_k \ln\ln^{c_2} n_k.$$

For this reason, in the rest of the paper $d(n_k)$, $s(n_k)$ and $b(n_k)$ shall denote them. Initially we provide the values a, c_1 and c_2 for them. Further on if an iteration does not provide new q-s, backtracking and repetition (when we force the same n_k with bigger and bigger parameters) is also possible. In these cases the parameters are increased by multiplying with a constant c, which is also a parameter, while the exponents remain unchanged.

2.2 Tricks

Some data used in the algorithm is independent of the choice of n_k, so it is possible to collect it in advance.

In stage (D1) we need a list of the $s(n_k)$-smooth discriminants up to $d(n_k)$, that will be suitable for n_k. We check two necessary but not sufficient conditions; Jacobi symbols $\left(\frac{D}{n_k}\right) = 1$ and $\left(\frac{n_k}{p}\right) = 1$ for each prime divisor p of D, cf. [1], in order to reduce the possibility of failure when reducing the quadratic form in (D1). We will refer to this check further on as the Jacobi symbol filter. We also have to extract the square root of the D-s $\mod n_k$; doing this for the prime divisors of the discriminants will be more efficient as many primes will occur for several discriminants. Both of these points suggest to store the discriminants together with their prime factors and the primes themselves in preprocessed files. For estimation purposes [3] we also need the class number of the discriminants, which will be preprocessed too.

This way we can run through the prime file up to $s(n_k)$, checking $\left(\frac{n_k}{p}\right) = 1$ and extract the square roots $\mod n_k$, and then collect those discriminants up to $d(n_k)$ which have only appropriate prime divisors and build up the square roots of them after checking $\left(\frac{D}{n_k}\right) = 1$. We have such a file of discriminants up to 10^9. As our initial value of $d(n_k)$ is

$$\frac{1}{16e^{2\cdot\gamma}} \ln^2 n_k \ln\ln^{-2} n_k,$$

running out of discriminants should not be a problem. The primes are collected up to $2.7 \cdot 10^9$.

In stage (D2) we have to factor the m-s in order to acquire the input for the next iteration. In our implementation we use Batch Trial Division [3] up to $2.7 \cdot 10^9$ and the Pollard ρ method after that. In Batch Trial Division one takes

GCDs of products of number that are to be factored with products of primes, a list of prime products is stored to avoid the time consuming multiplication of primes on the fly. We store pairs of prime products with size $2^t \cdot 10^6$ where $t = 0, \dots, 12$. If we use factorization limit b, the size of the product of the primes up to b is around e^b. As $2^{2^t \cdot 10^6} \asymp e^b$, we have $b \asymp 2^t \cdot 10^6 \cdot \ln 2$. This way we have a product from 0 up to around $2^t \cdot 10^6 \cdot \ln 2$ and from around $2^t \cdot 10^6 \cdot \ln 2$ up to around $2^{t+1} \cdot 10^6 \cdot \ln 2$ and they are both of size $2^t \cdot 10^6$ bits. This is useful, because depending on $b(n_k)$, we just have to find the right place in the file and get the appropriate product. We store pairs to have the possibility to start from 0 if it is the first factoring attempt, or from $2^t \cdot 10^6 \cdot \ln 2$ if we have already tried to factor below that.

2.3 Strategy

The detailed theoretical background of the strategy that the program uses is described in [3]. Here we list the most important notation and facts.

By $e_k = e\big(s(n_k), d(n_k)\big)$ we denote the number of m-s that we gain in stage (D2) after processing a set of $s(n_k)$-smooth discriminants up to $d(n_k)$. The expected value of e_k is

$$\bar{e}\big(s(n_k), d(n_k)\big) = \sum_D \frac{1}{h(D)},$$

where $h(D)$ denotes the class number of discriminant D.

After applying the Jacobi symbol filter, that uses arithmetic properties of n_k and its prime factors, this expected value changes to

$$\bar{e_k} = \bar{e}\big(n_k, s(n_k), d(n_k)\big) = \sum_D \frac{2^t}{h(D)},$$

where t is the number of different prime factors of D.

The expected number of m-s that split as required (that is, for which the second largest prime divisor is less than $b(n_k)$) is

$$\lambda_k = \lambda\big(s(n_k), d(n_k), b(n_k)\big) = e^{\gamma} \frac{\ln b(n_k)}{\ln n_k} e_k.$$

The progress we make is measured by the size difference between n_k and the q that is produced by (D) with n_k as input; the expected value of that 'gain' is

$$G_k = G\big(b(n_k)\big) = \ln b(n_k).$$

In the rest of the paper we denote by q numbers that have been produced already, and by q' we denote numbers that are not produced yet, but for which an estimation has been made for the amount of work needed to produce them for a given n_k. Note that q-s become n_k-s, when they become one of the inputs of the next iteration.

As we already mentioned, a single iteration can have one or more inputs and outputs. A larger number of inputs increases the probability of reaching the small primes, but on the other hand, it slows down the computation. Our aim is to find a balanced situation, where the implementation is reasonably fast but we still have a good chance to terminate successfully.

Choosing the best q

We saw that λ_k is the expected number the successful m-s, so the expected number of new q-s. From [3] we know that we are likely to succeed when λ_k exceeds 1.

On the other hand all the new q-s become the input of the next iteration and we run an initial (D) on them with certain (small) values of parameters s, d and b. If this does not provide at least one new q, then we select the *best* n_k as the input of the next (D).

The reason why we cannot choose the *best* n_k right away and have to run an initial (D) on the newly produced numbers, lies in our definition of *best*. There are two aspects to this.

First of all, we have to consider that the numbers are not equally appropriate. Applying the Jacobi symbol filters on n_{k_1} could filter out more discriminants than on n_{k_2}, so to produce the same number of new q-s we would have to process a larger number of discriminants or use bigger factoring bound for n_{k_1}. Higher bounds imply more execution time, as we need to deal with bigger discriminants, primes. So the first aspect is the time it takes to produce a given number of q-s on input n_k.

The second aspect is the size of the produced q-s. The smaller they are, the faster we get to the small primes.

The first aspect we can estimate with the help of λ; for estimating the running time to produce certain amount of q-s, we collect some actual running times to see how the numbers behave. This information can be collected from the initial (D) runs. The parameters of these initial runs are chosen as follows:

$$s_0(n_k) = \lambda_0 \cdot \frac{1}{2 \cdot e^{\gamma}} \ln n_i \ln \ln^{-1} n_k$$

$$d_0(n_k) = \lambda_0^2 \cdot \frac{1}{4 \cdot e^{2\cdot\gamma}} \ln^2 n_i \ln\ln^{-2} n_k$$

$$b_0(n_k) = \ln^2 n_k$$

The choice for $s_0(n_k)$ comes from taking $\lambda = e^{\gamma} \frac{\ln b(n_k)}{\ln n_k} e_k$ if we suppose that $\lambda = \lambda_0$ is a parameter and $s(n_k) \approx e_k$; we take $d(n_k)$ equal to $s^2(n_k)$ despite of [3], where we state that it is better to keep the value of s below $\sqrt{d}$, because in practice, on small numbers, taking $s < \sqrt{d}$ led to some difficulties.

During the initial runs we store the time needed for extracting the modular square roots, for the reduction algorithm and for factoring, and we also see how many new q-s are produced. With this information, we can estimate the time needed when we increase s, d or b separately. For the first two we can estimate the increment in the value of $\bar{e_k}$ when the smoothness bound for the discriminants up to $d_0(n_k)$ is increased from $s_0(n_k)$ to $c \cdot s_0(n_k)$, and separately, the effect of including the $s_0(n_k)$-smooth discriminants up to $c \cdot d_0(n_k)$ rather than $d_0(n_k)$. We filter out the discriminants that are appropriate for n_k and determine $\bar{e_k}$ in both cases. We can directly determine the value of $b(n_k)$ using the actual e_k.

Now we can compute λ_k in all three cases and we can also estimate the time t_k it would take in all three cases to execute (D) with the estimated values of one parameter while the other two remain unchanged. Then we can store the different $\frac{t_k}{\lambda_k}$ values.

We now know the expected number of q'-s resulting from increasing one of the parameters, but we do not know the expected size of them. This can be determined with the help of the gain function G_k, which depends only on $b(n_k)$, the factorization effort on m. From this we can see that we gain q'-s with the smallest expected size if we increase b, also if we increase s or d, the expected size of the q' that we gain are the same. After incrementing s or d, we want to know how much effort it takes to reduce q' further: what is the average work per bit needed to decrease q'? This we can estimate from the previous iterations. After multiplication by the estimated size differences, we obtain a value, a_k-s for the expected effort of reducing q'-s to the size of the smallest one. Then compare the values $\frac{t_k}{\lambda_k} + a_k$ and select that parameters for which this value is minimal. We denote this minimal value by mt_k, for the given n_k. Now the *best* number is the n_k for which this value mt_k is the smallest.

Note that the running time of the three bottleneck subroutines (extracting square roots modulo n_k, quadratic form reduction, integer factoring) depends (via the three parameters) only on n_k, so it is possible to express these running time as a function of n_k. Estimating the running time of square root extraction modulo a prime and of form reduction is fairly easy, because we can measure the running time of a single such operation and multiply by the number of times we need to perform them (which is the number of primes in n_k and the number of successful discriminants, respectively). In the case of factoring the running time of Batch Trial Division is linear neither to the number of m-s nor to the number of primes used, but as we do not use huge amount of m-s or primes simultaneously, linear approximation works well in practice. For $b(n_k) = 2^t \cdot 10^6 \cdot \ln 2$ we double the expected time if we increase t to $t+1$ as we have to deal with products of twice the size.

If no new q-s are produced, we need to be able to backtrack. We keep a window with a certain number of n_k-s for which we store all the data that is necessary to continue using this value of n_k if turns out to be the *best*. Newly found n_k-s are always going to the window, and if the number of the n_k-s in the window exceeds a limit, we throw away the *worst* ones. It is not possible to backtrack to a number that is not in the window anymore. We compute the expected work to decrease one bit for values n_k in this window and update it after each iteration.

Note that while processing a number, the running times that are stored will be updated with the new data; estimation takes place directly after processing, so we can base our decision on an up-to-date set of data.

Resulting algorithm

Now we describe briefly how we make our estimates and decisions, followed by the description of one iteration. For the notation used we refer to the previous subsection.

Algorithm 3 : Estimation Algorithm

(E) The Estimation Algorithm determines the value of mt_{i_j} *for a value of* n_{i_j}*. The input is* $s(n_{i_j})$*,* $d(n_{i_j})$ *and* $b(n_{i_j})$*, the values of the main parameters of the previous call of* (D) *on* n_{i_j} *as input.*

(1) Determine the effect of increasing s *or* d *by computing the values of* $e(n_{i_j}, c \cdot s(n_{i_j}), d(n_{i_j}))$*, and* $e(n_k, s(n_{i_j}), c \cdot d(n_{i_j}))$*: collect the* $c \cdot s(n_{i_j})$*-smooth discriminants up to* $d(n_{i_j})$ *and* $s(n_{i_j})$*-smooth discriminants up to* $c \cdot d(n_{i_j})$*, that are appropriate for* n_{i_j}*.*

(2) *From the actual running times stored for* n_{i_j}, *determine* t_{i_j} *for* $c \cdot s(n_{i_j})$, $c \cdot d(n_{i_j})$, $2^t \cdot b(n_{i_j})$.

(3) *Compute the expected gain* G_{i_j}, *and compute the values* a_{i_j} *for each parameter.*

(4) *Determine and store the value of* mt_{i_j} *together with the corresponding lists of discriminants and primes.*

Algorithm 4 : Modified-ECPP Iteration Step

(P) *With a set of one or more* n_{i_j}*-s as input, one iteration of the Modified-ECPP Algorithm runs until it finds one ore more new* q*-s that become the inputs for the next iteration. The following steps are carried out.*

(1) *Run* (D) *on each* n_{i_j} *with* $s_0(n_{i_j})$, $d_0(n_{i_j})$ *and* $b_0(n_{i_j})$.

(2) *Run* (E) *on each* n_{i_j} *in order to determine* mt_{i_j}, *with the data collected in the previous step on the running times. Add the* n_{i_j}*-s to the window. If we obtain new* q*-s go to* (1), *else to* (3)

(3) *Reorder the window by the values* mt_k *(all of them are up to date).*

(4) *Pick the best as* n_{i_j} *and run* (D) *on it.*

(5) *Run* (E) *on the selected* n_{i_j} *with the data collected in the previous step.*

(6) *If we have new* q*-s go to* (1), *else to* (3)

We list the additional important parameters, besides the ones mentioned in the previous sections.

Parameter λ_0 – In (1) this parameter provides the initial value of λ. If it is too big, (1) becomes too slow, and also the process would result in too many new q-s and the tree of n_k-s would expand too much. On the other hand if it is too small, we cannot collect realistic data about the running time. It seems to be appropriate to keep this value between 1/3 and 1/2.

Parameter c – In (E) we take $c \cdot s(n_k)$, $c \cdot d(n_k)$ as next values of these parameters. If this is too big, running (D) becomes too slow, if it is too small, we spend to much time on administration because we take too small steps. The value seems to be appropriate between 3/2 and 3.

Parameter α – In (E) we increase the value of λ with $\delta\lambda$. We require a certain lower bound on $\delta\lambda$, that is α. If $\delta\lambda$ is above α the value of c is acceptable, below it we have to multiply c with $\alpha/(\delta\lambda)$. We use $\alpha = 0.25$;

Parameter $wSize$ – This parameter denotes the size of the window, that we have mentioned above. We have to be careful with it because if we store too big window, the program becomes slow, otherwise, if it is too small, it is possible that the strategy would select a node that is outside of the window, so we restrain it. This parameter is of the form $\ln n/(c \ln\ln n)$.

3 Analysis of the strategy and running times

In the main experiment we tested the strategy with around 200 numbers each for $k{\cdot}1000$ decimal digits, with $k = 1, 2, \dots, 7$. We ran various other experiment on running time and on the strategy. We have produced many graphs, but presenting all of them here is impossible. They can be downloaded from the page `http://www.math.ru.nl/~gykiss`. In case our graphs cannot present data for different sizes of numbers, we will always display the graph for the largest size for which data are available. The experiments were run on computers with Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz processors and 8 GB memory.

Running times

Figure 1 shows the running times for the main experiment of the algorithm. The number of decimal digits of the input numbers appears on the x-axis on both graphs, while the y-axis indicates the running time in seconds on the first graph, the average running times of the algorithm on the same numbers; for $k = 1, 2, 3, \dots, 7$ vertically the average of the running times for the numbers of $k \cdot 1000$ digits is indicated as a single dot on the second graph.

Applying a Least Squares linear approximation method, we found that the best fit for the running time on a logarithmic scale is given by the line $y = 3.86 \cdot x - 21.00$. This is displayed in Figure 2.

It is possible to play around with the running time, for example by changing the parameter c that is responsible for the size of the increase of the parameters s and d. We tested numbers from 1000 up to 3000 digits with $c = 1.5, 2, 2.5, 3, 3.5, 4$.

The effect on the running time can be seen in Figure 3. On the x-axis the different values of c are given, and on the y-axis the running time. We see that effect of changing c in this range is insignificant.

The time spent in a single iteration can be separated into administration time and the execution time. We consider filtering the discriminants for given n_k and the estimation process as part of the 'administration', and time spent on extracting modular square roots $\mod n_k$, reduction of forms, factoring

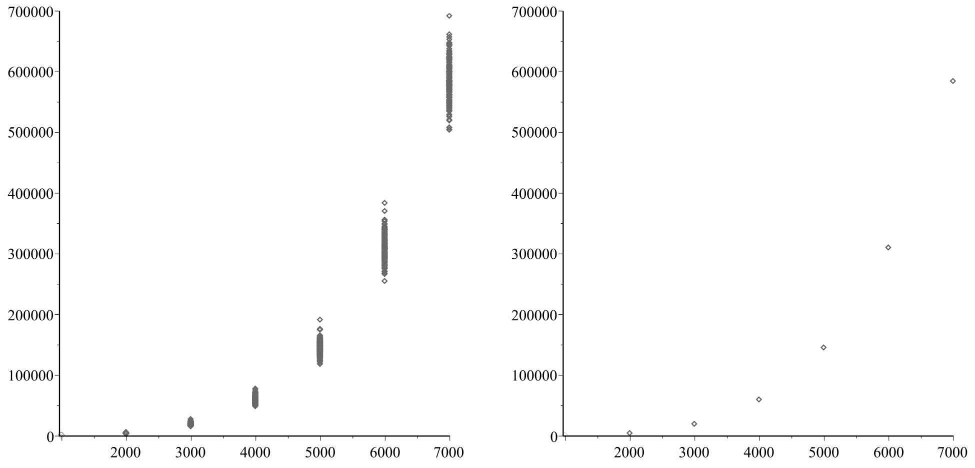


Figure 1: Running times

and Miller-Rabin tests on the new q-s as part of the 'execution'. We have to emphasize that in one iteration there will typically be several execution and administration steps, and these experiments are not meant to measure the time used in an iteration. Figure 4 shows the proportion of the total execution time per run to the total running time per run. The total running times are indicated on the x-axis, and on the y-axis the execution times for numbers from 1000 digits up to 7000 digits are displayed (together with the line $y = x$, for comparison). The clusters for the data for the numbers of the same size are clearly visible. As expected, the time spent on administration rather than execution is negligible.

Experiments on the strategy

In the analysis of the strategy we put emphasis on backtracks and repetitions. Repetitions and backtracks are very similar and occur for the same reason: they indicate that we could not provide a new q after executing the initial (D) as well as a run of (D) on the best available input (the initial run of D is only used to collect data on the number and is considered only as precomputation). In both cases we have to select a new value of n_k to continue on. The only difference is that when we backtrack we select a different number, whereas in a repetition the same number is selected again (because it is still 'best'). It is natural that repetitions occur frequently as we increase λ with around $\alpha = 0.25$ instead of 1, which means that one may expect to repeat the procedure as much

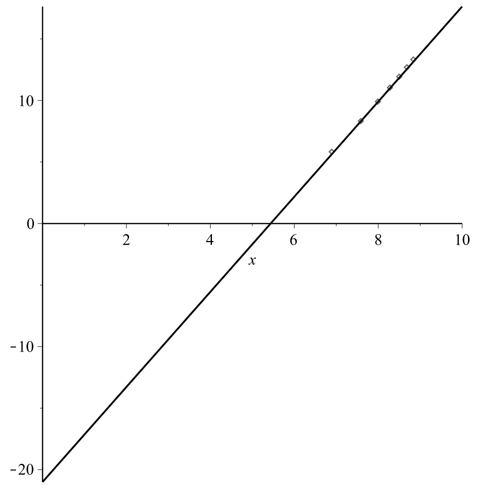


Figure 2: Average Running times on logarithmic scale

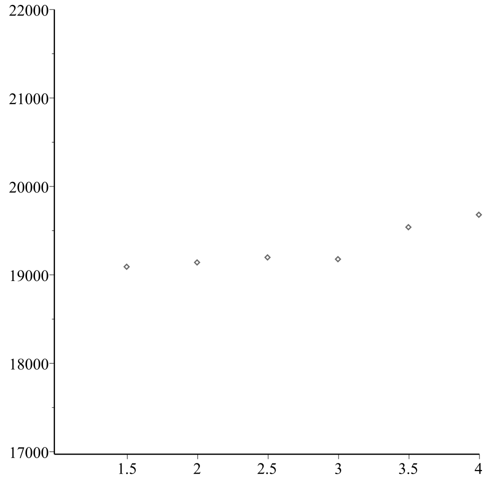


Figure 3: Changes of the running time if we change parameter c on 3000 digit numbers

as 4 times before producing a new q.

Note that when we backtrack, we may step either backwards or forward, as the window may contain numbers for both situations.

Note that the length of the path is not equal to the number of iterations, as we include each number on which (D) was ever called in the path, and in

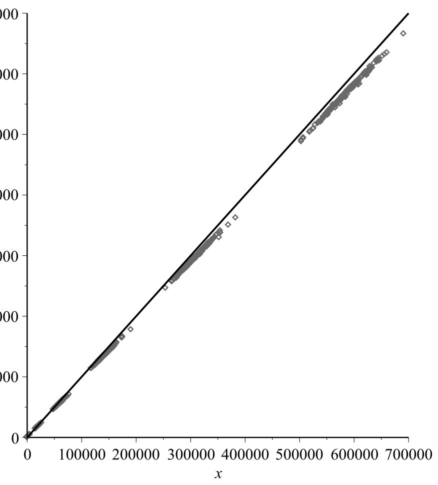


Figure 4: The proportion of the execution time compared to the total running time

one iteration (D) may be called several times. However, the number of the iterations is equal to the maximum *level*, as we consider numbers produced in the same iteration to be on the same level.

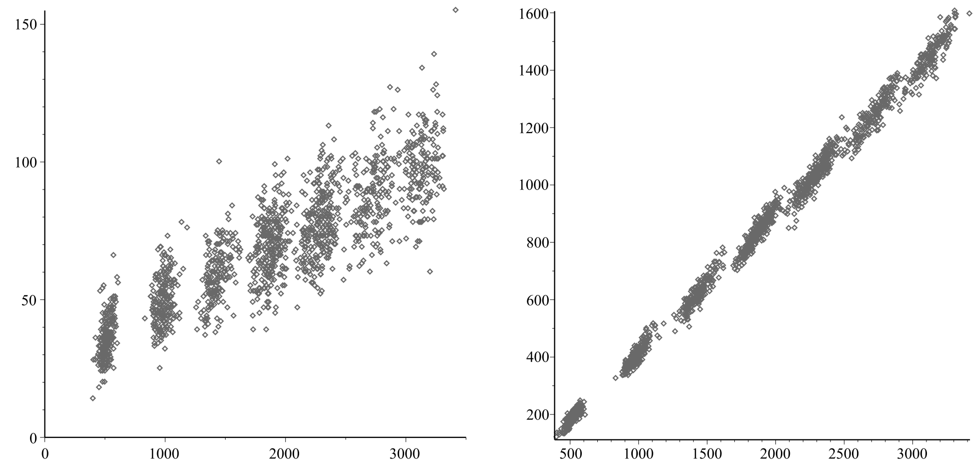


Figure 5: The number of backtracks as a function of the length of the paths

Figure 5 show the proportion of the number of backtracks and repetitions to the length of the path, for numbers of various sizes. On the x-axis the path lengths are indicated, on the y-axis, the number of backtracks (on the first graph) or repetitions (on the second graph). The length of the path is of order $O(\ln(n))$, and the graphs clearly show the 7 clusters corresponding to numbers of size $k \cdot 1000$, for $k = 1, \dots, 7$. From Figure 5 it is clear that we need backtracks during the Downrun, and as a rule of thumb, approximately once every 30 steps on the path. We can also see that around half of the path is repetition.

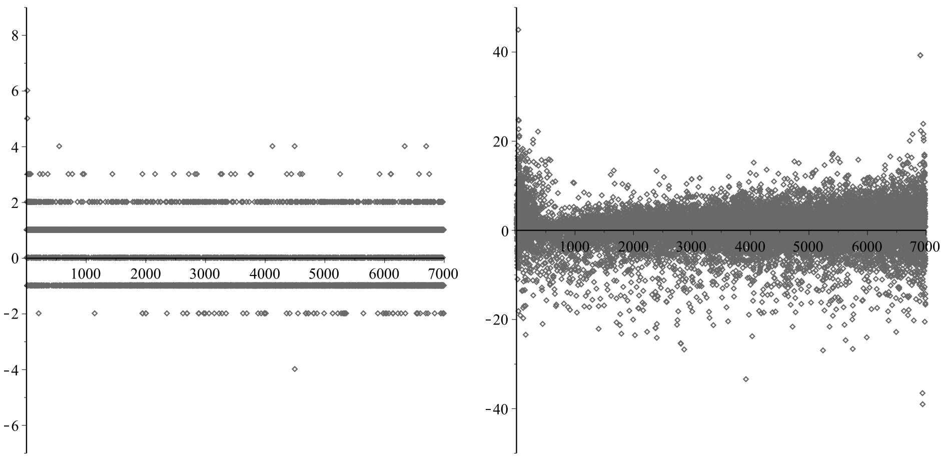


Figure 6: The level and size differences of backtracks

Besides knowing how often we backtrack in a run, we would like information on how far back we step (in terms of level or size) when backtracking. There should not be too big size or level differences, as in this case the effort invested in decreasing the size of the n_k-s is lost. In Figure 6 we present the level and size differences that occurred up to 7000 digits. The x-axis on both graphs indicates the size of the numbers from which we backtracked. On the y-axis on the graph on the left the level differences are indicated, and on the graph on the right the size differences (in number of digits). A positive number means that we stepped back, a negative number means that we stepped forward, 0 indicates that we stayed on the same level but we have selected a different number. It is clearly suggested by these graphs that neither the level nor the size differences depend on the size of the input; only for really small numbers, when the work that we loose with backtracking is small, the size differences can be relatively large. There are very few outliers for larger sizes.

The length of repetition sequences and how long they take, is also interest-

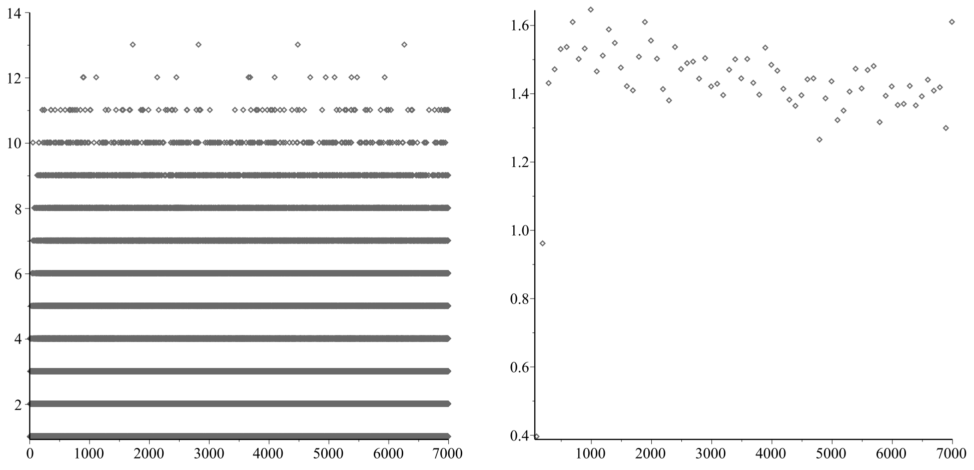


Figure 7: The length and proportion of the repetition sequences

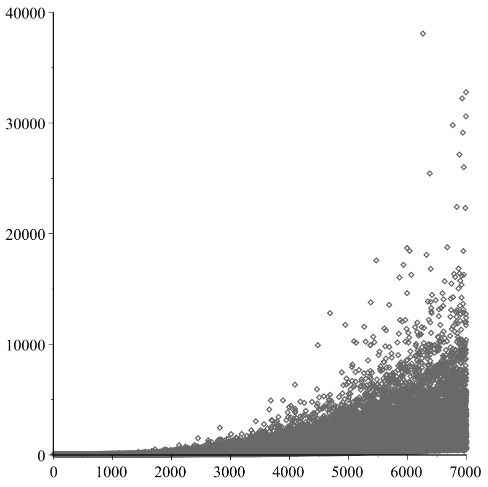


Figure 8: The time of the repetition sequences

ing. The result of our experiments on this can be seen in Figure 7 and Figure 8. In x-direction in each case the size of the number that is repeated is indicated. The y-axis in the first graph of Figure 7 indicates the length of the repetition sequence, in the second graph it indicates the percentage of the number of the repetitions on numbers of a certain size compared to the total amount of repetitions. In Figure 8 the y-axis indicates the time of the repetition sequences.

We see that the length of the repetition sequence does not really depend on the size of the input, it never exceeds 13. The percentage of the number of repetitions seems to be decreasing when the numbers are growing, though at 7000 digits the percentage is higher. The reason is that when we start on a number with 7000 digits we have to increase our effort on it until there are new q-s produced, as this is the only choice at that point. The time of course does depend on the size of the numbers, as for bigger numbers (D) takes longer; still the dependence is rather controlled with few outliers.

4 Conclusions and future improvements

To sum up the results from our experiments we make the following observations.

The evidence from experiments with numbers up to 7000 decimal digits is that the running time is below $o\big(\ln^4(n_0)\big)$. Of course there is no experimental way to show that this is asymptotically correct.

The proportion of the running time needed for administration of the strategy to be applied is small, which is necessary for the strategy to be useful.

Experiments show that in around half of the cases the strategy chooses to increment d (meaning: allow the use of larger discriminants) and in almost all other cases it chooses to increase s (that is: allow larger primes in the discriminants). For bigger numbers enlarging d is a bit more frequent, for smaller numbers enlarging s. Selection of b (that is: allow larger primes in the factorization of the m-s) hardly ever happens. As we saw that the number of repetitions does not exceed 13, the implementation should be able to work with numbers up to 10000 digits without running out of discriminants. After running out of discriminants it would be still possible to continue with increasing b (there is no upper limit to that parameter). Of course testing such big numbers would take very long.

The number of the backtracks and repetitions are proportional to the length of the path and seems to be independent from the size of the input.

The maximum level of backtracks and the lengths of repetitions seem to be the similar for different sizes of inputs. That is what we expect as the input selection depends on the estimated running times + the work that we have to do to reduce the new q-s to the same size. The work to reduce the new q-s; the $avgWork$, is growing for bigger inputs, but also the estimated running times, thus their relation should be the same. The size differences for backtracks are growing with bigger inputs, but the differences are negligible.

The time of the repetition sequences are growing also, but without too many extreme cases and of course for bigger input the execution time grows.

The overall conclusion is that the implementation seems to be working as it was intended, but there is still space for improvement. The goal is to provide an optimized implementation of the ECPP algorithm written in C that combines this strategy with a collection of highly optimized package written in C and Assembly.

References

[1] A. O. L. Atkin, F. Morain, Elliptic curves and primality proving, *Math. Comp.* **61,** 203 (1993) 29–68. ⇒125, 128, 129

[2] W. Bosma, J. Cannon, C. Playoust, The Magma algebra system I: the user language, *J. Symbolic Comput.*, **24,** 3 (1997) 235–265. ⇒126

[3] W. Bosma, E. Cator, A. Járai, Gy. Kiss, Primality proofs with elliptic curves: heuristics and analysis, *Ann. Univ. Sci. Budapest. Sect. Comput.*, in print ⇒ 126, 127, 129, 130, 131, 132

[4] G. Farkas, G. Kallós, Gy. Kiss, Large primes in generalized pascal triangles, *Acta Univ. Sapientiae*, *Informatica* **3,** 2 (2011) 158–171. ⇒126

[5] A. Járai, Gy. Kiss, Finding suitable paths for the elliptic curve primality proving algorithm, *Acta Univ. Sapientiae*, *Informatica* **5,** 1 (2013) 35–52. ⇒126

[6] A. K. Lenstra, H. W. Lenstra, Jr., Algorithms in number theory, in: *Handbook of Theoretical Computer Science, Vol. A. Algorithms and Complexity* (ed. J. van Leeuwen), Elsevier, 1990, pp. 673–716. ⇒125, 127

[7] F. Morain, Implementing the asymptotically fast version of the elliptic curve primality proving algorithm, *Math. Comp.* **76,** 257 (2007) 493–505. ⇒125, 126

Received: November 16, 2015 • Revised: December 13, 2015

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 143–150

DOI: 10.1515/ausi-2015-0016

Factoring multi power RSA moduli with a class of secret exponents

Omar AKCHICHE
Laboratory of Mathematics,
Cryptography and Mechanics, Fstm
University Hassan II of Casablanca,
Morocco
email: omar.akchiche@hotmail.com

Omar KHADIR
Laboratory of Mathematics,
Cryptography and Mechanics, Fstm
University Hassan II of Casablanca,
Morocco
email: khadir@hotmail.com

Abstract. In this paper, we consider the RSA variant based on the key equation $ed \equiv 1 \pmod{\phi(N)}$ where $N = p^r q$, $r \geq 2$. We show that if the secret exponent d is close to any multiple of the prime factor p or its powers, then it is possible to factor N in polynomial time in $\log N$.

1 Introduction

Factoring large integers is a well established problem in number theory and cryptography. The security of many cryptosystems such as RSA [15] is based on its presumed difficulty. Fermat method is efficient to factor a product of two numbers that are close one to another (see e.g. [4]). In 1931, the continued fraction method was invented [5]. Pollard [11] described the $p-1$ method in 1974. Some years later, he discovered the ρ algorithm [12]. The first is known to be practical when the prime factors of $p-1$ are small for some prime divisor of N. The second applies a cycle detection technique. In [7], Lenstra employed the elliptic curves properties to get prime factors of large numbers.

Computing Classification System 1998: F.2.1 E.3
Mathematics Subject Classification 2010: 11Y05 94A60
Key words and phrases: integer factorization problem, multipower RSA, public key cryptography

The Quadratic Sieve was published in 1980s by Pomerance [13]. Nowadays, the most efficient factoring algorithm is the Number Field Sieve (NFS) [6, p. 103] that was elaborated by Pollard in 1988. Since then, the NFS has been further ameliorated.

In order to speed up the decryption/encryption time, it was suggested to use RSA with moduli $N = p^r q$ (see e.g. [2]). Numerous previous papers studied the security of such protocols. Boneh, Durfee, and Howgrave-Graham [1] established that only $\frac{1}{1+r}$ fraction of the bits of p suffice to recover the entire p. May [9] generalizes many cryptanalysis methods to schemes with $N = p^r q$. Some of the work in [9] was improved in [16] for $r \leq 5$. It was proved in [8] that leaking the bits of some blocks of the prime factors of a modulus $N = p^r q$ enables its factorization under certain circumstances. All the previous researches pointed out that integers $N = p^r q$ are more vulnerable than a standard RSA modulus, in particular, when r becomes large.

In this paper, the RSA variant based on the key equation $ed \equiv 1 \pmod{\phi(N)}$ where $N = p^r q$ is considered. We show that using a secret exponent d which is close to any multiple of p or its powers can lead to the factorization of the public modulus.

The article is organized as follows. In section 2, we state our main result after recalling the Coppersmith's theorem for finding small roots of univariate modular polynomials. In section 3, we generalize the method to other RSA-type systems. Finally, we conclude in section 4.

Throughout the sequel, for integers a, b and c, we write $a \equiv b \pmod{c}$ if c divides the difference $a - b$, and $a = b \mod c$ if a is the remainder in the division of b by c. We denote by $\gcd(a, b)$ the greatest common divisor of a and b. All the logarithms should be interpreted as logarithms to the base 2.

2 Our contribution

In this section, we describe our main result. However, we start by presenting the Coppersmith [3] result for computing small roots of modular polynomials. In particular, we use the slight generalized version as was depicted by May in [9, 10]. One can find in [10] a thorough treatment about the method and its implementation. This technique will be needed in establishing our Theorem 2.

Theorem 1 ([10]) *Let N be an integer of unknown factorization, which has a divisor $b \geq N^{\beta}$, $0 < \beta \leq 1$. Furthermore, let $f(x)$ be a univariate monic polynomial of degree δ and let $c \geq 1$. Then, we can find all solutions x_0 for* *the equation:*

$$f(x_0) \equiv 0 \pmod{b} \text{ with } |x_0| \le cN^{\frac{\beta^2}{\delta}}$$

in time $T = O(c\delta^5 \log^9 N)$.

Now, we state the main contribution:

Theorem 2 *Let* $N = p^r q$ *where* $r \ge 2$ *is a given integer constant and* p, q *are primes of the same bit-size. We denote by* (e, d) *the public-key/secret-key pair satisfying* $ed \equiv 1 \pmod{\phi(N)}$. *Assume that there exist two integers* $i \ge 1$ *and* j *that verify* $|d - jp^i| \le N^{\left(\frac{\min(i,r-1)}{r+1}\right)^2}$ *with* $(d - jp^i) \not\equiv e^{-1} \pmod{q}$. *Parameters* i *and* j *are not necessary known. Then, we can factor* N *in polynomial time in* $\log N$.

Proof. The RSA equation is $ed \equiv 1 \pmod{\phi(N)}$. Hence, there exists an integer k such that $ed = 1 + kp^{r-1}(p-1)(q-1)$. Let $d = jp^i + \Delta$ where $\Delta \in \mathbb{Z}$ and put $l = \min(i, r-1)$. Working modulo p^l, it follows that $e\Delta - 1 \equiv 0 \pmod{p^l}$. Setting the polynomial $f(x) = x - (e^{-1} \bmod N)$, it is clear that Δ is a root of $f(x)$ modulo p^l. We assume that the inverse of e modulo N is well defined. Otherwise, we have already a non trivial divisor of N. When $q < p$, $N^{\frac{l}{r+1}} < p^l$. By hypothesis, $|\Delta| \le N^{\left(\frac{l}{r+1}\right)^2}$. So, we can determine Δ by using Theorem 1 with $b = p^l$, $\beta = \dfrac{l}{r+1}$, $\delta = 1$ and $c = 1$. Since $\Delta \not\equiv e^{-1} \pmod{q}$, it is readily seen that $\gcd(|e\Delta - 1|, N)$ splits N as $e\Delta - 1$ and N are both multiples of p^l. If $p < q$, then $q < 2p$ as p and q are of the same bit-size. Thus, $\dfrac{N^{\frac{l}{r+1}}}{2} < p^l$. Let $\beta = \dfrac{l}{r+1} - \dfrac{1}{\log N}$. We have $|\Delta| \le N^{\frac{\beta^2}{\delta}}$ whenever $|\Delta| \le N^{\left(\frac{l}{r+1}\right)^2}$. This comes from observing that $N^{\left(\frac{l}{r+1}\right)^2} = 4N^{\left(\frac{l}{r+1}\right)^2 - \frac{2}{\log N}} < 4N^{\left(\frac{l}{r+1}\right)^2 - \frac{2l}{(r+1)\log N} + \frac{1}{\log^2 N}} = 4N^{\frac{\beta^2}{\delta}}$. Hence, we obtain Δ by applying Theorem 1 with $b = p^l$, $\beta = \dfrac{l}{r+1} - \dfrac{1}{\log N}$, $\delta = 1$ and $c = 4$. We then get the factorization of N by computing $\gcd(|e\Delta - 1|, N)$.

The running time of our method is dominated by that of Theorem 1 which is polynomial in $\log N$. So, the result is proved.

$\square$

Theorem 2 leads to the following algorithm:

Input: A public multi power RSA key (N, e) where $N = p^r q$ for a given constant $r \ge 2$.

Output: The prime decomposition of N or "Failure".

```
1. Set the modular polynomial $f(x) = x - (e^{-1} \mod N)$.
2. Apply Coppersmith method as presented in Theorem 1 to compute all the integer roots of $f(x)$ modulo $p^l$ where $l \leq r - 1$ is a fixed integer. It is not required to know the value of $p$.
3. Denote by $x_i, i = 1, 2, \ldots, \mathtt{length}$, the solutions founded in step 2.
4. flag $\leftarrow 0$, $i \leftarrow 0$.
5. While flag $= 0$ and $i \leq \mathtt{length}$ do:
    5.1. $i \leftarrow i + 1$.
    5.2. $\Delta \leftarrow x_i$.
    5.3. $f \leftarrow \gcd(|e\Delta - 1|, N)$.
    5.4. If $1 < f < N$ then:
        5.4.1. flag $\leftarrow 1$.
        5.4.2. If $f$ is not a prime power, then $q \leftarrow f$, $p \leftarrow \left(\frac{N}{q}\right)^{\frac{1}{r}}$.
        5.4.3. Else, determine the prime $p$ that divides $f$, $q \leftarrow \frac{N}{p^r}$.
        5.4.4. Output $(p, q)$.
6. If $i > \mathtt{length}$, then output "Failure".
```

For a multi power RSA modulus $N = p^r q$, it is generally recommended to choose a small value of r. Indeed, the more r is large, the less the cryptosystem is secure, see e.g. [1, 9, 8, 16]. Setting $r = 2$, we obtain the next corollary:

Corollary 3 *Let* $N = p^2 q$ *where* p *and* q *are primes of the same bit-size. We denote by* (e, d) *the public-key/secret-key pair satisfying* $ed \equiv 1 \pmod{\phi(N)}$*. Assume that there are two integers* $i \geq 1$ *and* j *such that* $|d - jp^i| \leq N^{\frac{1}{9}}$ *with* $(d - jp^i) \not\equiv e^{-1} \pmod{q}$*. Then, we can factor* N *in polynomial time in* $\log N$*.*

The bound $N^{\left(\frac{\min(i,r-1)}{r+1}\right)^2}$ in Theorem 2 is optimal for $i \geq r - 1$. Under this situation, it is roughly equal to N when r becomes larger.

In the next section, we investigate the threat of our method to other RSA variants.

3 Extension of our result

The straightforward multi power RSA is obtained by taking $N = p^r q$ in the standard RSA key equation $ed \equiv 1 \pmod{\phi(N)}$. The Takagi crypotsystem [17] is based on $ed_p \equiv 1 \pmod{p-1}$ and $ed_q \equiv 1 \pmod{q-1}$. For this protocol, we have:

Proposition 4 *Let* $N = p^r q$ *where* $r \geq 2$ *is a given integer constant and* p, q *are primes of the same bit-size. We denote by* (e, d_p) *the public-key/secret-key pair satisfying* $ed_p = 1 + k_p(p-1)$, *i.e.* $ed_p \equiv 1 \pmod{p-1}$. *Assume that* $|d_p - p| \leq N^{\frac{1}{(r+1)^2}}$ *with* $(d_p - p) \not\equiv (1-k_p)e^{-1} \pmod{q}$. *Then, we can factor* N *in time* $eO(\log^9 N)$.

Proof. By definition, $ed_p = 1 + k_p(p-1)$ for some integer k_p. Put $d_p = p + \Delta$. Hence $e\Delta + k_p - 1 \equiv 0 \pmod{p}$. The parameter k_p lands in the set $\{1, 2, \ldots, e-1\}$. Indeed, $k_p = \dfrac{ed_p - 1}{p-1} < e$. Put $f(x) = x + (k_p - 1)(e^{-1} \mod N)$. For the true guess for k_p, Δ is a root of the polynomial $f(x)$ modulo p.

If $q < p$, then $N^{\frac{1}{1+r}} < p$. By hypothesis, $|\Delta| \leq N^{\frac{1}{(1+r)^2}}$. So, it is possible to find Δ by applying Theorem 1 to $f(x)$ with $\beta = \dfrac{1}{1+r}$, $\delta = 1$ and $c = 1$. Moreover, $\Delta \not\equiv (1-k_p)e^{-1} \pmod{q}$. Thus, $\gcd(|e\Delta + k_p - 1|, N)$ is a prime power that divides N, since both $e\Delta + k_p - 1$ and N are multiples of p. The most time consuming part is Coppersmith method which has a running time $O(\log^9 N)$. All the steps must be repeated for each trial k_p. Therefore, the whole complexity is $eO(\log^9 N)$.

Now, suppose that $p < q$. The primes p and q are of the same bit-size, so $q < 2p$. It follows that $\dfrac{N^{\frac{1}{1+r}}}{2} < p$. We have $N^{\frac{1}{(r+1)^2}} = 4N^{\left(\frac{1}{r+1}\right)^2 - \frac{2}{\log N}} < 4N^{\left(\frac{1}{r+1}\right)^2 - \frac{2}{(r+1)\log N} + \frac{1}{\log^2 N}} = 4N^{\frac{\beta^2}{\delta}}$. Using Theorem 1 with $f(x) = x + (k_p - 1)(e^{-1} \mod N)$, $\beta = \dfrac{1}{1+r} - \dfrac{1}{\log N}$, $\delta = 1$ and $c = 4$ leads to recovering Δ. Like in the previous case, $\gcd(|e\Delta + k_p - 1|, N)$ is a prime power divisor of N which achieves the proof.

□

For an RSA-type modulus $N = p^r q$, the primes p and q are not symmetric. We can establish:

Proposition 5 *Let* $N = p^r q$ *where* $r \geq 2$ *is a given integer constant and* p, q *are primes of the same bit-size. We denote by* (e, d_q) *the public-key/secret-key* *pair satisfying* $ed_q = 1 + k_q(q-1)$, *i.e.* $ed_q \equiv 1 \pmod{q-1}$. *Assume that* $|d_q - q| \leq N^{\frac{1}{(r+1)^2}}$ *with* $(d_q - q) \not\equiv (1-k_q)e^{-1} \pmod{p^u}$ *for all* $u \leq r$. *Then, we can factor* N *in time* $eO(\log^9 N)$.

Proof. The RSA key equation satisfies $ed_q = 1 + k_q(q-1)$ for some integer k_q. The following process will be repeated for each candidate for k_q. If $d_q = q + \Delta$ where $\Delta \in \mathbb{Z}$, then $e\Delta + k_q - 1 \equiv 0 \pmod q$. We define the function $f(x) = x + (k_q - 1)(e^{-1} \mod N)$. Let $p < q$. Thus, $N^{\frac{1}{1+r}} < q$. By hypothesis, $|\Delta| \leq N^{\frac{1}{(r+1)^2}}$. Setting $\beta = \dfrac{1}{1+r}$, $\delta = 1$ and $c = 1$, we efficiently determine Δ by Theorem 1. If $q < p$, $\dfrac{N^{\frac{1}{1+r}}}{2} < q$ since p and q are of the same bit-size. One shows that it suffices that $|\Delta| \leq N^{\frac{1}{(r+1)^2}}$ in order to use Coppermsith's result with $\beta = \dfrac{1}{1+r} - \dfrac{1}{\log N}$, $\delta = 1$ and $c = 4$.

As both $e\Delta + k_q - 1$ and N are divided by q, the condition $(d_q - q) \not\equiv (1-k_q)e^{-1} \pmod{p^u}$ for all $u \leq r$ guarantees that $\gcd(|e\Delta + k_q - 1|, N) = q$. All the steps are executed at most e times given that $k_q < e$. Hence, the running time of the method is $eO(\log^9 N)$ which demonstrates the result.

□

Suppose that a private exponent d_p or d_q satisfies the hypothesis of Proposition 4 or 5 respectively. It is clear that if there exists an oracle that outputs the values of k_p or k_q such that $ed_p = 1 + k_p(p-1)$ or $ed_q = 1 + k_q(q-1)$, then N can be factored in polynomial time in $\log N$.

In the following proposition, we apply the technique for RSA systems that use Chinese remainder theorem in decrypting, CRT-RSA (see e.g. [14] for an explicit description). We obtain:

Proposition 6 *Let* $N = pq$ *an RSA modulus where* p *and* q *are primes of the same bit-size. We denote by* (e, d_p) *the public-key/secret-key pair satisfying* $ed_p = 1 + k_p(p-1)$, *i.e.* $ed_p \equiv 1 \pmod{p-1}$. *Assume that* $|d_p - p| \leq N^{\frac{1}{4}}$ *with* $(d_p - p) \not\equiv (1 - k_p)e^{-1} \pmod q$. *Then, we can factor* N *in time* $eO(\log^9 N)$.

Proof. By the RSA key equation, $ed_p = 1 + k_p(p-1)$ where $k_p \in \mathbb{N}$. It follows that $ed_p + k_p - 1 \equiv 0 \pmod p$. We know that $d_p = p + \Delta$, so $e\Delta + k_p - 1 \equiv 0 \pmod p$. The modulus $N = pq$ is balanced. Consider the polynomial $f(x) = x + (k_p - 1)(e^{-1} \mod N)$ whose degree is $\delta = 1$. The value of $|\Delta|$ is upper bounded by $N^{\frac{1}{4}}$. It is possible to compute efficiently Δ by Theorem 1 with

$\beta = \frac{1}{2}$, $c = 1$ if $q < p$, and $\beta = \frac{1}{2} - \frac{1}{\log N}$, $c = 2$ if not. By hypothesis, $\Delta \not\equiv (1 - k_p)e^{-1} \pmod{q}$, so $\gcd(|e\Delta + k_p - 1|, N) = p$. We must execute the method for each candidate for k_p. As $k_p = \frac{ed_p - 1}{p - 1}$ and $d_p < p - 1$, $k_p < e$. So, the running time is $eO(c \log^9 N)$ where $c = 1$ if $q < p$ and $c = 2$ otherwise. □

Let d_p a private exponent that fulfil the hypothesis of Proposition 6. If the value of k_p such that $ed_p = 1 + k_p(p - 1)$ is leaked, then we can efficiently compute the prime decomposition of N.

4 Conclusion

In this paper, we proposed an attack against the RSA variant based on the key equation $ed \equiv 1 \pmod{\phi(N)}$ where $N = p^r q$, $r \geq 2$. We showed that if d is close to any multiple of the prime factor p or its powers, then N can be factored in polynomial time in $\log N$, and thus the cryptosystem is completely broken.

Acknowledgements

This work is supported by the MMSyOrientation project.

References

[1] D. Boneh, G. Durfee, and N. Howgrave-Graham, Factoring $N = p^r q$ for large r, *Advances in Cryptology (CRYPTO'99), Lecture Notes in Computer Science* **1666** (1999) 326–337. ⇒144, 146

[2] D. Boneh and H. Shacham, Fast variants of RSA, *CryptoBytes* **5,** 1 (2002) 1–9. ⇒144

[3] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerabilities, *J. Cryptology* **10,** 4 (1997) 233–260. ⇒144

[4] B. De Weger, Cryptanalysis of RSA with small prime difference, *Appl. Algebra Engrg. Comm. Comput.* **13,** 1 (2002) 17–28. ⇒143

[5] Derrick H. Lehmer and Richard E. Powers, On factoring large numbers, *Bull. Amer. Math. Soc.* **37,** 10 (1931) 770–776. ⇒143

[6] Arjen K. Lenstra, Hendrik W. Lenstra Jr., *The development of the number field sieve*, Lecture Notes in Mathematics **1554,** Springer-Verlag, 1993. ⇒144

[7] Hendrik W. Lenstra Jr., Factoring integers with elliptic curves, *Ann. of Math.* **126,** 3 (1987) 649–673. ⇒143

[8] Y. Lu, R. Zhang, and D. Lin, Factoring multi-power RSA modulus $N = p^r q$ with partial known bits, *Information Security and Privacy (ACISP 2013)*, *Lecture Notes in Computer Science* **7959** (2013) pp. 57–71. ⇒144, 146

[9] A. May, Secret exponent attacks on RSA-type schemes with moduli $N = p^r q$, *Public Key Cryptography (PKC 2004)*, *Lecture Notes in Computer Science* **2947** (2004) 218–230. ⇒144, 146

[10] A. May, Using LLL-reduction for solving RSA and factorization problems, *The LLL Algorithm, Phong Q. Nguyen and Brigitte Vallée, eds.*, *Information Security and Cryptography*, Springer Berlin Heidelberg, (2010) 315–348. ⇒144

[11] John M. Pollard, Theorems on factorization and primality testing, *Math. Proc. Cambridge Philos. Soc.* **76,** 3 (1974) 521–528. ⇒143

[12] John M. Pollard, A monte carlo method for factorization, *BIT* **15,** 3 (1975) 331–334. ⇒143

[13] C. Pomerance, The quadratic sieve factoring algorithm, *Advances in Cryptology (EUROCRYPT'84)*, *Lecture Notes in Computer Science* **209** (1985) 169–182. ⇒144

[14] J.J. Quisquater and C. Couvreur, Fast decipherment algorithm for RSA public-key cryptosystem, *Electronics letters* **18,** 21 (1982) 905–907. ⇒148

[15] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21,** 2 (1978) 120–126. ⇒143

[16] S. Sarkar, Small secret exponent attack on RSA variant with modulus $N = p^r q$, *Des. Codes Cryptogr.* **73**, 2 (2014) 383–392. ⇒144, 146

[17] T. Takagi, Fast RSA-type cryptosystem modulo $p^k q$, *Advances in Cryptology (CRYPTO'98)*, *Lecture Notes in Computer Science* **1462** (1998) 318–326. ⇒ 147

Received: August 8, 2015 • Revised: October 26, 2015

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 151–185

DOI: 10.1515/ausi-2015-0017

Simple scalable nucleotic FPGA based short read aligner for exhaustive search of substitution errors

Péter FEHÉR
Eötvös Loránd University
email: thepetest@gmail.com

Ágnes FÜLÖP
Eötvös Loránd University
email: fulop@caesar.elte.hu

Gergely DEBRECZENI
Wigner Institute
email: gergely.debreczeni@cern.ch

Máté NAGY-EGRI
Wigner Institute
email: nagy-egri.mate@wigner.mta.hu

György VESZTERGOMBI
Wigner Institute and Eötvös Loránd University
email: veszter@rmki.kfki.hu

Abstract. With the advent of the new and continuously improving technologies, in a couple of years DNA sequencing can be as commonplace as a simple blood test. The growth of sequencing efficiency has a larger exponent than the Moore's law of standard processors, hence alignment and further processing of sequenced data is the bottleneck. The usage of FPGA (Field Programmable Gate Arrays) technology may provide an efficient alternative. We propose a simple algorithm for DNA sequence alignment, which can be realized efficiently by nucleotic principal agents of Non.Neumann nature. The prototype FPGA implementation runs on a small Terasic DE1-SoC demo board with a Cyclone V chip. We present test results and furthermore analyse the theoretical scalability of this

Computing Classification System 1998: I.2.1, D.1.3
Mathematics Subject Classification 2010: 92D20
Key words and phrases: FPGA, parallel computing, DNA sequent

system, showing that the execution time is independent of the length of reference genome sequences. A special advantage of this parallel algorithm is that it performs exhaustive search producing all match variants up to a predetermined number of point (mutation) errors.

1 Introduction

Revolution in microbiology and genetics are producing incredible amount of data which calls for the application of the most modern tools of informatics [1]. This may include analysis of sequences from related organisms, or from apparently unrelated species. In order for a geneticist to perform analyses on genomic data it has to be obtained through a process called genome sequencing. This task can be broken down into two main sub-processes, the first of which involves extracting raw data from a sample using various instruments and the second is short read alignment, which is a purely computational problem. Recently, several short read alignment applications have been developed. The state of the art short read aligners (e.g. Bowtie[7], nvBowtie[17]) use the the Burrows-Wheeler transform[4].

The problem is the following: For every short read find the position where it best matches the reference, i.e. can be aligned to the reference sequence with the lowest number of differences. Differences can be insertions, deletions or substitution errors (also called point mutation errors). Insertions and deletions are commonly referred to as indels.

In practice the percentage of substitution errors is much higher than that of indels, therefore in this article we shall concentrate on the algorithms dealing with only substitution errors. In special cases it will be specified if indels are also taken into account.

Reference sequences are publicly available for a wide variety of species including the human genome. Sometimes shorter fragments are used instead of whole genomes in order to narrow down the search space and speed up the process. Short read data is generally produced using special sequencing instruments such as the Illumina HiSeq X Ten but it is also possible to find existing data from previous experiments using public databases.

In practice there can be more than one reference sequence, which could for example belong to different chromosomes of an organism. However, this doesn't alter the theoretical nature of the problem. So in our demonstration we decided to use a single reference sequence as our input.

In the Section (2.1), (2.2) we reviewed the numeric algorithms of DNA sequencing as Smith Waterman scosing system, Burrows-Wheeler transform.

The basic idea of our model is introduced in the Section (2.3). The Local Boolean alignment algorithms contains the Sliding windows sequential algorithm in the Subsection (3.1), Coarse grain K parallelism in the Subsection (3.2), Fine grain N-parallelism in the Subsection (3.3), these are close to hardware application. In the Section (4.) we introduced the Nucleotic algorithms, which is treating big-data strongly parallel on scalable way to realise by FPGA. This is an effective method for DNA sequence alignment. In the Section (5.) the FPGA implementation is shown on DE1-SoC Board to compare with GPU. In the Section (5.5) we presented our method in the case of Lambda virus. We compared these results with Bowtie method using a random and real sample string of Lambda phage. The correlation coefficients show significant difference between the faul and exacting matching result.

2 Numeric transformation algorithms

Traditionally the computers with the standard CPUs are ideal to perform formula calculations therefore the alignment algorithms usually applied some numerical or analytic transformation to the data which provided some computational procedure to reach the desired result. The sequence alignment in these algorithms can be applied locally or globally. The Smith-Waterman algorithm, the Burrows-Wheeler Transformation and circular convolution algorithms all perform local alignments. The global approach shown by Figure 1 is used for comparing sequences of similar length and is not discussed in this article.

```
Global: CGCGGAATGTACGATA
        CGC--AAT-TAC-ATA

Local:  CGCGGAATGTACGATA
        ---GGAAT-TACGA--
```

Figure 1: Global and local alignment

Another important concept must be introduced before moving on to the discussion of various algorithms. The DNA molecule consists of two strands: 5'→3'(forward) and 3'→5'(reverse). On the reverse strand every letter is determined by the forward strand letter in the corresponding position and vice versa (A is opposite of T and C is opposite of G). It is possible to ensure that the sequencing instruments always reads in the 5'→3' direction but it can happen on both the forward strand and the reverse strand. The direction of the reference sequence is regarded as the forward strand direction. Since a short read could have originated from the reverse strand a special transformation must be performed in order to create a version that is suitable for alignment on the forward strand. This transformed version is essentially the reverse of the original sequence after replacing each letter with its opposite. Thus for every short read we must run our alignment algorithm for the original version plus this new version called the reverse complement. This is illustrated in Figure 2.

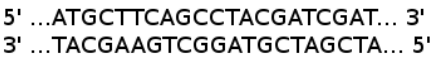

5' ...ATGCTTCAGCCTACGATCGAT... 3'
3' ...TACGAAGTCGGATGCTAGCTA... 5'

Figure 2: Forward and reverse strands of DNA

2.1 Smith-Waterman scoring system

The **Smith-Waterman algorithm**[10] performs local sequence alignment; that is, for determining similar regions between two strings or nucleotide or protein sequences. Instead of looking at the total sequence, the Smith-Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure using the H scoring matrix. Backtracking starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment.

The basic element of this universal method which is able to deal simultaneously with substitution, point errors and indels is the calculation of the $H(i,j)$ scoring matrix:

$H(i,0)=0,\ 1\le i\le m,\ H(0,j)=0,\ 1\le j\le n$

$$H(i,j)=max\left(\begin{array}{rl} 0 & \\ H(i-1,j-1)+s(a_i,b_j) & \text{Match/Mismatch} \\ max_{k\ge 1}H(i-k,j)+W_k & \text{Deletion} \\ max_{l\ge 1}H(i,j-l)+W_l & \text{Insertion} \end{array}\right)$$

where $1\le i\le m,\ 1\le j\le n$

We use the next notation:

a, b = Strings over the alphabet , $m = length(a)$, $n = length(b)$,

$s(a,b)$ is a similarity function on the alphabet W_i is the gap-scoring scheme.

We show one example:

Sequence 1 = ACACACTA Sequence 2 = AGCACACA

$$s(a,b) = \begin{cases} +2, & \text{if } a = b \text{ (match)} \\ -1, & \text{if } a \neq b \text{ (mismatch)} \end{cases}$$

and $W_i = -i$.

The H matrix is the following

$$H = \left(\begin{array}{cccccccccc} & - & A & C & A & C & A & C & T & A \\ - & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \mathbf{2} & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & \mathbf{3} & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & \mathbf{5} & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & \mathbf{7} & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & \mathbf{9} & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & \mathbf{11} & \mathbf{10} & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & \mathbf{12} \end{array} \right)$$

The alignment is reconstructed as follows: one is starting with the highest value stepping toward the next highest. A diagonal jump implies there is an alignment (either a match or a mismatch=point error). A top-down jump implies there is a deletion. A left-right jump implies there is an insertion.

For the example, the results are:

Sequence 1 = A-CACACTA

Sequence 2 = AGCACAC-A

The motivation for local alignment is the difficulty of obtaining correct alignments in regions of low similarity between distantly related biological sequences, because mutations have added too much 'noise' over evolutionary time to allow for a meaningful comparison of those regions. Local alignment avoids such regions altogether and focuses on those with a positive score, i.e. those with an evolutionary conserved signal of similarity.

The Smith-Waterman algorithm is fairly demanding of time: To align two sequences of lengths m and n, $O(mn)$ time is required. Smith-Waterman local similarity scores can be calculated in $O(m)$ (linear) space if only the optimal alignment needs to be found, but naive algorithms to produce the alignment require $O(mn)$ space.

2.2 Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT)[4] is applied on blocks of input data (symbols). It is usually the case that larger blocks result in greater compressibility of the transformed data at the expense of time and system resources.

One of the effects of BWT is to produce blocks of data with more and longer 'runs' (= strings of identical symbols) than those found in the original data. The increasing the number of these 'runs' and their lengths tends to improve the compressibility of data.

The first step of BWT is to read the T string in a block of N symbols.

The second step is adding a $ character as ending symbol assigning the lowest character value to it in the alphabetic order.

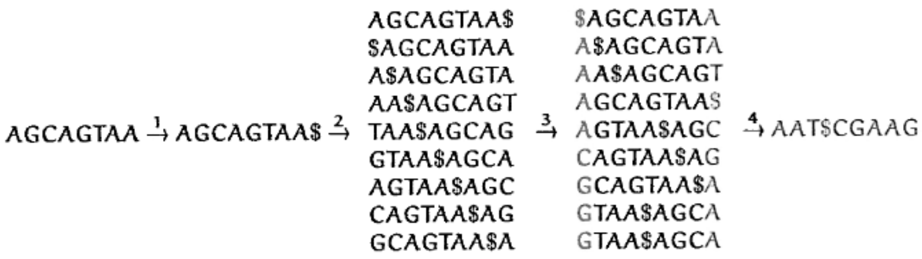


Figure 3: Burrows-Wheeler transformation steps, where red letterr are noted by F, and green characters are correspond to green L

The next step is to think of the block as a cyclic buffer: N strings (rotations). The rotation matrix may be constructed in such a way, containing the shifted blocks line by line.

The fourth step of BWT is to lexicographically sort the matrix lines (Figure 3). The first column of the matrix is denoted by F, the last column L is defined to be the Burrows-Wheeler transform of T:

$$\mathsf{L} = \mathsf{BWT}(\mathsf{T}).$$

In short:

$$\mathsf{T} = \mathsf{AGCAGTAA} \rightarrow \mathsf{AGCAGTAA\$} \rightarrow \mathsf{L} = \mathsf{AAT\$CGAAG} \rightarrow \mathsf{F} = \mathsf{\$AAAACGGT}$$

It is a very remarkable mathematical fact that knowing only L one can restore uniquely the original T string.

The first step of the reversing process is that one creates F from L by lexicographical ordering.

The basic trick of the reverse transform is the Last-to-First-Mapping property of the L and F strings.

$$
\begin{array}{c}
\mathsf{L} \rightarrow \mathsf{F} \\
\\
\mathsf{A} \leftarrow \$ \\
\mathsf{A} \leftarrow \mathsf{A} \\
\mathsf{T} \leftarrow \mathsf{A} \\
\$ \leftarrow \mathsf{A} \\
\mathsf{C} \leftarrow \mathsf{A} \\
\mathsf{G} \leftarrow \mathsf{C} \\
\mathsf{A} \leftarrow \mathsf{G} \\
\mathsf{A} \leftarrow \mathsf{G} \\
\mathsf{G} \leftarrow \mathsf{T}
\end{array}
$$

Each element of F is pointing to the symbol of L which is preceding it in T, i.e. one has from the beginning **the pair wise reconstruction of T** in the $\mathsf{L(i)F(i)}$ combinations. Thus one needs only to connect them in right order.

The symbols of the T string are produced in reverse order which means that one should start from the ending character $.

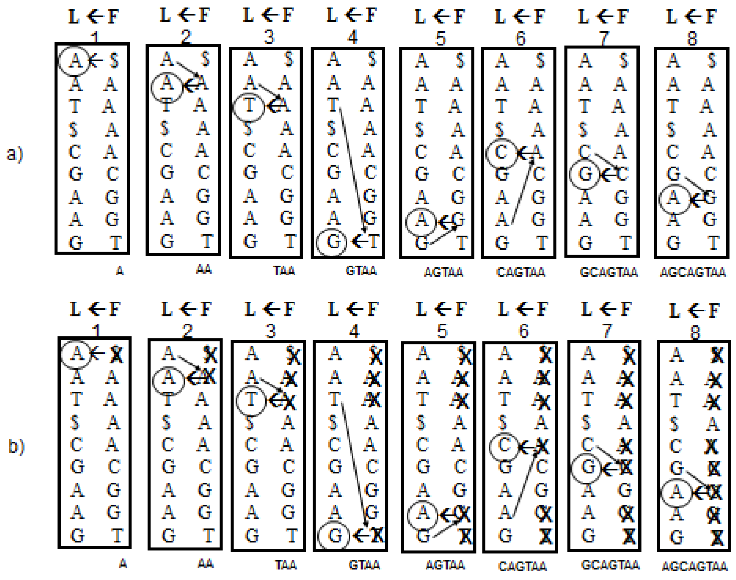


Figure 4: BWT reverse, part b) with X in F for used characters

It is worth to mark the already selected pairs in F with an 'X' as-shown in Figure 4 (b).

The horizontal arrow in the first line from F to L provides the N-th, i.e. the last symbol of T which is A.

In the next step one is jumping from the last L position (A in line #1) to the nearest F position containing the same symbol, which ensures the piecewise continuity. Thus in the above example one is ending in the second line of F.

The next horizontal arrow from F to L gives symbol A, which is really identical with the $(\mathsf{N}-1)$-th character of T.

The procedure is repeated from the second line of L, selecting the nearest A in F which is not in the same line. Thus we reach the A character in line #3 of F.

And so on one can repeat the horizontal $\mathsf{F} \to \mathsf{L}$ and inclined $\mathsf{L} \to \mathsf{F}$ steps until one gets the final AGCAGTAA (Figure 4).

Of course, the procedure can be formulated in a more exact way too, it is based on two tables. The first is giving

Number of Preceding Symbols Matching Symbol in Current Position in L;

the second one is derived from F:

Number of Symbols Lexicographically Less Than Current Symbol

which are described in detail in ref [Burrows-Wheeler Transform Discussion and Implementation, talk by Michael Dipperstein [13]]

How can one use BWT for alignment of short reads? One can prepare the BWT of the known reference sequence containing of N characters. If the short read with m characters is identical with some part of the reference sequence, then one can assume that using the last character of the short read as starting character in F one can execute a reverse transformation from this point. As a simple test we can check in the above T as reference whether it contains the CAG combination.

In general there is not a single solution. E.g. one finds 2 combinations for CAG (Figure 5).

Try to find ACAG short read or TCAG. No way, because from the last position where the C was found one cannot go further. Let us assume that due to a point error the short read was recorded by a point error as ATCAGTAA, which has no exact matching with the reference sequence. In this case one can use some kind of heuristic method, the so called backtracking. In case of unsuccessful search the program executes some backward steps and the recorded character is changed to a new one. The selection of the position and value of the new character is depends on the measured quality of the recorded characters which is monitored during the measuring process. It is important to remark, that if one can execute exhaustive research for point error cases, then one doesnt need to apply such heuristic algorithms, which can be demonstrated

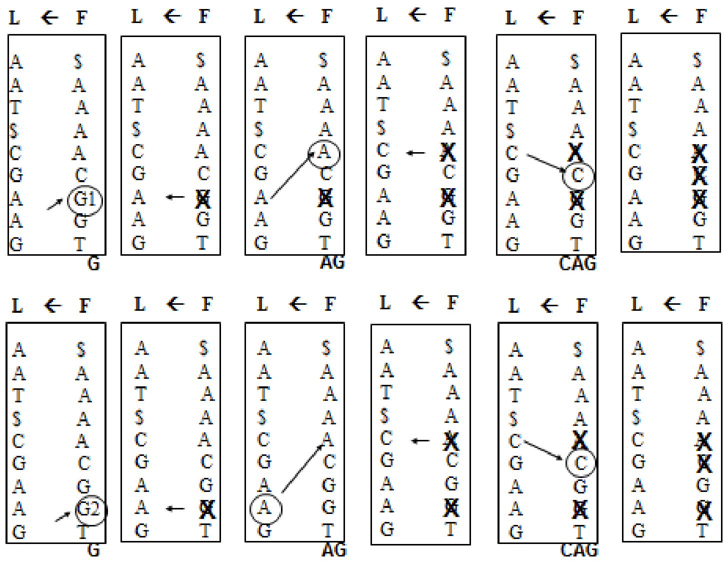


Figure 5: BWT search for partial string 'CAG', two possible solution

by the following algorithm.

2.3 Pseudo-binary circular convolution

The circular convolution is a frequently used reduced version of the general convolution formula. One can define it in the following way:

$$y(n) = h(n)@u(n) = \sum_{i=0}^{N-1} h(i) \cdot (u(n-i))_N,$$

or:

$$y(n) = h(n)@u(n) = \sum_{i=0}^{N-1} h(i) \cdot (u(n+i))_N,$$

where: $(u(n))_N$,N-point periodic extension of u(n). 'Cyclic'='circular'.
Order: 'N-point' or 'order N', $y(n)$; $h(n)$; $u(n)$ all have length N.

Here we want to specialize it further to accommodate the DNA alignment case. It will be assumed that the $u(n)$ function will correspond to the reference genome sequence of length N, whereas the short reads will be represented by $h(i)$ having non-zero values only for $0 \leq i \leq m-1$, where $m < N$, $h(i)$ is equal to zero above this value till $i = N$. It is assumed that the characters of u and h are written in binary form of 1s and 0s, thus the total length will be increased from N to $N_b = n_{bit} \cdot N$, where n_{bit} is the number of bits required to identify the character symbols.

This binary circular convolution can have a special physical meaning if one applies and additional trick by converting the zero values to -1 in u and h functions.

Example:

$N = 5$, $m = 4$ and $n_{bit} = 2$

Symbols: a, b, c, d binary representation:	00,	01,	10,	11
pseudo binary:	-1-1,	-1 1,	1 -1,	11

Reference string: u bacad → binary:	0	1	0	0	1	0	0	1	1	1
bacad → pseudo-binary:	-1	1	-1	-1	1	-1	-1	1	1	1
Short read string: h acad → binary:	0	0	1	0	0	1	1	1	0	0
acad → pseudo-binary:	-1	-1	1	-1	-1	1	1	1	0	0

padding zeros at the end.

Binary convolution:

$$\begin{aligned}
y(0) &= h(0)\cdot u(0) + h(1)\cdot u(1) + h(2)\cdot u(2) + \cdots + h(9)\cdot u(9)\\
&= 0\cdot 0 + 0\cdot 1 + 1\cdot 0 + 0\cdot 0 + 0\cdot 1 + 1\cdot 0 + 1\cdot 0 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 1\\
y(1) &= h(0)\cdot u(1) + \ldots\\
y(2) &= h(0)\cdot u(2) + h(1)\cdot u(3) + h(2)\cdot u(4) + \cdots + h(9)\cdot u(1)\\
&= 0\cdot 0 + 0\cdot 0 + 1\cdot 1 + 0\cdot 0 + 0\cdot 0 + 0\cdot 0 + 1\cdot 1 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 3\\
y(3) &= h(0)\cdot u(3) + \ldots\\
&\vdots\\
y(9) &= h(0)\cdot u(9) + \ldots
\end{aligned}$$

Pseudo-binary convolution:

$$\begin{aligned}
y(0) &= h(0)\cdot u(0) + h(1)\cdot u(1) + h(2)\cdot u(2) + \cdots + h(9)\cdot u(9)\\
&= (-1)\cdot(-1) + (-1)\cdot 1 + 1\cdot(-1) + (-1)\cdot(-1) + (-1)\cdot 1 + 1\cdot(-1) +\\
&\quad 1\cdot(-1) + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = -2\\
y(1) &= h(0)\cdot u(1) + \ldots\\
y(2) &= h(0)\cdot u(2) + h(1)\cdot u(3) + h(2)\cdot u(4) + \cdots + h(9)\cdot u(1)\\
&= (-1)\cdot(-1) + (-1)\cdot(-1) + 1\cdot 1 + (-1)\cdot(-1) + (-1)\cdot(-1) +\\
&\quad (-1)\cdot(-1) + 1\cdot 1 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 8\\
y(3) &= h(0)\cdot u(3) + \ldots\\
&\vdots\\
y(9) &= h(0)\cdot u(9) + \ldots
\end{aligned}$$

From this example it is obvious that the pseudo binary circular convolution gives the exact number of bit matches between the reference sequence and the short read and the y index provides the position for that number of matching. Exact matching gives the value $y(n) = N_b$. It provides exhaustive search, because if there are more than one exact matching position then for all the n_i values one gets N_b.

In general, an error decreases the sum by 2, thus the number of matches is equal

$$M = (y(n) + m)/2.$$

The second remarkable feature of this formula is, that it works exactly in the similar exhaustive way, if we allow a given number of mismatching bits.

The third interesting fact is that one can speed up the calculations, which requires $N \cdot N$ steps, by using Fast Fourier Transform of h and u. The calculation time of the convolution will be reduced to $N \cdot \log(N)$ steps. In some architecture this can be the optimal solution, but in the next we propose even faster practical solutions.

3 Local Boolean alignment algorithms

From the definition it is obvious that the solution of the alignment problem does not require intense numerical calculations, therefore in the next we concentrate on the bit-level or string character manipulating algorithms which can be optimally executed in FPGA and ASIC systems.

3.1 Sliding window sequential algorithm

Let us assume that the reference genome has N base pair. One is looking for the alignment of short reads with the length of m base pairs. For simplicity, we assume that $N = K \cdot m$

The characters in reference genome and short read are compared individually within a sliding window (Figure 6). The number of sequential sliding steps is equals to $N - m + 1$.

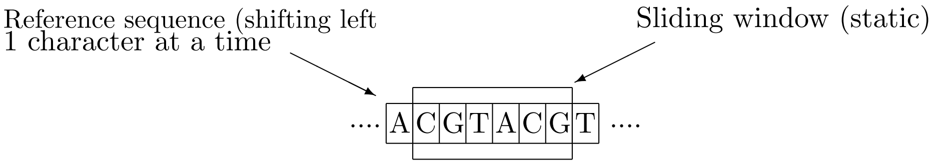


Figure 6: Sliding window principle

If one takes into account that the sequencing instruments do not give an exact copy of the measured specimen, in general there is no existing unique solution for the alignment problem. Therefore one applies statistical multiple measurement in the analysis as it is illustrated in Figure 7.

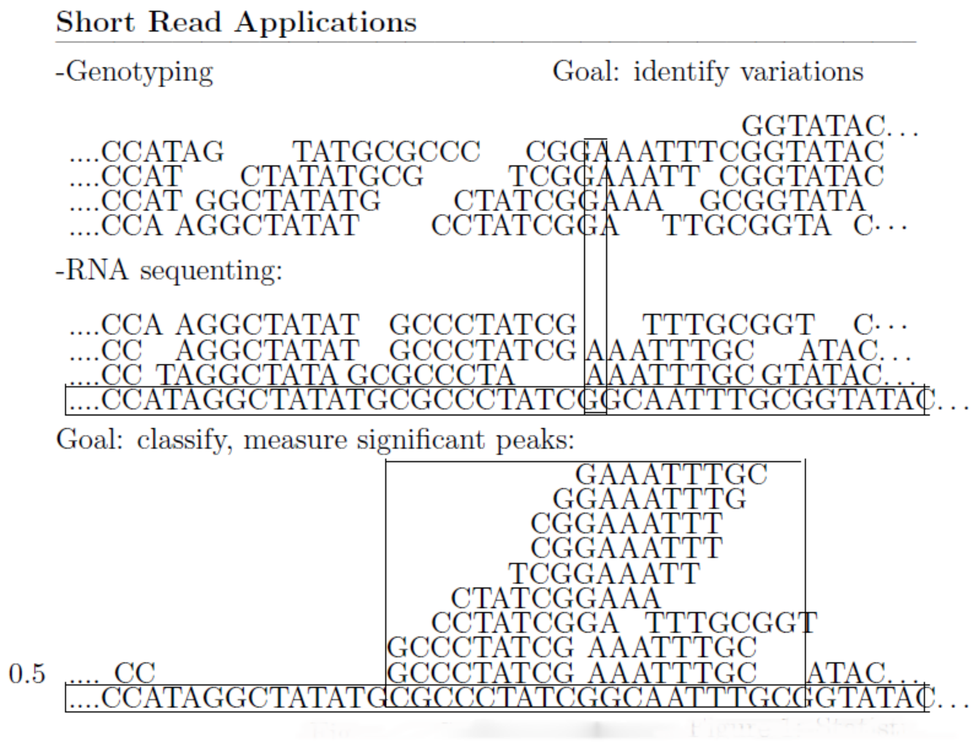

Short Read Applications
-Genotyping
Goal: identify variations
GGTATAC...
....CCATAG TATGCGCCC CGGAAATTTCGGTATAC
....CCAT CTATATGCG TCGGAAATT CGGTATAC
....CCAT GGCTATATG CTATCGGAAA GCGGTATA
....CCA AGGCTATAT CCTATCGGA TTGCGGTA C...
-RNA sequenting:
....CCA AGGCTATAT GCCCTATCG TTTGCGGT C...
....CC AGGCTATAT GCCCTATCG AAATTTGC ATAC...
....CC TAGGCTATA GCGCCCTA AAATTTGC GTATAC...
....CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...
Goal: classify, measure significant peaks:
GAAATTTGC
GGAAATTTG
CGGAAATTT
CGGAAATTT
TCGGAAATT
CTATCGGAAA
CCTATCGGA TTTGCGGT
GCCCTATCG AAATTTGC
0.5 .... CC GCCCTATCG AAATTTGC ATAC...
....CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

Figure 7: Statistical analysis

The base pair symbols can be converted to binary representation as it is shown in some simple examples (Figure 8).

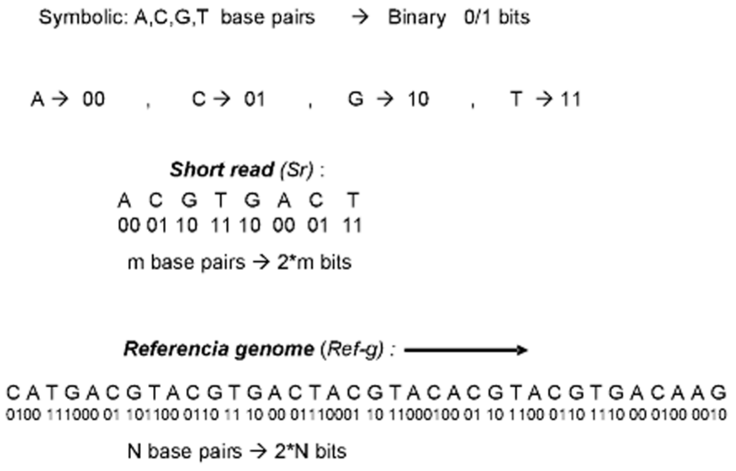


Figure 8: Illustration for symbolic binary transition

Using a single processor it takes $(N - m + 1) \cdot m$ steps to check all the combinations, which can be a very long time if N is large (Figure 9). It is not worth to compare the last $m - 1$ positions because it is not possible to have exact matching with the m-long short read.

3.2 Coarse grain K parallelism

If one applies $K = N/m$ processors then one needs only m sliding steps which reduces the execution time to $m \cdot m$ steps (Figure 10) which can be a very considerable speed up, because in general $m << N$.

The last processor will work only for the $m = 0$ case because the reference genome runs out.

3.3 Fine grain N-parallelism

If one has enough money to buy $N - m + 1$ processors then the execution will require only m steps which is very small relative to the sequential single process $(N - m + 1) \cdot m$ case (Figure 11).

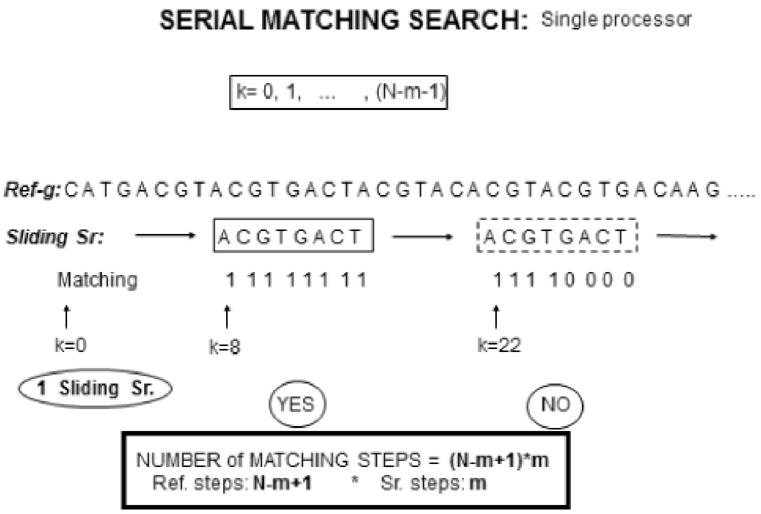

SERIAL MATCHING SEARCH: Single processor
k= 0, 1, ... , (N-m-1)
Ref-g: CATGACGTACGTGACTACGTACACGTACGTGACAAG .....
Sliding Sr:
ACGTGACT
ACGTGACT
Matching
1 11 111 11
111 10 000 0
k=0
k=8
k=22
1 Sliding Sr.
YES
NO
NUMBER of MATCHING STEPS = (N-m+1)*m
Ref. steps: N-m+1 * Sr. steps: m

Figure 9: Single principle agent realization for serial matching search

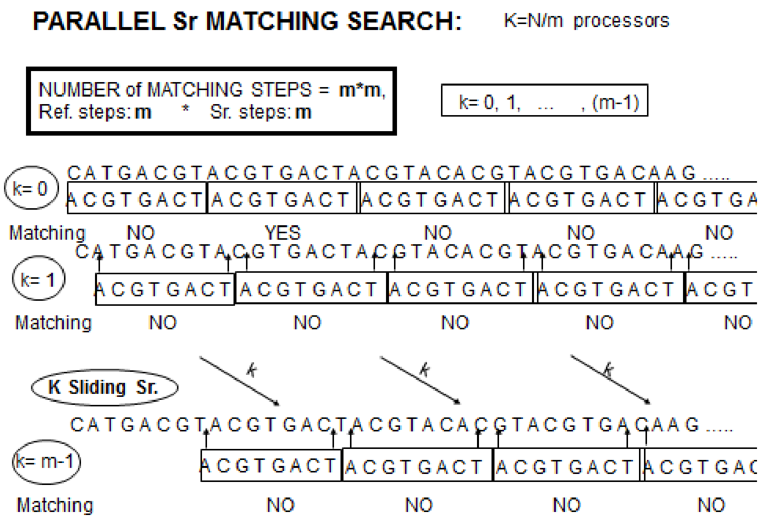

PARALLEL Sr MATCHING SEARCH: K=N/m processors
NUMBER of MATCHING STEPS = m*m,
Ref. steps: m * Sr. steps: m
k= 0, 1, ... , (m-1)
k= 0
k= 1
k= m-1
CATGACGTACGTGACTACGTACACGTACGTGACAAG .....
ACGTGACT
Matching
NO
YES
K Sliding Sr.
k

Figure 10: Moderate number of principle agents realization for coarse grain matching search

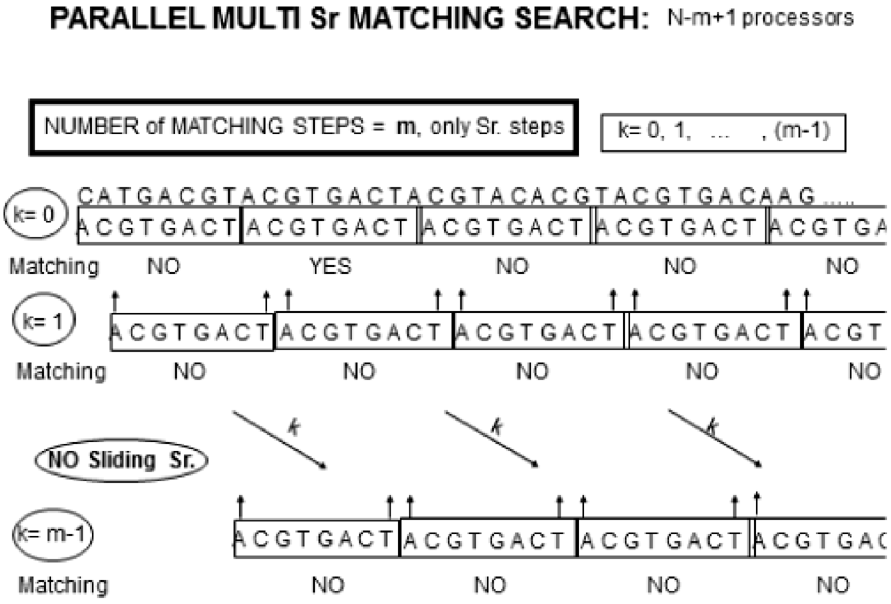

PARALLEL MULTI Sr MATCHING SEARCH: N-m+1 processors
NUMBER of MATCHING STEPS = m, only Sr. steps
k= 0, 1, ... , (m-1)
k= 0
CATGACGTACGTGACTACGTACACGTACGTGACAAG .....
ACGTGACT ACGTGACT ACGTGACT ACGTGACT ACGTGA
Matching NO YES NO NO NO
k= 1
ACGTGACT ACGTGACT ACGTGACT ACGTGACT ACGT
Matching NO NO NO NO NO
NO Sliding Sr.
k= m-1
ACGTGACT ACGTGACT ACGTGACT ACGTGAC
Matching NO NO NO NO

Figure 11: Fine grain matching search

4 Nucleotic algorithms

The Von Neumann architecture, also known as the Von Neumann model and Princeton architecture, is a computer architecture based on that described in 1945 a mathematician an physicist John Von Neumann and others in the First Draft of a Report on the EDVAC [9]. This describes a design architecture for an electronic digital computer with parts consisting of a processing unit containing an arithmetic logic unit and processor registers, a control unit containing an instruction register and program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms. The meaning has evolved to be any stored-program computer in which an instruction fetch and data operation cannot occur at the same time because they share a common bus. This is referred to as the Von Neumann bottleneck and often limits the performance of the system.

The proposed NON-Neumann architecture (NONN) is applying FPGA reconfigurable hardware realizing computation directly in the memory cells avoiding the CPU-memory bottle-neck. This NONN approach can be applied only for specific problems which are treating big-data massively parallel on SCAL-

ABLE way. The idea of massively parallel 1-bit CPU system is not new, e.g. a special ASIC design existed already 25 years ago [11]. The interesting fact is that large class of the presently unsolvable problems are falling in this category in physics, chemistry, biology, life sciences, materials, climate, geosciences, etc.

Exascale computing in general is a very nice idea, but in practice it seems to be a non realistic aim. Here we should like to reach this aim only in case of a limited set of problems which are important enough to be worth to invest into them. In the real physical world the matter is consisting from atoms, but the dominant element is the atomic nucleus containing 99.95 % of the mass. The solid structure of the objects is ensured by the crystal or amorphous arrangements of the ionic nuclei. The single and double helix in biological system is based on the nuclear acid base pairs. If one can follow the history of this nucleotic agents one can control the system. In wider context in cosmology stars and galaxies can play this nucleotic role. In general numerical solution of theoretical partial differential equations is achieved by discretization of space and time. The lattice nodes with definite calculation procedures can be regarded also as nucleotic objects. In a heuristic way one can define as nucleotic system those arrangements which are consisting of elements with precisely defined properties and are mainly in interaction only with other elements in their neighbourhood. This definition gives rather wide set of possibilities, the systems can have regular, amorphous, tree-like or general graph etc. structures.

According to the definition of nucleotic problems one can ensure ultrascalability, if there is a possibility to identify the so-called **principal agent** [12]. The principle agent executes the universal activity at each nucleotic site driven by the common Clock-signal. In more complex cases one can have several different types of different principal agents which are activated by special control logic at appropriate times. The procedure executed by individual principal agent can take T clock cycles corresponding to its type.

The principal agents can be regarded as vertices of a graph. The information flow is indicated by directed edges.

4.1 Bit-serial principle agent

In all the above cases each processor executes character comparison which is a two-by-two bit process, i.e. one evaluates a double-hit coincidence. Preliminary step for coincidence matrix creation in DNA principal agent is shown in Figure 12. Thus the main algorithm will work on the N*m coincidence matrix, because the two bit comparisons are restoring the symbol count length independently the coding length of the characters to N and m.

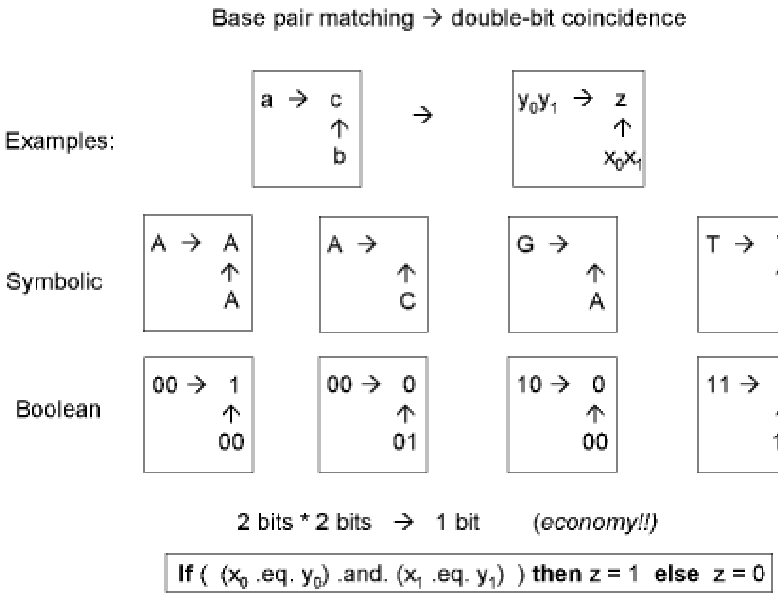


Figure 12: Preliminary step for coincidence matrix creation in DNA principal agent

The above defined single processor serial matching search algorithm can be realized in FPGA by 3 hardware elements: 2 shift-registers the first for the reference genome with length N and the second for short reads of length $\mathfrak{m}$, plus one principal agent.

The principal agent algorithm is extremely simple for each clock pulse the bit from reference genome is compared to the corresponding bit of the short read. The XOR logics provides output 1 in case of different inputs, thus the counter will be incremented by 1 if there was a mismatch, i.e. a point-error (Figure 13). The error counter is working in two-complement mode. Let us define the allowable maximal number of errors as Maxerr. At the start of each $\mathfrak{m}$-bits comparison cycle the error counter is set to -Maxerr, thus the positive value in the error counter will indicate automatically if the number of errors exceeded Maxerr.

The readout is organized through a so-called serializer to a FIFO transmitting the number of errors not exceeding Maxerr and the actual number of shifts in the reference genome to indicate the start of the matching section.

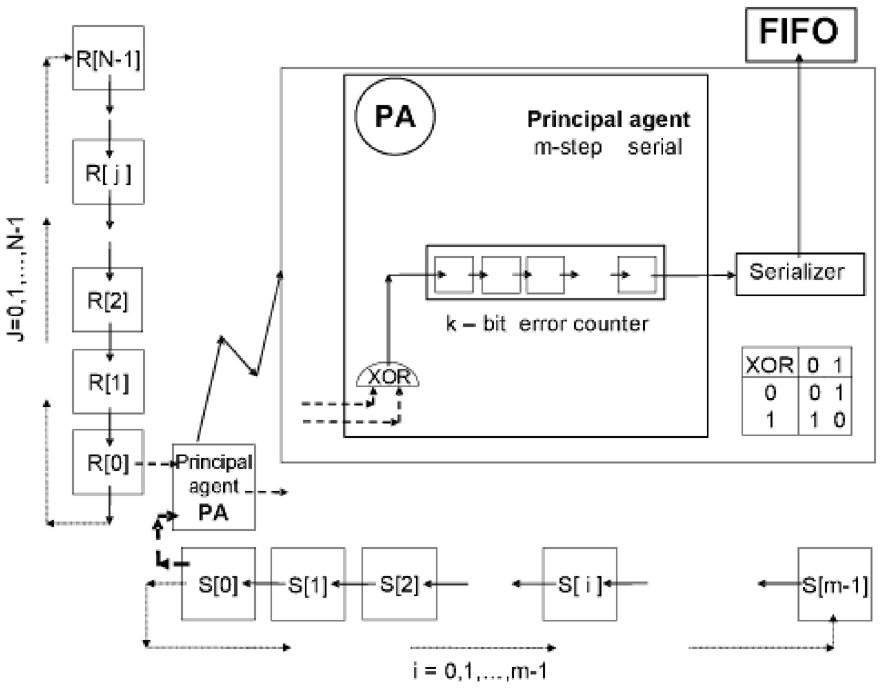


Figure 13: Bit-serial principal agent for DNA sequence alignment

One requires a serializer because due to the exhaustive search there can be more than one solutions.

In case of K principal agents the 3 main FPGA processing elements are the same. With this coarse grain design one observes a $K = N/m$ fold speed-up relative to the serial case (Figure 14). In this system each element of the reference genome is wired to one principal agent.

Here the role of serializer is more emphasized because any pair of principal agents can have simultaneous hits.

If one can afford $N - m + 1$ number of principal agents then the processing time will be independent from the length of the reference genome (Figure 15). In this system each element of the reference genome is wired to m principal agent.

It is important to remark that the 3 main FPGA elements are staying the same in all the three cases. This scaling property is an essential element of the nucleotic algorithms.

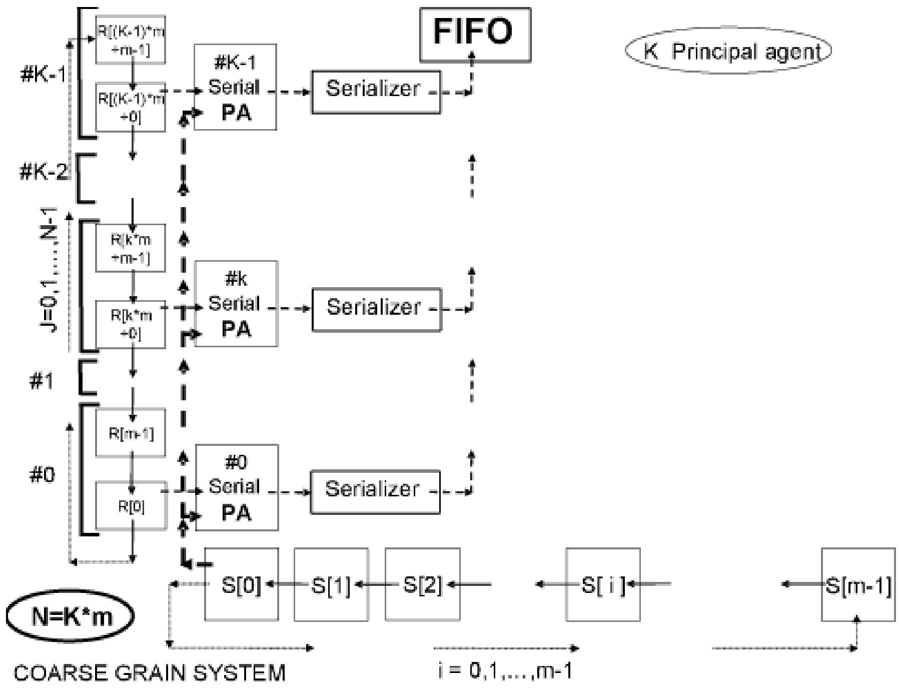


Figure 14: Coarse grain principal agents for DNA sequence alignment

4.2 ULTRASCALABILITY with bit-parallel principal agents

One can speed up the execution time and increase the efficiency of calculations by applying more complex processors. So far it was assumed that the processors were comparing one character of the reference sequence to one character of the given short read. One can perform in an FPGA (or ASIC) processor more than one comparison simultaneously.

In a special case $N = 24$, $m = 8$ and $K = 3$ the serial, coarse and fine grain systems with bit-serial principal agents are shown schematically in Figure 16.

One can apply however in the same structures instead of the bit-serial PAs so-called **bit-parallel PAs** too. In this case the exact matching can be achieved in a single clock cycle (Figure 17).

The new bit-parallel PA will have similarly simple structure, just the single XOR gate will be replaced by m pieces of XNOR gates and a m-fold AND gate to produce the exact matching trigger signal T.

This algorithm reserves the exhaustive feature of the bit-serial PA, but it

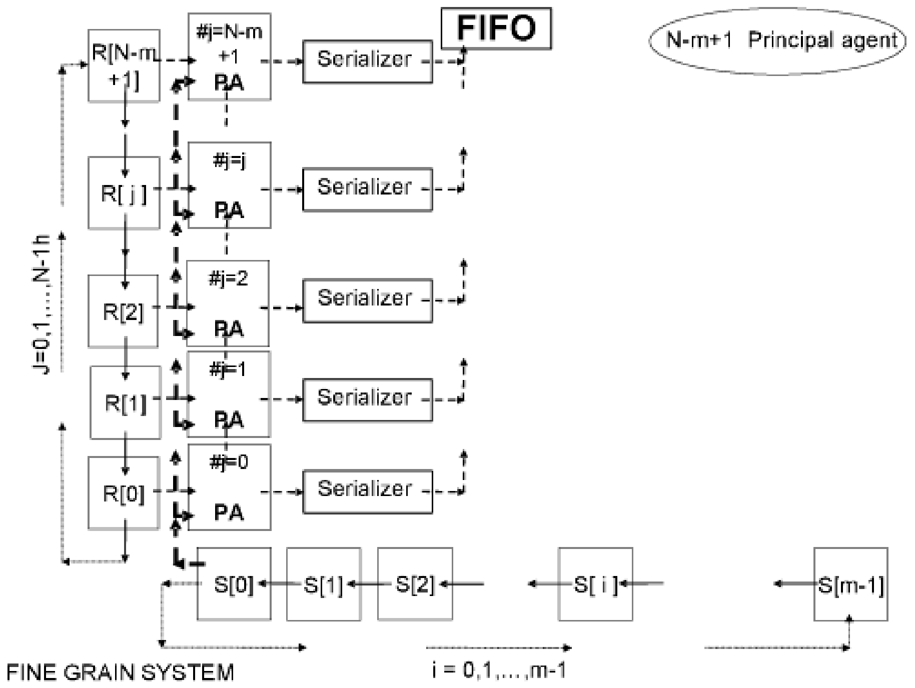


Figure 15: Fine grain principal agents for DNA sequence alignment

fails for point-errors, therefore *the speed-up was traded for performance.* One can regain part of the losses with relatively small new investments.

Let us divide the short read e.g. into $m_{reduce} = 2$ pieces and apply processors which compare m/m_{reduce} characters simultaneously. This cutting into half of the AND gate will produce m times gain is execution time and will provide extremely important information for matchings with point errors. This algorithm was proposed in [14] (Figure 18).

This simplified initial-model for DNA sequence alignment can illustrate how can one build an **ULTRASCALABLE** computer system, **where the processing time is independent from the size of the problem** if one provides the hardware which is proportional to the actual size.

In our specialized basic-model the alignment procedure can produce 3 different type of results:

a) Exact matching T=T1.AND.T2: provides the list of pointers pointing to the position of the base pair in the reference genome from where the actual short read is coinciding exactly with this part of reference genome.

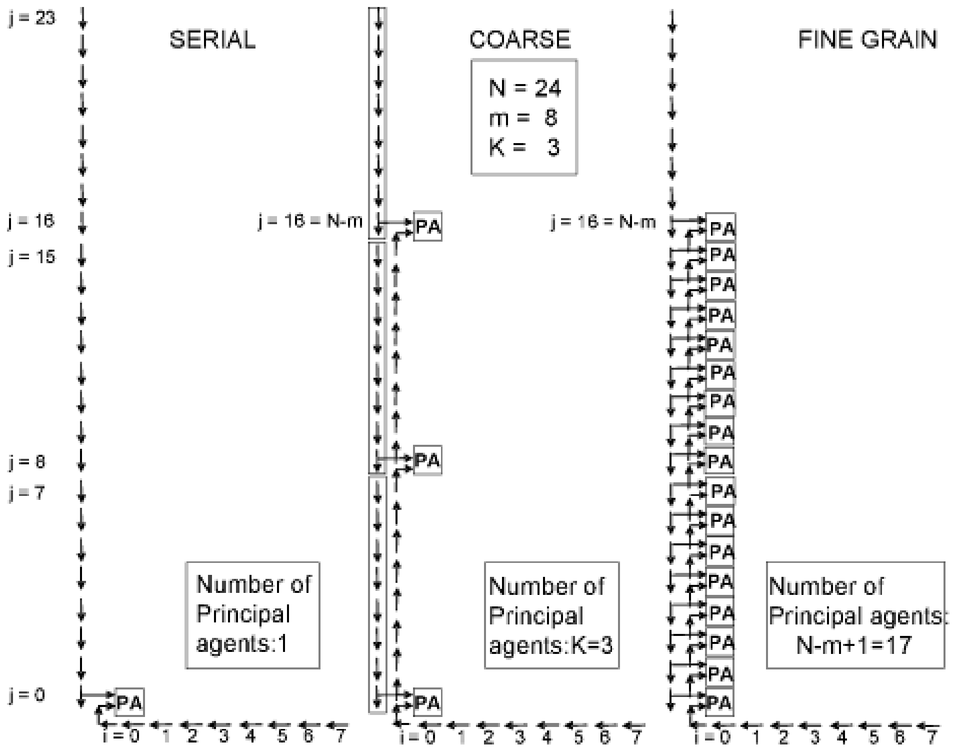


Figure 16: Summary of bit-serial principle agents

b) Half matching H=T1.XOR:T2 represents the list of exclusive OR cases, where first or second half of the short read has complete match at least of length $m/2$. If $m >> 2$ then this selection can be already very effective, therefore it is worth to sort out these cases for second part of the aligner algorithm. (If one can afford a bit more hardware for T1,T2,T3 and T4 logics then one can ensure 75% matching, allowing mismatch only in one segment shown in Figure 18 c.

c) No matching. This is the most frequent outcome. For illustration purposes intentionally we selected a combination with absolute minimal number of AND/OR gates, which simplifies the processing logics. One can easily create systems looking for more than one substitution point-errors.

One can realize the m times speed-up in both K and N parallelism.

In the K parallelism case in one clock cycle one can test the short read alignment only in those positions where the value of pointer index is a multiple

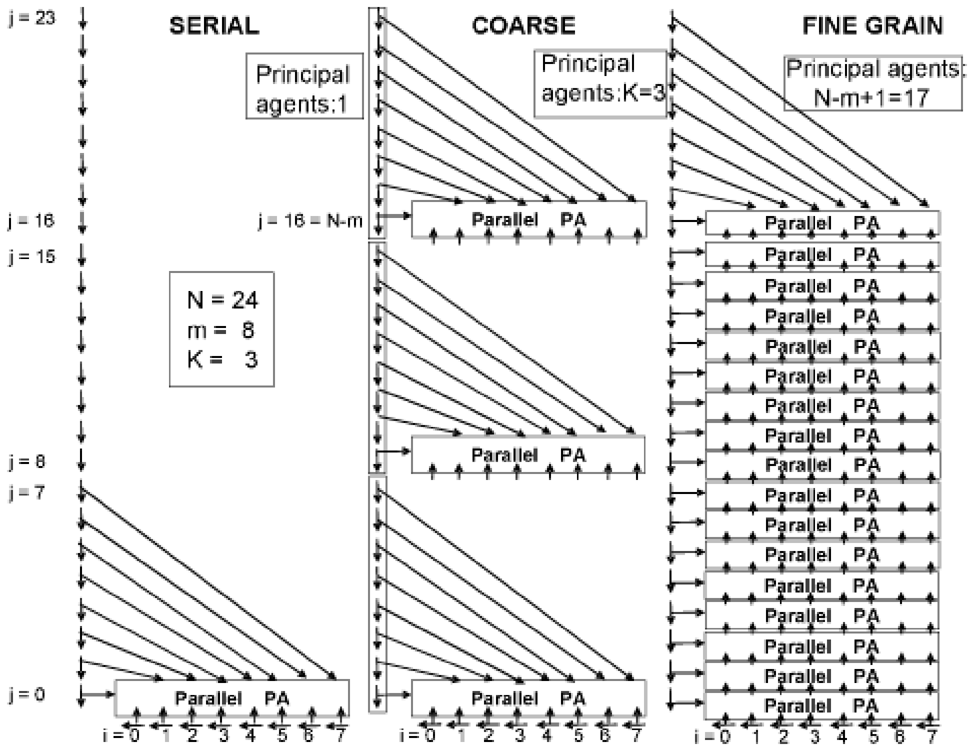
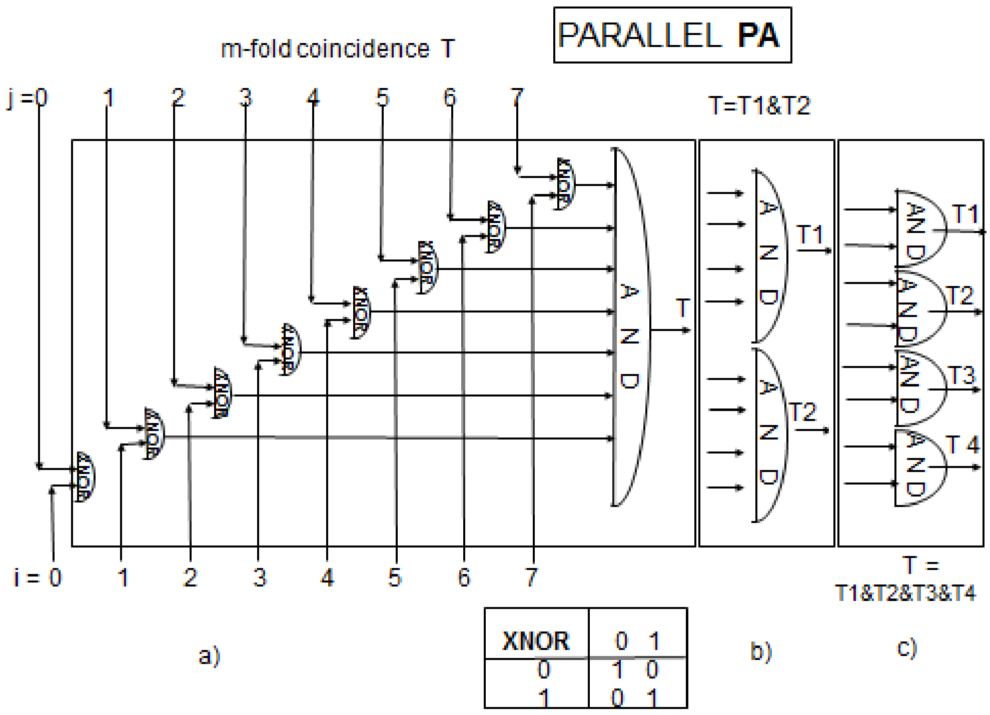


Figure 17: Summary of bit-parallel principal agents

of m, assuming that the starting index value is equal to 0. For the other values from 0 to $(m-1)$ one should make $m-1$ shifts in the reference genome and check for alignment one-by-one.

Of course, in the N parallel case one assign to each $N-m+1$ line this complex processor. Then one can get all the exact matches in single step. As additional bonus one will get a relatively short list for positions which contain all the cases where exactly one error occurred. Unfortunately there can be more than one mismatch in the indicated segment, therefore to fix this additional information a second round of tests is required. It stays however on the $O(1)$ level, because with reasonable design one can limit the expected number of multiple solutions below 10.



XNOR	0	1
0	1	0
1	0	1

Figure 18: a) Principal agent for DNA sequence alignment with m-fold coincidence for T; b) m/2-fold coincidence + 2-fold AND for T=T1*T2; c) m/4-fold coincidence + 4-fold AND for T=T1*T2*T3*T4

5 Practical demonstration

We looked at existing FPGA-based solutions for parallel short read alignment but we did not find any that attempts ultrascalability of a system although numerous papers concluded that FPGAs provide an excellent platform on which to run sequence alignment, and that clusters of reconfigurable computers will be able to cope far more easily with the vast quantities of data produced by new ultra-highthroughput sequencers[5][6].

Some solutions use higher level languages (e.g. handel-C)[3][2] which makes them easier to implement but in most cases leads to a significant decrease in efficiency. Several papers target slow but more accurate dynamic programming approaches (e.g. Smith-Waterman algorithm)[3][6]. One particular paper[5] discusses the implementation of a similar algorithm (Eland algorithm) on very similar hardware (DE2-SoC) but the implementation takes a more conventional approach involving hash functions and lookup tables which render ultrascalability unfeasible.

In this section we discuss the implementation of the algorithm introduced earlier. The source for the FPGA and the GPU implementation can be found in our public repository:

`https://bitbucket.org/exascalemultiscience/de1-soc-exaligner`

The proposed simple point-error search algorithm was realized in two different hardwares:

- a GPU system using CUDA with an Intel Core i7 CPU 920 2.67GHz processor and an NVIDIA GeForce GTX 980.
- an FPGA SoC(System on a chip) with a Dual-core ARM Cortex-A9 (HPS) processor and a Cyclone V SoC 5CSEMA5F31C6 Device with 85K Programmable Logic Elements

The performance as well as the result of the CUDA GPU system and the FPGA system was compared to Bowtie, a public short read aligner.

Here is a concise version of the algorithm:

```
open reference_file
while not reached end of reference_file
    reset hardware
    read reference_segment from reference_file
    write reference_segment to hardware
    open reads_file
    while not reached end of short_read_file
        read short_read from short_read_file
        write short_read to hardware
        wait until hardware is finished
        read and store results from hardware
    end while
    close short_reads_file
end while
close reference_file
open sam_output_file
```

```
for all short_read
    write alignment_data of short_read into sam_ouput_file
end for
close sam_ouput_file
```

5.1 FPGA implementation on DE1-SoC Board

The DE1-SoC Development Kit presents a robust hardware design platform built around the Altera System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility. Alteras SoC integrates an ARM-based hard processor system (HPS) consisting of processor, peripherals and memory interfaces tied seamlessly with the FPGA fabric using a high-bandwidth interconnect backbone. The DE1-SoC development board includes hardware such as high-speed DDR3 memory, video and audio capabilities, Ethernet networking, and much more.The DE1-SOC Development Kit contains all components needed to use the board in conjunction with a computer that runs the Microsoft Windows XP or later (64-bit OS and Quartus II 64-bit are required to compile projects for DE1-SoC) [18].

The schematic diagram of the implementation can be seen in Figure 15. In this figure the principal agents are handled as separate functional elements but in reality it is much more efficient to implement them as part of a larger functional element which carries out the computations on a large array of registers in parallel. A single instance of the prinicpal agent is illustrated in Figure 19 in detail. Each principal agent has two bits for the reference nucleotide and two bits for the short read nucleotide as input. If the output bit is 1, it means there was a match. These output bits are produced in parellel and they must be processed sequentially because the FIFO has only one input. Normally this would cause a major bottleneck, however in this problem we can discard the results with a 0 value because we don't need to process mismatches at all. This is performed by the serializers. The serializers in Figure 15 fig15(!!!!!!!!!) can also be aggregated into a larger functional element. There is some communication between the serializers in order to determine which result will be propagated to the FIFO. The number of clock cycles required for processing every principal agent output is the same as the number of matches. Most of the time there will be zero or one match in the entire reference. The case of more than one match is possible but fairly rare.

In this section ranges are always inclusive unless otherwise specified. For the implementation of the FPGA design we used Qsys and the Quartus II

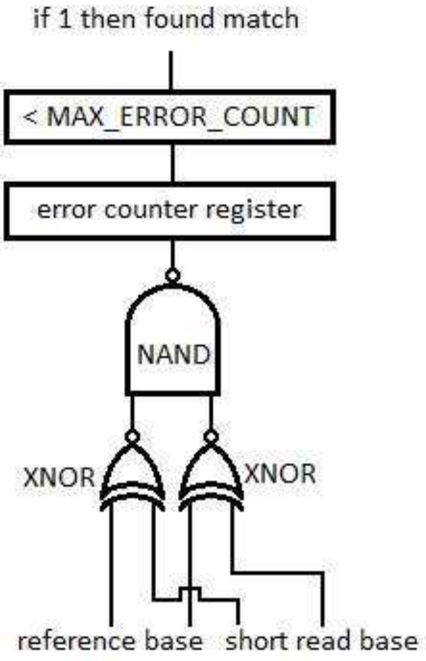


Figure 19: The single element of the fine grain principal agents

software. The HPS communicates with the FPGA fabric through a 32 bit AXI bus. The communication protocol is built according to the Avalon Memory-Mapped (Avalon MM) Interface. The interconnect between the HPS and the FPGA design is generated using the Qsys system integration tool.

Essentially, what the Qsys tool does is creating glue code, also known as interconnect, mostly consisting of buses and arbiters between the individual components of the system. For example every component has a clock input which is connected to the output of the Clock component. Another example would be the master/slave relationship between the HPS and the on-chip memory component.

There is a golden hardware reference design(GHRD) provided by the manufacturer of the DE1-SoC, Terasic. In the GHRD project there is an existing Qsys setup that was specifically designed for this device. This project perfectly matches the capabilities of the device and it can be easily extended with additional functionality.

During Avalon MM transfers the processor takes the role of the master and the FPGA accelerator logic behaves as the slave. This means that transfers are always initiated by the C program and the FPGA design reacts to it within a few clock cycles.

In the C code Avalon MM transfers are simple read and write operations at a virtual memory address, which can be calculated by adding the appropriate offset to the virtual base memory address corresponding to the FPGA accelerator logic. The offset values and the virtual base memory addresses have to be synchronised between the Qsys setup and the C code.

From the perspective of the FPGA design the memory ranges from 0 to 3 (4x32 bits). Values in this range can be encoded using a 2-bit wide value. In the C code the virtual memory offset values range from 0x0 to 0xb (4x4 bytes).

According to the communication protocol it is the responsibility of the accelerator logic to keep track of whether the next input is part of a short read or the reference. Calculations start as soon as the loading of all necessary inputs has finished. The results are then pushed into a FIFO. The C code uses polling on a designated memory address to determine whether the FIFO holds some data, i.e. a result is available to read. Reading the actual results takes place on another memory address specifically allocated for the task. The FPGA can also be reset from the C code using an Avalon MM transfer to the appropriate memory address.

5.2 GPU implementation

In the GPU implementation most of the C code is the same as in the FPGA implementation. The main difference is in the communication between the two components (CPU and GPU). Instead of explicit transfers through an interface the CPU has indirect access to the allocated memory where the GPU calculations take place. Before and after a kernel call the data has to be copied between the CPU memory and the GPU memory. Though it is considered good practice to over-issue work to the GPU to help to the device memory latency.

From an algorithmic perspective we can observe a correspondence between principal agents in the FPGA and CUDA kernel threads. We applied the same pseudo code (Section 5.) in both cases, despite the fact that the GPU would be able to scan the entire string in the memory at one time.

5.3 Comparison

Bowtie numeric vs Boolean: There is no difficulty in finding one match if there is any. The parallel nature of Boolean approach guarantees the exhaustive search whereas in Bowtie it would need special effort to look for multiple solutions. Of course, the existence of multiple solutions requires additional programming to serialize the candidates.

In case of the heuristic Bowtie and its more advanced versions the processing of almost good matchings with few number of errors is a real challenge. One of the possible solutions is the backtracking in BWT which means a trial and error procedure to look for exact matching artificially modifying characters in the measured short reads assuming recording errors in the given position.

GPU vs FPGA: Programming in GPU and FPGA requires completely different methodology. One should learn new languages CUDA (or OpenCL).

Nucleotic nature of the problem: extreme simple algorithms.

Hardware difference: For the FPGA minimal resources are required and calculations are executed in the memory cells, whereas in the GPU system complete threads are sacrificed, essentially a small CPU is used for each PA.

GPU threads have separate local memory and a very much reduced but still rather complex mini-CPU for real and integer operations. Few number of threads per square cm of silicon.

In FPGA memory cells and logical gates are together in logical elements ideal for bit level programming, one could say that calculation is performed inside the memory cells. The logical elements are relatively universal, but simple enough not to waste silicon surface for unused resources. One can have millions of logical elements per square cm on silicon.

The use of ASIC hardware can be even more economical. One can design only with the minimally necessary memory cells and gates, thus no unused elements are sitting in reserve. Its density can reach billions per square cm on silicon.

The scalability problem occurs for both architectures because the same signal should be delivered to more and more elements. FPGA and ASIC have a much larger density per chip resulting in shorter transmitting routes!! Furthermore the network topology of such a system can be very flexible as it can conform to the custom data-flow of any algorithm while GPUs have a fixed infrastructure for transmitting data.

In case of nucleotic problems the network consists of mainly two components:

In first case there are connections only between neighbouring principal agents. This structure is ideal for ultrascalability, because one can increase the network by simple connections at the edges.

The other part of network is given by a few global transmissions propagating information to a large number of elements.It is not a problem to send signals on a single line to 10 destinations, but the signal propagation becomes questionable if one aims for millions of destinations with a single signal. In principle, one can use a binary tree instead of a single line with appropriate amplifying elements. Of course, the expansion of this binary distribution tree is much more complex between chips and boards than inside a chip. The same logics can be applied in GPU, FPGA and ASIC principal agents, but it is obvious that it costs practically nothing in energy and cost at ASIC, but can be prohibitive in case of GPU boards.

The I/O capabilities in FPGAs are larger by several orders of magnitude (due to the pin counts) making it easier to incorporate it into a larger system.

Longer clock cycles are a serious disadvantage in FPGA and ASIC systems but as the technology is advancing they are approaching speed of traditional CPUs. The clock frequency also depends on the timing constraints of the RTL design.

The number of FPGA processing units can not be increased further (as it was explained in Section 5.3) due to a hardware limit, over up to these values we can extrapolate, assuming the same scale behaviour. GPU have been possible to go further, but we did not want to compare the theoretical values with the measured run time. We might expect that longer chains and multi-thread processor, the GPU efficiency will grow roughly up to 4-8 times of the processing units, which reaches its peak efficiency and run time does not improve in the future.

The table below illustrates the difference between the running time of the FPGA and GPU implementations. The first column contains the number of principal agents/threads used in each run and the second and third columns contain the running times for the Lambda phage example detailed in a later section. It only takes 1024 principal agents to run faster than the GPU due to the ultrascalability property of the FPGA implementation. In the GPU implementation the more threads the longer it takes to evaluate the result of every thread because it must be performed serially by the CPU. It would be interesting to compare the two solutions using an even higher degree of parallelization but due to the limited capacity, a design with more than 1024 principal agents doesn't fit in the Altera Cyclone V FPGA.

The runtimes of introduced method on FPGA and GPU are shown in Table 1. In comparison if we run the algorithm sequentially it takes 147940.670 seconds (approximately 41 hours).

	FPGA[s]	GPU[s]
64	2427.410	2282.875
128	1214.420	1159.218
256	608.870	588.058
512	304.890	298.164
1024	154.280	157.256

Table 1: Runtime in FPGA and GPU

5.4 Simple example

In this section we describe the different implementations of the exaligner algorithm.

Running the CUDA application is extremely simple. One should execute the following command:

- exaligner-gpu example_reference.fa example_reads.fq example.sam

The FPGA application is very similar:

- exaligner-fpga example_reference.fa example_reads.fq example.sam

With Bowtie it is a little different:

- Command line for index creation:

  bowtie2-build reference/example_reference.fa example_reference

- Command line for sequence alignment:

  bowtie2 -x example_reference -U reads/example_reads.fq -S example.sam

Relevant positions in the output sam file:

Position #1.: short read ID
Position #2.: bitset, possible values in our case: 0,4,16; if 0, then forward maching; if 16, then reverse matching; if 4, then no alignment found
Position #4.: aligned reference genome position, indexing starts with 1.
Position #10.: short read sequence

The first few lines of the example.sam output file:

```
@HD VN:1.0 SO:unsorted
@SQ SN:example_reference1 LN:420
@PG ID:Exaligner VN:1.0 CL:'./exaligner-fpga example_reference.fa
example_reads.fq example.sam'
example_read1 0 example_reference1 5 42 16M * 0
              0 TGATGGTCGTCCATTA .:7@3<6&10EG2<7<
example_read2 16 example_reference1 4 42 16M * 0
              0 TTGATGGTCGTCCATT <7<2GE01&6<3@7:.
```

Below is the first matching short read from the example_short_reads.fq file. The last row describes the sequencing quality and can be ignored.

```
@example_read1
TGATGGTCGTCCATTA
+
.:7@3<6&10EG2<7<
```

In this example the following simple reference file was used:

```
> example_reference1
GCCTGTATGGTCGTCCATTAAGTACGCTAAGTCACAGCGCGCTGC
GCCAGGGCGTGGCAATGGTGCAGCAAGATCCGGTGGTGCTGGCGG
ATACCTTCCTCGCCAACGTGACGCTGGCACGTGATATCTCTGAAG
AACGCGTCTGGCAGGCGCTGGAAATCGTGCAGCTGGCGGAGCTGG
CGCGTAGCATGAGTGATGGTATTTACACGCCGCTTGGCGAGCAGG
GGATAAATCTCTCAGTCGGGCAAAAGCAACTGCTGGCACTGGCGC
GCGTGCTGGTGGAGACGCCGCAAATCCTGATCCTTGATGAGGCAA
CCGCCAGCATTGACTCCGGGACTGAATAGGCGATTCAACATGCTC
TGGCGGCGGTGCGTGAACATACTACGCTTGTGGTGATTGCTCACC
GCTTATCAACTATTG
```

The .sam output of the FPGA and GPU applications are identical. One should not expect identical results from Bowtie because its algorithm uses heuristics and the output is random. However, as the next section describes the output files compare favourably.

5.5 Lambda virus

In this article we introduced an algorithm using the advantage of the FPGA options to determine the DNA sequences. We present below the case of Lambda virus, because the DNA molecule of 48502 base-pairs is linear.

In 1950, Esther Lederberg an American microbiologist, who performed an experiments on E.coli mixtures. His research led to employment of Lambda phage as a model organism in microbial genetics as well as in molecular genetics [8].

In this article we study the short reads each comparison performed by the reference sequence, are determined for exact matching, 1, 2, and 3 cases of error.

We ignore the indel ie. the insertion, when an extra element appears and the deletion case, when an item is missing in the test sequent.

We calculated the distribution of short reads over Lambda virus, where the length of short reads is 50. The number of short reads depends on the reference position, which was calculated by exaligner algorithm (5.5 Section). This method is able to accurately determine individual cases of error occurrences.

The generated sample file is created by wgsim program [15]. The exact matching and 1 cases of error have been shown in Figure 20. The 2 and 3 cases of error have been presented in Figure 21. The generated and real [16] sample string of Lambda phage was studied by Bowtie algorithm also, which was shown in Figure 22.

Since the set of measurements considered as random sequence, therefore we can characterize this series with the expected value, standard deviation and the correlation coefficient in the Table 2.

There are significant difference of frequency value between the faul (Figure 20 (a)) and the exact matching (Figure 20 (b)) release. The correlation coefficient changes significantly in this case.

	0	1	2	3	B-g	B-r
0	1	0.6144	0.4194	0.3792	0.4660	0.0241
1	0.6144	1	0.8215	0.7708	0.8502	0.0391
2	0.4194	0.8215	1	0.9854	0.9718	0.0218
3	0.3792	0.7708	0.9854	1	0.9612	0.0159
B-g	0.4660	0.8502	0.9718	0.9612	1	0.0227
B-r	0.0241	0.0391	0.0218	0.0159	0.0227	1

Table 2: Correlation coefficient (B-g: Bowtie alg. on generated string, B-r: Bowtie alg. on real string)

The FPGA method is reconfigurable and scalable, so this algorithm can be developed further to find the indels and more complicated and longer DNA sequence.

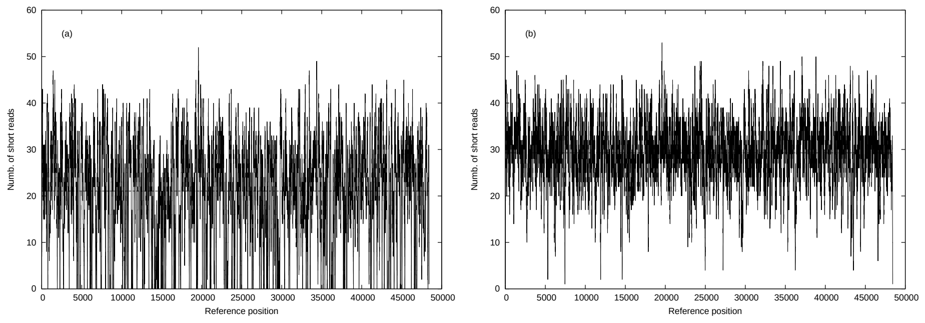


Figure 20: The diagram for DNA sequence alignment, which consist of 0 (a) resp. 1 (b) error

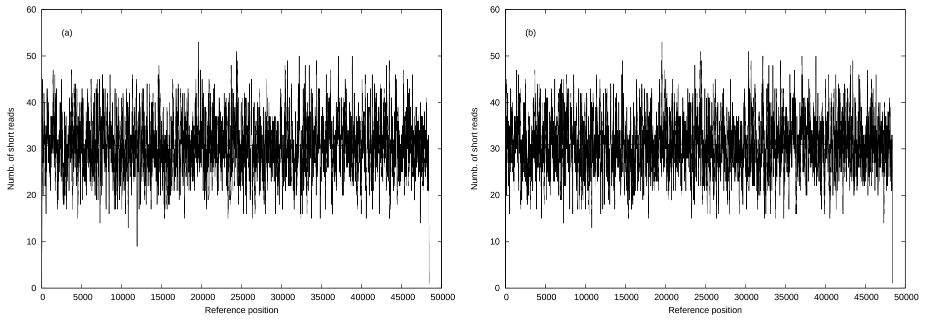


Figure 21: The diagram for DNA sequence alignment, which consist of 2 (a) resp. 3 (b) errors

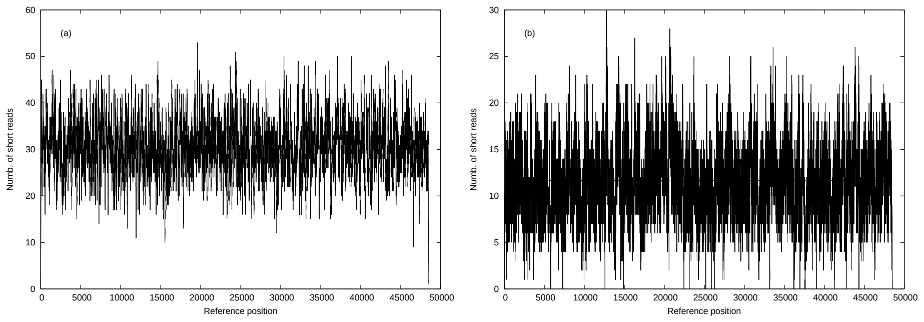


Figure 22: The diagram for DNA sequence alignment using Bowtie alg., which consist of the generated sample resp. (a) real string (b)

6 Summarize

In this article, we introduced a new nucleotid method that is suitable for processing large amounts of data, which is close to the hardware algorithms (FPGA). We are shown in case of DNA sequenceces of lambda virus to use the exaligner procedure. This method is reconfigurable and rescaling, therefore it can be developed on the more effective tools to study much larger database as the human genom sequence.

7 Acknowledgements

The authors thank prof. I. Csabai for useful discussions and valuable remarks.

References

[1] L. B. Alexandrov, Serena Nik-Zainal, et al., Signatures of mutational processes in human cancer, *Nature* **500 (7463)** (2013) 415-421. ⇒152
[2] J. Arram, K. H. Tsoi, Wayne Luk, P. Jiang, Hardware Acceleration of Genetic Sequence, Chapter: Reconfigurable Computing: Architectures, Tools and Applications, *Lecture Notes in Comp. Sci.*, **7806** 13–24. ⇒173
[3] K. Benkrid, Liu Ying, A. Benkrid, A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment, very large scale integration (VLSI) systems, *IEEE Transactions* **17,** 4 (2009) 561–570. ⇒173
[4] M. Burrows, D. J. Wheeler, **124** (1994), *A block sorting lossless data compression algorithm*, Technical Report, Digital System Research Center. ⇒152, 155

[5] Y. S. Dandass, S. C. Burgess, M. Lawrence, S. M. Bridges, Accelerating string set matching in FPGA hardware for bioinformatics research, *BMC Bioinformatics* **9** (2008) 197. ⇒173

[6] R. K. Karanam, A. Ravindran, A. Mukherjee, C. Gibas, A. B. Wilkinson, Using fpga-based hybrid computers for bioinformatics applications, *Xilinx Xcell Journal* **58** (2006) 80–83. ⇒173

[7] B. Langmead, C. Trapnell, M. Pop, Sl. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome *Genome Biol.* 10:R25., ⇒152

[8] E. Lederberg, Lysogenicity in Eescherichia coli strain K-12, *Microbial Genetics Bulletin*, **1** (1950) 5–8. ⇒182

[9] J. von Neumann, *First Draft of a Report on the EDVAC* pp. 149. University of Pennsylvania, June 30. 1945. ⇒165

[10] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, *Journal of Molecular Biology* **147** (1981) 195-197. ⇒154

[11] G. Vesztergombi, 'Iconic' tracking algorithms for high energy physics using the trax-I massively parallel processor, CHEP, *Computer Physics Communications*, **57** (1989) 290–296. ⇒166

[12] G. Vesztergombi, One billion processors program's demo on FPGA emulator board, IEEEXplore, ReConFigurable Computing and FPGAs (ReConFig), (8–10 Dec. 2014) International Conference, Cancun. ⇒166

[13] ∗ ∗ ∗ Burrows-Wheeler Transform Discussion and Implementation Homepage: http://michael.dipperstein.com/bwt/ ⇒158

[14] ∗ ∗ ∗ EXAMS project submitted to EU call: FET-Proactive – towards exascale high performance computing H2020-FETHPC-2014. Publication date 2013-12-11 Deadline Date 2014-11-25 17:00:00. Specific challenge: The challenge is to achieve, by 2020, the full range of technological capabilities needed for delivering a broad spectrum of extreme scale HPC systems. (Private communication) ⇒ 170

[15] ∗ ∗ ∗ Generated sample: https://github.com/lh3/wgsim. ⇒182

[16] ∗ ∗ ∗ Real Lambda phage Homepage: `ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF_000840245.1_ViralProj14204/GCF_000840245.1_ViralProj14204_genomic.fna.gz` ⇒182

[17] ∗ ∗ ∗ NVBIO: nvBowtie, 2015,
Homepage: http://nvlabs.github.io/nvbio/nvbowtie_page.html. ⇒152

[18] ∗ ∗ ∗ Terasic – DE Main Boards, datum, Homepage: http://de1-soc.terasic.com. ⇒175

Received: September 15, 2015 • Revised: December 22, 2015

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 186–199

DOI: 10.1515/ausi-2015-0018

Sapiness–sentiment analyser

Katalin Tünde JÁNOSI-RANCZ
Sapientia Hungarian University of Transylvania
Dept. Mathematics and Informatics, Târgu-Mureş
email: `tsuto@ms.sapientia.ro`

Zoltán KÁTAI
Sapientia Hungarian University of Transylvania
Dept. Mathematics and Informatics, Târgu-Mureş
email: `katai_zoltan@ms.sapientia.ro`

Roland BOGOSI
Sapientia Hungarian University of Transylvania
Dept. Mathematics and Informatics, Târgu-Mureş
email: `root@rolisoft.net`

Abstract.
In our ever-evolving world, the importance of social networks is bigger now than ever. The purpose of this paper is to develop a sentiment analyzer for the Hungarian language, which we can then use to analyze any text and conduct further experiments. One such experiment is an application which can interface with social networks, and run sentiment analysis on the logged-in users friends' posts and comments, while the other experiment is the use of sentiment analysis in order to visualize the evolution of relationships between characters in a text.

1 Introduction

Sentiment analysis [10] is a technique used to determine the amount of positive and negative sentiment in a piece of text. Accurate opinion mining software

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R15
Key words and phrases: Social networks, sentiment analysis, semantic web, prediction, classification

is very desirable due to the many uses it can serve, however this is one of the topics which does not easily carry over, as new parsers, dictionaries and neural networks have to be developed, compiled and trained, all while accounting for the features of the language.

Our sentiment analyzer targets the Hungarian language. Many challenges have presented themselves during the development, one of the main challenges was the handling of negation within a sentence. This is particularly hard, as without understanding the meaning of the words, a language parser will have a hard time deciding where to start and when to end a negation.

Suppose we are examining the following sentence:
"A kijelző minősége nem a legjobb és a kamerája is hagy kívánnivalót maga után, de ár/érték arányban verhetetlen a piacon!" ("The quality of the display is not the best and its camera also leaves something to be desired, but in value for the money it is unbeatable on the market!")

When analyzed by a human, this sentence is mostly neutral, as the device referred to in the sentence has both its ups and downs. However, an application will have trouble deciding how does the negation apply in the sentence above. Different implementation methods will yield different results. As described in section 3.2, if we try to invert the polarity of a fixed number of subsequent words, we will end up with a sentence which has a negative half, followed by a positive half, yielding a neutral sentence overall. However, if we decide to invert the polarity until the end of the sentence, we will get a negative sentence as the overall score. Unfortunately this can go both ways, as there are examples where the other implementation deems to be more accurate.

The Hungarian language (orthography) uses diacritics, and there are words which can have different meanings when written with and without diacritics. Throughout one of our experiments outlined in section 4.1, we have observed that the majority of posts and comments do not use diacritics. In an even worse situation, a sentence may contain mixed use of diacriticized and non-diacriticized words that should have been written with diacritics. In a sentence where diacritics are not used, and the word can have multiple meanings without the possibility of us distinguishing between them, we have to average the polarity of both meanings. However, in a mixed sentence, no averaging will be done, and the essentially misspelled word's meaning will be used, which may erroneously impact the overall score of the sentence.

Ambiguous words end up in the same boat as words with diacritics that were not diacriticized, however in this case it is not the user's fault, it is the fault of the language parser. Without knowing the proper meaning which the author intended to use in case of ambiguous words, we will have to average the polarity of the meanings, and use that. Unfortunately this is not an accurate solution, but a better solution is not possible with our implementation at this time.

Slang usage is another challenge, however this one can be solved by mapping the slang words to their non-slang counterpart. While this solves the issue, this also means an up-to-date database has to be kept with all the slangs, otherwise meanings may be missed when assessing sentences from the Internet.

Different domains may use words as different technical terms, and may even end up redefining the connotation of said words as a result. In order to correctly assess the connotation of a sentence in a specific domain, the application needs to be re-trained with regards to the connotation of the technical terms that it may encounter.

Sentences containing sarcasm and irony cannot be accurately assessed with simple language parsers. This results in a rather significant issue, as the use of sarcasm generally means the connotation of the sentence is completely inaccurate, as the meaning should have been negated, which is another challenge in and of itself. The sentiment analyzer presented within this paper does not address the use of sarcasm and irony.

2 Related work

In the field of Computer Science, Natural Language Processing is a wide subject, which has been broadly discussed. Most of the research done focuses on the English language, however due to the difference between the languages, the solutions proposed and implemented in those papers may not be easily applied to languages they did not focus on, as such this subject is highly language-dependent. The ascent of social media has attracted significant interest in sentiment analysis [10] techniques such as opinion mining.

Experiments with sentiment analysis which also use SentiWordNet[1] as a lexical resource, but focus on a language other than English have been conducted. A paper which discusses opinion mining user-generated movie and hotel reviews in the Portuguese language is [7], while a similar one exists for evaluating French movie reviews in [6]. A paper comparing various methods for sentiment analysis for the German language is in [12].

However, since opinion mining mainly targets user-generated content on social media, the use of humor, sarcasm and irony is rampant[13]. A paper targeting sentiment and irony analysis for the Italian language in [4] observes how users of social media generally use humor and irony, and how this affects methods used in sentiment analysis. In [14] a corpus of tweets is presented, where every tweet is annotated with an associated emotion, and can be used for further testing in this regard.

A publicly available corpus for the Hungarian language exists under the name of OpinHuBank[11], however we saw it unfit for our purposes as we are not doing entity-oriented analysis. The values assigned to sentences within OpinHuBank are -1, 0 and 1, which does not fit into our use case, as we would need to know the extent of positive and negative connotation. We ended up building our own corpus, as described in section 3.2.

A number of papers have been published previously which discuss sentiment analysis with a focus on the Hungarian language[16, 8]. In [3] the interaction of users is being analyzed and used to enhance traditional language processing techniques.

3 Methodology

3.1 Lexical approach

For the first version of the application, hereinafter referred to as *v1.0*, we took a lexical approach, with a database where every word is associated with a value representing its positive, negative and neutral connotations.

Since we were unable to find a fitting database openly available for the Hungarian language, but found enough for the English language, we decided to adapt an existing English database to Hungarian as a start. SentiWordNet[1] was ultimately found to be a fitting database whose structure suits our purposes. Their database contains 117,659 words in English with thesaurus attached and the words annotated with their polarity.

As we only had to do dictionary look-ups, we turned to Google Translate's API for translating the batch of words. Even though the translation was successful, and in theory it should have worked fine, upon inspecting the end result, we noticed multiple issues. The machine translation did not account for the correct meaning of words with multiple meanings, and as such most synonyms have been translated to the first Hungarian meaning, regardless what the actual meaning would have been. In an extreme edge-case, 30 different English words were translated to the same Hungarian word. To solve this, we tried to find synonyms for these Hungarian words. The translator would also ignore the part of speech of the translated word, for example the verb "*(to)* duck" would be translated to the noun "kacsa". In other cases, the assigned polarity would get invalidated, as the translated word does not share the same semantic orientation as the original, such as the relatively negative attribute "cheesy" would get translated to the neutral word "sajtos" in Hungarian.

After the failed machine translation attempt, we decided to manually translate the whole English database, carefully accounting for different meanings, part of speech, synonyms, and so on. This was a long, tedious and time-consuming operation, as even though we filtered the database to exclude words without significant polarity attached, we still had to manually comb through 50,973 words. Second step in the process was the processing of the input text we receive from the user, which we will query against the database. As we needed a Hungarian word stemmer, we used a library called magyarlánc[18], which was developed at the University of Szeged as a complete toolkit for linguistic processing of Hungarian texts. The database of the library was not complete at the time, and as such we had to extend it in order to support many other words. One example would be the word "román" (adj. "Romanian") which was stemmed to "rom" (n. "ruin") even though that is not the correct stem.

The web application, whose backend was developed in PHP, would process the input from the user, look up the polarities of the participating words, and compute the final arithmetic mean from their sum, in order to determine whether the specified text carried a predominantly positive, negative or neutral connotation.

This version is rather primitive as far as what is possible in the field of natural language processing. We compared it to several commercial products in subsection 4.2.

3.2 Neural networks

For the second version of our application, henceforth referred to as *v2.0*, we started from scratch as far as the algorithm is concerned, and ventured into the field of neural networks.

We greatly improved the Hungarian translation of our database in the meantime between the two versions, and then we went on to experiment with training a supervised learning linear classifier. After a few trial and error runs, we found the suitable configuration of the neural network to be consisting of 8 input values, 2 hidden layers and 1 output. The 8 input values are the positive and negative sentimental values, separately, of the verbs, nouns, adjectives and adverbs from the database. When given a sentence, we compute the sum of the sentimental values of verbs, nouns, adjectives and adverbs in the sentence, which is then fed to the input. The output of the neural network is a value between $[-1, 1]$ indicating the polarity of the specified sentence. For training, we used an annotated dataset consisting of 500 sentences from the Sentiment Treebank [15] which contains movie reviews.

While the initial results of the new application were promising, we set on to fine-tune the neural network by feeding it further data. However, in order to do this, first we had to compile another suitable training set.

In order to create a corpus in Hungarian for use as a training set for our sentiment analyzer application, we created a user-friendly web application whose purpose was to let prospective visitors give their feedback with regards to what level of positive, negative or neutral connotation they think each displayed sentence had. The sentences to be annotated consisted of user reviews fetched from various web sites, including car and hotel reviews, as well as the opinion of famous people about various topics. The sentences were stripped of any names, including brands and personal names. The annotation is a 2-way polarity with a numeric score for positive and negative sentimental value each. A total of 70 students used the application. The list of sentences to be annotated was generated in such a way, so as to always list the sentences with the least amount of annotations first, out of the 500 total, in order to ensure no sentences would be left without annotations. Students each annotated on average 60 to 120 sentences, resulting in each sentence being annotated at least 10 to 13 times.

The existing neural network was further trained with the 500 freshly annotated sentences. The application featuring the improved version of the neural network will be referred to as *v2.1* in later benchmarks, so as to easily distinguish the progress and improvements between the two networks.

We also implemented negation in this version. While there is no agreed-upon standard way to handle negation in a sentence, as described in papers [5, 9] the typical way to handle it is by inverting the polarity of the words surrounding the negation. As for the number of words to invert, aforementioned papers list the following possibilities to consider:

- Negate the sentence until the end.
- Negate a fixed number of subsequent words.
- Invert the polarity of the first sentiment-carrying word.
- Invert the polarity of the next non-adverb word.

We have determined the best approach is to invert the polarity of a fixed window of 3 words immediately following the negation, with the exclusion of stop words, and thus this is the solution we have ended up implementing in the application.

After much work on improving the efficiency of the sentiment analyzer, we have heavily benchmarked the results, which are available in subsection 4.2.

4 Experiments

In order to use the sentiment analyzer, we have come up with a few experiments which rely on real-world data and vary greatly in order to test all the edge-cases and implementation difficulties outlined in section 1.

4.1 Social network analysis

The idea of this experiment is to provide a web interface, on which the users can log in, and query a phrase they are interested in, after which the application will use data from the user's connected social networks, and determine the sentiment of the user's friends regarding the given query.

For the social network component, we have chosen to implement Facebook, as after the user has authorized us to do so, we had the ability to fetch the public posts and comments of the user's friends. As a result, a user may log in to our web interface, and they may search for a phrase, such as a movie or brand name, and the application will determine whether the friends of the user are mostly positive or negative with regards to it, if there are enough posts available regarding the queried phrase.

A few of the issues we have faced included the language variety, as not all the posts and comments of the examined users were in Hungarian. Secondly, the posts which were in fact in Hungarian, varied greatly on the use of diacritics: some were fully diacriticized, some were not at all, and some used diacritics interchangeably. If a text did not contain any diacritics, it is assumed to have been written without diacritics in the first place, and we implemented a special handling for those. We did not try to restore diacritics into the words, instead we generated a list from the database of the words which have the same letters when written without diacritics, and took the average of their polarity. In situations where diacritics were used interchangeably, the analyzer determined the sentence to have been written originally with diacritics due to the occurrence of at least one diacritic. In such cases, words which had meanings in both diacriticized and non-diacriticized forms were not properly accounted for. Last but not least, a significant amount of posts contained misspellings or unrecognized slang, which we had to try and solve by updating our synonyms database. We ended up manually checking 17,759 words and assigning them synonyms which were not yet in the database at that point.

We also implemented a complementary web browser extension for Google Chrome, which allowed users to select any phrase from any web page and perform sentiment analysis on it with our application, while also getting results on the sentiment of the user's friends regarding the queried phrase.

It should be noted that shortly after conducting the experiment, the API version used by our application (v1.0) was deprecated by Facebook on April 30th of 2015, thus we would be forced to switch if we were to continue the experiment. Newer versions of the Facebook API do not support the permissions we require in order to fetch posts and comments.

4.2 Comparison results

In order to determine the precision of our sentiment analyzer and get a progression indicator of our improvements, we continually tested our application against a few widely known existing implementations:

- *Alchemy API*[17] is a commercial product developed by IBM specializing in providing natural language processing services to developers via an API. The list of exposed services includes sentiment analysis.
- *Sentiment Treebank*[15] is a sentiment analysis application developed by the Stanford University. Their approach is to train a recursive neural network using a training set generated by users assigning a polarity to movie reviews.

For testing, we used $n = 100$ sentences containing movie reviews from imdb.com. After translating them to Hungarian we had two parallel sentence-sequences to compare the three applications. We denote by a_i, tb_i, s_i, $i = \overline{1, n}$ (**A**lchemy, **S**entiment **T**reebank, **S**apiness) the resulted sentiment polarity sequences. The performance of our in-house application against the aforementioned services is outlined in table 1.

The mean of deviation sequence between the two tested applications ($|a_i - tb_i|, i = \overline{1, n}$) is 32.29%, while the means of deviation sequences between our application and tested ones ($|a_i - s_i|, |tb_i - s_i|, i = \overline{1, n}$) range from 27.31% to 33.66%, well within the range of comparable results.

We conducted an experiment in which we take a 100 random sentences and their annotations from OpinHuBank's[11] database, and consider them to be the golden standard when testing against Sapiness. Out of the 100 sentences,

Comparison	The means of deviation
Alchemy – Treebank	32.29
Alchemy – Sapiness v1.0	52.16
Alchemy – Sapiness v2.0	36.54
Alchemy – Sapiness v2.1	33.66
Treebank – Sapiness v1.0	40.51
Treebank – Sapiness v2.0	30.34
Treebank – Sapiness v2.1	27.31

Table 1: Comparison of Sentiment Analyzers

Sapiness computed a correct sentimental value for 72 sentences, resulting in 72% correctness.

4.3 Plot evolution analysis

The main goal of our project is to detect and examine the informal e-communication network of the employees of Sapientia University (Faculty of Technical and Human Sciences), and to make organizational management suggestions based on this analysis (including the sentiment analysis of the members' e-messages). Since the employees of our faculty are not significantly present on social networks and we have not yet received permission to their email communication contents, we decided to test Sapiness by examining the relationship between characters from the bible.

In this experiment we tried to aim for a more advanced use of the analyzer, and made it our goal to analyze a character's evolution within a story as the plot progresses. We ran the analysis on a modern translation of the bible, examining the relationship between the characters, their interactions and relative emotional trajectory. For the purposes of this paper, we are going to present our findings when analyzing the relationship between David and God.

When preparing the text for processing, we removed all sentences which were irrelevant. We define "irrelevant" in this case as a sentence which does not have both the words "David" and "God" occurring in it. In such cases, we cannot infer a sentimental connotation onto these characters. The selected verses (containing the names of God and David and reflecting their relationship) were arranged in chronological order to characterize the relationship of the two characters over the time (horizontal axes). We assumed that the sen-

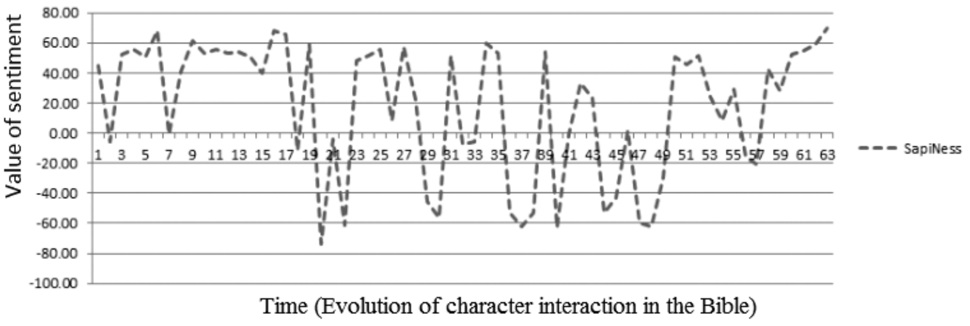


Figure 1: Relationship between David and God as determined by Sapiness

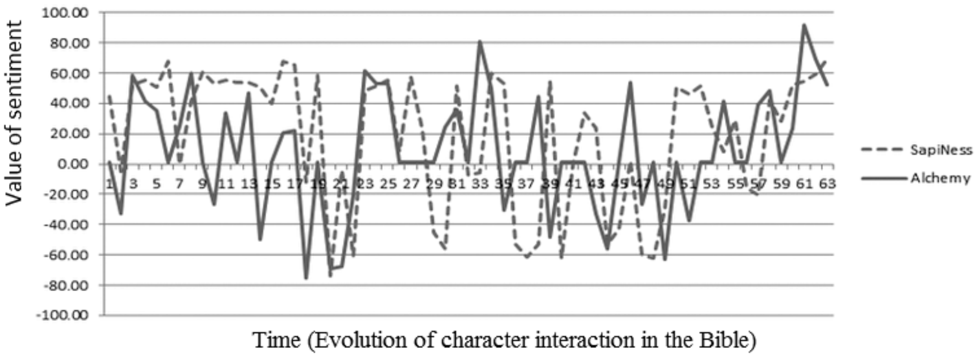


Figure 2: Comparison between Sapiness and Alchemy API

timent value sequence of these verses will model the analyzed relationship appropriately. As we expected, negative sentiment value sub-sequences reflect negative periods in the relationship between God and David. The analysis was done on a verse-by-verse basis, meaning that we assigned a sentimental value to each verse separately. If the verse contained multiple sentences, we calculated the sentimental value of each sentence, and then used the arithmetic mean of these values as the final sentimental value of the verse.

After running the analysis, we plotted the initial results in figure 1.

In order to benchmark the precision of our sentiment analyzer, we tried to re-run the analysis using a commercial application intended for this purpose, namely Alchemy[17]. As the chosen web service did not support the Hungarian language, we had to search for an English translation, which also used a modern language.

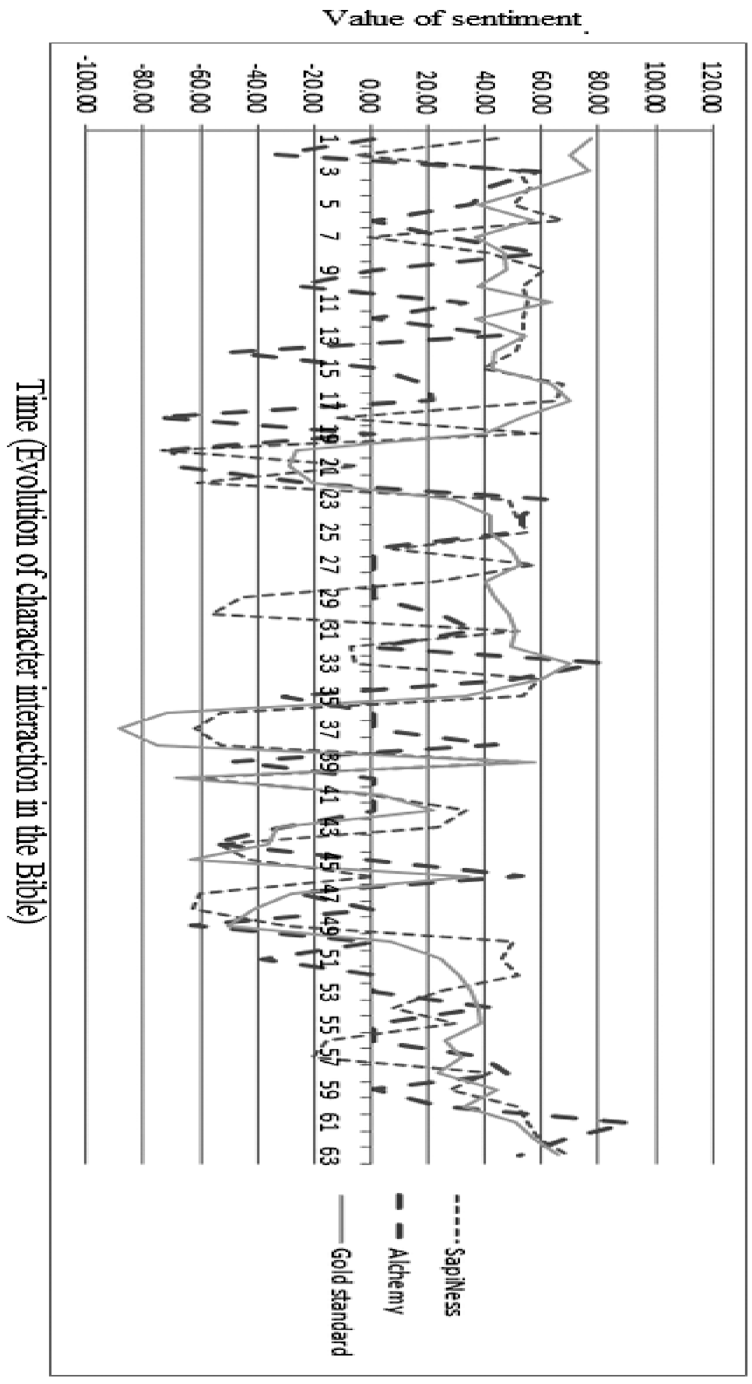


Figure 3: Comparison between AI and gold standard

The matches and discrepancies between our application and the commercial one is visualized in figure 2:

As the two results in figure 2 are the results of two automated non-human sentiment analyzers, we asked three independent person to provide their own opinions on what positive, negative and neutral connotation each analyzed sentence had.

The results of this manual analysis, overlapped with the previous two results, are visible in figure 3:

We chose as gold standard (GS: $g_i, i = 1, n$) the sentiment polarity sequence that was given by the three independent person ($g_i = (e1_i + e2_i + e3_i)/3, i = 1, n$). The means of deviation sequences "GS vs. Sapiness" ($|g_i - s_i|, i = 1, n$) and "GS vs. Alchemy" ($|g_i - a_i|, i = 1, n$) were 23.82 and 36.77, respectively. Applying a paired t-test we found that the Sapiness result was significantly better than the Alchemy's one ($p = 0.005 < 0.05$).

Analyzing the bible was our own idea, however after analysis we found that similar experiments have already been conducted, but no paper has been published on the subject.

5 Conclusions and future work

We took the idea of sentiment analysis, and implemented it using two different methods, while testing it against existing commercial products and research applications. We also conducted two experiments which mirrored practical applications and whose main building block was our sentiment analyzer. After examining the results of our benchmarks in subsection 4.2, the results of the experiment in subsection 4.3, we can conclude that we have significantly improved our sentiment analyzer application with every iteration, and we are currently at a point where our application produces results which are comparable to commercial applications and research products.

Future work on the project may include a variety of objectives, but the main goal is improving the accuracy of the analyzer. We may take multiple roads to approach this goal, such as trying to compile better training sets, try to tweak the configuration of the neural network, and even trying other types of neural networks which can be used for our purposes, such as Support Vector Machines and Naïve Bayes.

As for the experiment conducted in section 4.3, we may try to compile a training set from a literature piece of an author, then try to use the newly trained neural network on a different work of fiction from the same author, and test how much improvement, if any of significance, can be had by training the neural network to recognize connotations for a specific style of writing, instead of generalizing it.

Further experimentation can be conducted in order to compare the accuracy of different handling modes for negation, as listed in section 3.2.

We should also study the use of links between SentiWordNet[1] and wordnet synsets in other languages, as presented in paper [2], in order to automatically generate a corpus for any language, solving the issue presented in section 3.2 regarding words with multiple meanings.

We are planning to use the presented application (Sapiness – Sentiment Analyser) for detecting, modeling and characterizing the informal network of the Faculty of Technical and Human Sciences of Sapientia University (based on the day-to-day electronic interactions of its members.) After the informal social digraph has been established each arc will be characterized by the sentimental content of the corresponding messages. We hope that the formal management of our Faculty will take advantage of the expected results of this research.

References

[1] S. Baccianella, A. Esuli, F. Sebastiani, SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining, *LREC 2010, Seventh International Conference on Language Resources and Evaluation*, Valetta, Malta, May 17–23, 2010, pp. 2200–2204. ⇒188, 189, 198

[2] V. Basile, M. Nissim, Sentiment analysis on Italian tweets, *Proc. 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, June 14, 2013, Atlanta, Georgia, USA, pp. 100–107. ⇒198

[3] G. Berend, R. Farkas, Opinion mining in Hungarian based on textual and graphical clues, *Proc. 8th WSEAS International Conference on Simulation, Modelling and Optimization*, September 23–25, 2008, Santander, Spain. ⇒189

[4] C. Bosco, V. Patti, A. Bolioli, Developing corpora for sentiment analysis: The case of Irony and SentiTUT, *Intelligent Systems, IEEE* **28,** 2 (2013) 55–63. ⇒ 188

[5] M. Dadvar, C. Hauff, F. de Jong, Scope of negation detection in sentiment analysis, *DIR 2011: the eleventh Dutch-Belgian information retrieval workshop*, 2011, pp. 16–19. ⇒191

[6] H. Ghorbel, D. Jacot, Sentiment analysis of french movie reviews, *Advances in Distributed Agent-Based Retrieval Tools*, Vol. 361, 2011, pp 97–108. ⇒188

[7] L. A. Freitas, R. Vieira, Ontology based feature level opinion mining for portuguese reviews, *Proc. WWW '13 Companion Proceedings of the 22nd International Conference on World Wide Web*, Rio de Janeiro, May 13–17, 2013, pp. 367–370. ⇒188

[8] V. Hangya, R. Farkas, G. Berend, Entitásorientált véleménydetekció webes híranyagokból, In: Tanács, A., Varga V., Vincze V. (eds.) *XI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2015)*, Szeged, Jan. 14–15, 2015. Univ. Szeged, pp. 227–234. ⇒189

[9] A. Hogenboom, P. van Iterson, B. Heerschop, F. Frasincar, U. Kaymak, Determining Nnegation scope and strength in sentiment analysis, *2011 IEEE International Conference on Systems, Man and Cybernetics – SMC*, October 9–12, 2011, Hilton Anchorage, Anchorage, AK, USA, pp. 2589–2594. ⇒191

[10] B. Liu, Sentiment analysis and opinion mining, *Synthesis Lectures on Human Language Technologies* **5**, 1 (2012) 1–167 ⇒186, 188

[11] M. Miháltz, OpinHuBank: szabadon hozzáférhető annotált korpusz magyar nyelvű véleményelemzéshez. *IX. Magyar Számítógépes Nyelvészeti Konferencia*, 2013, pp. 343–345. ⇒189, 193

[12] S. Momtazi, Fine-grained German sentiment analysis on social media, *Proc. LREC'12 Eighth International Conference on Language Resources and Evaluation*, Istanbul, Turkey, May 21–27, 2012, pp. 1215–1220. ⇒188

[13] A. Reyes, Paolo Rosso, D. Buscaldi, From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Eng.* **74** (2013) 1–12. ⇒188

[14] K. Roberts, M. A. Roach, J. Johnson, J. Guthrie, S. M. Harabagiu, Empatweet: Annotating and detecting emotions on Twitter. *Proc. LREC'12 Eighth International Conference on Language Resources and Evaluation*, Istanbul, Turkey, May 21–27, 2012,pp. 3806–3813. ⇒189

[15] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, Ch. D. Manning, A. Y. Ng; Ch. Potts, Recursive deep models for semantic compositionality over a sentiment treebank *Proc. EMNLP 2014: Conf. on Empirical Methods in Natural Language Processing*, October 25–29, 2014, Doha, Qatar, vol. 1631, p. 1642. ⇒191, 193

[16] M. K. Szabó, V. Vincze, Egy magyar nyelvű szentimentkorpusz l etrehozásának tapasztalatai, *XI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2015)*, Szeged, Jan. 14–15, 2015. Univ. Szeged, pp. 219–226 ⇒189

[17] J. Turian, *Using AlchemyAPI for Enterprise-Grade Text Analysis.* Technical Report, AlchemyAPI, August 2013. ⇒193, 195

[18] J. Zsibrita, V. Vincze, R. Farkas, Magyarlanc: A Toolkit for Morphological and Dependency Parsing of Hungarian, *Proc. RANLP 2013*, September 7–17, Hissar, Bulgaria, pp. 763–771. ⇒190

Received: November 5, 2015 • Revised: December 30, 2015

Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 200–215

DOI: 10.1515/ausi-2015-0019

On the scores and degrees in hypertournaments

Shariefuddin PIRZADA
University of Kashmir
Srinagar, India
email:
`mailto:pirzadasd@kashmiruniversity.ac.in`

Rameez RAJA
University of Kashmir
Srinagar, India
email: `rameeznaqash@gmail.com`

Antal IVÁNYI
Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary
email: `tony@inf.elte.hu`

Abstract. A k-hypertournament $H = (V, A)$, where V is the vertex set and A is an arc set, is a complete k-hypergraph with each k-edge endowed with an orientation, that is, a linear arrangement of the vertices contained in the edge. In a k-hypertournament, the score s_i(losing score r_i) of a vertex is the number of edges containing v_i in which v_i is not the last element(in which v_i is the last element) and the total score of a vertex v_i is $t_i = s_i - r_i$. For $v \in V$ we denote $d_H^+ = \sum\limits_{a \in H} \rho(v, a)$ (or simply $d^+(v)$) the degree of a vertex where, $\rho(v, a)$ is $k - i$ if $v \in a \in A$ and v is the ith entry in a, otherwise zero. In this paper, we obtain necessary and sufficient conditions for a k-hypertournament to be degree regular. We use the inequalities of Holder and Chebyshev from mathematical analysis to study the score and degree structure of the k-hypertournaments.

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C30, 05C50
Key words and phrases: tournaments, hypertournaments, score, losing score, degree

1 Introduction

Hypertournaments which are the generalizations of tournaments, have been studied by number of authors like R. Assous [1], Barbut and Bialostocki [2], P. Frank [4] and Gutin and Yeo [5]. These authors raise the problem of extending the most important results on tournaments to hypertournaments. G. Zhou et al. [26] extended the concept of scores in tournaments to that of scores and losing scores in hypertournaments, and derived various results [6, 14, 15, 16, 17, 18, 19, 21, 10, 11, 12, 13, 22, 23, 25].

Given two non-negative integers n and k with $n \geq k > 1$, a k-hypertournament H on n vertices is a pair (V, A), where V is the set of vertices with $|V| = n$, and A is the set of k-tuples of vertices called arcs, such that for any k-subset S of V, A contains exactly one of the $k!$ k tuples whose entries belong to S.

Zhou et al. [26] extended the concept of scores in tournaments to that of scores and losing scores in hypertournaments, and derived a result analogous to Landau's theorem [9] on tournaments. The score $s(v_i)$ or s_i of a vertex v_i is the number of arcs containing v_i in which v_i is not the last element and the losing score $r(v_i)$ or r_i of a vertex v_i is the number of arcs containing v_i in which v_i is the last element. The score sequence (losing score sequence) is formed by listing the scores(losing scores) in non-decreasing order. A k-hypertournament is said to be regular if the scores of each vertex (equivalently the losing scores) are same.

Let H be a k-hypertournament and let $v \in V$ be any vertex and $a = (v_1, v_2, \ldots, v_n) \in A$ be an arc of H. We denote by $d_H^+ = \sum\limits_{a \in H} \rho(v, a)$ (or simply $d^+(v)$) the degree of a vertex $v \in V$ where,

$$\rho(v, a) = \begin{cases} k - i, & \mathit{if} \;\; v \in a \;\; \mathit{and} \;\; v \;\; \mathit{is} \;\; \mathit{the} \;\; i^{th} \;\; \mathit{entry} \;\; \mathit{of} \;\; a, \\ 0, & \mathit{if} \;\; v \notin a. \end{cases}$$

A hypertournament is said to be degree regular if all the vertices have the same degree. The degree sequence of a k-hypertournament in non-decreasing order is a sequence of non-negative integers $[d_1, d_2, \ldots, d_n]$, where each d_i is the degree of some vertex in V.

The following characterizations of losing score sequences of k-hypertournaments can be found in [26], and a new short proof is given by Pirzada et al. in [20].

Theorem 1 *Given two non-negative integers n and k with $n \geq k > 1$, a non-decreasing sequence $R = [r_1, r_2, \ldots, r_n]$ of non-negative integers is a losing score sequence of some k-hypertournament if and only if for each $1 \leq j \leq n$,*

$$\sum_{i=1}^{j} r_i \geq \binom{j}{k}, \tag{1}$$

with equality when $j = n$.

Theorem 2 *Given two integers n and k with $n \geq k > 1$, a non-decreasing sequence $S = [s_1, s_2, \ldots, s_n]$ of non-negative integers is a score sequence of some k-hypertournament if and only if for each $1 \leq j \leq n$,*

$$\sum_{i=1}^{j} s_i \geq j\binom{n-1}{k-1} + \binom{n-j}{k} - \binom{n}{k}, \tag{2}$$

with equality when $j = n$.

Koh and Ree [7] defined the k-hypertournament matrix $M = M(H)$ associated with a k-hypertournament $H = (V, A)$ as the incidence matrix $M = [m_{ij}]$ of size $n \times \binom{n}{k}$ of H, where for $1 \leq i \leq n$ and $1 \leq j \leq \binom{n}{k}$, m_{ij} is given by

$$m_{ij} = \begin{cases} 1, & \text{if } v_i \in e_j \text{ and } v_i \text{ is not the last element of } e_j, \\ -1, & \text{if } v_i \in e_j \text{ and } v_i \text{ is the last element of } e_j, \\ 0, & \text{if } v_i \notin e_j. \end{cases}$$

Since $s_i + r_i = \binom{n-1}{k-1}$ for each i, then clearly a given sequence $s_1 \geq s_2 \geq \ldots \geq s_n \geq 0$ is a score sequence of a k-hypertournament.

Theorem 3 *A non-increasing sequence of non-negative integers $s_1 \geq s_2 \geq \ldots \geq s_n \geq 0$ is a score sequence of a k-hypertournament H if and only if*

$$\sum_{i=1}^{l} s_i \leq l\binom{n-1}{k-1} - l\binom{l}{k},$$

for $l = \{1, 2, \ldots, n\}$ with equality when $l = n$.

Theorem 4 *Sequences* $0 \leq s_1 \leq s_2 \leq \ldots \leq s_n \leq$ *and* $r_1 \geq r_2 \geq \ldots \geq r_n \geq 0$ *are the score and losing score sequence of a* k*-hypertournament* H *if and only if they satisfy* $s_i + r_i = \binom{n-1}{k-1}$ *for all* $i = \{1, 2, \ldots, n\}$,

$$\sum_{i=1}^{l} s_i \geq l\binom{n-1}{k-1} + l\binom{n-l}{k} - \binom{n}{k},$$

and

$$\sum_{i=1}^{l} r_i \leq \binom{n}{k} - \binom{n-l}{k},$$

for $l = \{1, 2, \ldots, n\}$ *with equality when* $l = n$.

The following result [7] gives the condition for a sequence to be the total score sequence of a k-hypertournament matrix.

Theorem 5 *A non-increasing sequence of integers* $T = [t_1, t_2, \ldots, t_n]$ *is the total score sequence of a* k*-hypertournament matrix* M *on* n *vertices if and only if* t_i *has the same parity as that of* $\binom{n-1}{k-1}$, *for all* $i = \{1, 2, \ldots, n\}$ *and*

$$\sum_{i=1}^{l} t_i \leq l\binom{n-1}{k-1} - 2\binom{l}{k},$$

for $l = \{1, 2, \ldots, n\}$ *with equality when* $l = n$.

Khan, Pirzada and Kayibi [8] applied the inequalities from mathematical analysis like Holder's, Minkowski's and Mahler's inequalities to the powers of scores and losing scores of k-hypertournaments and obtained the following results. The result below [8] gives a lower bound on $\sum_{i=i}^{j} r_i^g$, where $1 < g < \infty$ is a real number.

Theorem 6 [8]. *Let* n *and* k *be two non-negative integers with* $n \geq k > 1$. *If* $[r_1, r_2, \ldots, r_n]$ *is a losing score sequence of a* k*-hypertournament, then for* $1 < g < \infty$

$$\sum_{i=i}^{j} r_i^g \geq \frac{j}{k^g}\binom{j-1}{k-1}^g,$$

where $1 \leq j < n$. *In particular, for* $j = n$

$$\sum_{i=i}^{n} r_i^g \geq \frac{n}{k^g}\binom{n-1}{k-1}^g, \tag{3}$$

with equality in (3) holds if and only if the hypertournament is regular.

The next result [8] gives an upper bound for the inner product of score and losing score vectors in R^n. The bound given in Theorem 7 is best possible in the sense that it is realized by regular hypertournaments. It should also be noted that Theorem 7 does not depend on the order of s_i and r_i, and holds for any arbitrary ordering of scores and losing scores.

Theorem 7 *Let* n *and* k *be two non-negative integers with* $n \geq k > 1$. *If* $S = [s_1, s_2, \ldots, s_n]$ *and* $R = [r_1, r_2, \ldots, r_n]$ *are respectively the score and losing score sequence of a* k*-hypertournament, then*

$$\langle S, R\rangle = \sum_{i=1}^{n} s_i r_i \leq \frac{k-1}{k}\binom{n}{k}\binom{n-1}{k-1},$$

with equality holds if and only if the hypertournament is regular.

For $k = 2$ the degree sequence is identical to the score sequence given in [9]. In [27] Zhou and Zhang conjectured that a nondecreasing sequence $D = [d_1, d_2, \ldots, d_n]$ of nonnegative integers is a degree sequence of some k-hypertournament under some conditions, and proved for the case $k = 3$.

The conjecture raised by Zhou and Zhang in [27] was settled by Chao and Zhou [24] and was obtained the following result.

Theorem 8 *Given two positive integers* n *and* k *with* $n > k > 1$, *a nondecreasing sequence* $[d_1, d_2, \ldots, d_n]$ *of nonnegative integers is a degree sequence of some* k*-hypertournament if and only if*

$$\sum_{i=1}^{r} d_i \geq \binom{r}{2}\binom{n-2}{k-2},$$

for all $1 \leq r \leq n$ *with equality for* $r = n$.

Let H be a an r-uniform hypergraph with $r \geq 2$ and let $\alpha(H)$ be the vertex independence number of H. In 2014 Chisthi, Zhou, Pirzada and Iványi [3] gave bounds for $\alpha(H)$ for different uniform hypergraphs.

2 On stronger bounds in hypertournaments

The following result is an equivalent form of Theorem 7 for scores and losing scores in a k-hypertournament. Here we give a different proof of this result.

Theorem 9 *Given two nonnegative integers* n *and* k *with* $n \geq k > 1$, *if* $S = [s_1, s_2, \ldots, s_n]$ *of nonnegative integers in nonincreasing order is a score sequence and* $R = [r_1, r_2, \ldots, r_n]$ *in non-decreasing order is the losing score sequence of some* k*-hypertournament, then*

$$\sum_{i=1}^{j} s_i r_i \leq \binom{n}{k} \left\{ \binom{n-1}{k-1} - \binom{n}{k} \frac{1}{n} \right\}, \tag{4}$$

with equality holds if and only if the hypertournament is regular.

Proof. Let $S = [s_1, s_2, \ldots, s_n]$ and $R = [r_1, r_2, \ldots, r_n]$ be respectively the score sequence and losing score sequence of a k-hypertournament H, with S being non-increasing and R being nondecreasing. Then as a consequence of Cauchy-Schwartz inequality, we have

$$\left(\frac{r_1 + r_2 + \ldots + r_n}{n} \right) \left(\frac{s_1 + s_2, \ldots + s_n}{n} \right) \geq \frac{r_1 s_1 + r_2 s_2 + \ldots + r_n s_n}{n},$$

or

$$\frac{1}{n} \sum_{i=1}^{n} s_i r_i \leq \frac{1}{n^2} \sum_{i=1}^{n} r_i \sum_{i=1}^{n} s_i,$$

or

$$\sum_{i=1}^{n} s_i r_i \leq \frac{1}{n} \sum_{i=1}^{n} r_i \sum_{i=1}^{n} s_i. \tag{5}$$

Now,

$$\sum_{i=1}^{n} (s_i + r_i) r_i = \sum_{i=1}^{n} s_i r_i + \sum_{i=1}^{n} r_i^2.$$

This gives,

$$\sum_{i=1}^{n} \binom{n-1}{k-1} r_i = \sum_{i=1}^{n} s_i r_i + \sum_{i=1}^{n} r_i^2,$$

(because $s_i + r_i = \binom{n-1}{k-1}$, $\quad 1 \leq i \leq n$).
So,

$$\binom{n-1}{k-1} \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} s_i r_i + \sum_{i=1}^{n} r_i^2,$$

or

$$\binom{n-1}{k-1} \binom{n}{k} = \sum_{i=1}^{n} s_i r_i + \sum_{i=1}^{n} r_i^2, \tag{6}$$

(by the equality in Theorem 1).

Further, by the Chebyshev's inequality, we have,

$$\sum_{i=1}^{n} s_i r_i \geq \frac{1}{n} \sum_{i=1}^{n} s_i \sum_{i=1}^{n} r_i.$$

Using this in (6), we get

$$\binom{n-1}{k-1}\binom{n}{k} \geq \frac{1}{n} \sum_{i=1}^{n} s_i \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} r_i^2. \tag{7}$$

Since the arithmetic mean of n non-negative real numbers never exceeds their root mean square, that is,

$$\sqrt{\frac{\sum_{i=1}^{n} r_i^2}{n}} \geq \frac{\sum_{i=1}^{n} r_i}{n},$$

with equality if and only if $r_1 = r_2 = \ldots = r_n$,
or

$$\sum_{i=1}^{n} r_i^2 \geq \frac{\left(\sum_{i=1}^{n} r_i\right)^2}{n}. \tag{8}$$

Using (8) in (7) we get

$$\binom{n-1}{k-1}\binom{n}{k} \geq \frac{1}{n} \sum_{i=1}^{n} s_i \sum_{i=1}^{n} r_i + \frac{\left(\sum_{i=1}^{n} r_i\right)^2}{n}.$$

Therefore,

$$\binom{n-1}{k-1}\binom{n}{k} \geq \frac{1}{n} \sum_{i=1}^{n} s_i \sum_{i=1}^{n} r_i + \binom{n}{k}^2 \frac{1}{n},$$

or

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^{n} s_i \sum_{i=1}^{n} r_i &\leq \binom{n-1}{k-1}\binom{n}{k} - \frac{1}{n}\binom{n}{k}^2 \\ &= \binom{n}{k}\left\{\binom{n-1}{k-1} - \binom{n}{k}\frac{1}{n}\right\}. \end{aligned}$$

Using this in (5), we get

$$\sum_{i=1}^{n} s_i r_i \leq \binom{n}{k}\left\{\binom{n-1}{k-1} - \binom{n}{k}\frac{1}{n}\right\}. \tag{9}$$

Now we show that the equality in (9) holds if and only if the hypertournament is regular, that is, if and only if $r_1 = r_2 = \ldots = r_n = r$.

Let $r_1 = r_2 = \ldots = r_n = r$ in (9).

Then equality holds if and only if

$$r\sum_{i=1}^{n} s_i = \binom{n}{k}\left\{\binom{n-1}{k-1} - \binom{n}{k}\frac{1}{n}\right\}.$$

That is, if and only if

$$\frac{1}{n}\binom{n}{k}\sum_{i=1}^{n} s_i = \binom{n}{k}\left\{\binom{n-1}{k-1} - \binom{n}{k}\frac{1}{n}\right\},$$

(because by the equality in Theorem 1).

That is, if and only if

$$\frac{1}{n}\binom{n}{k}\left\{n\binom{n-1}{k-1} + 0 - \binom{n}{k}\right\} = \binom{n}{k}\left\{\binom{n-1}{k-1} - \binom{n}{k}\frac{1}{n}\right\},$$

(because by the equality in Theorem 2).

Therefore,

$$\frac{1}{n}\left\{n\binom{n-1}{k-1} - \binom{n}{k}\right\} = \binom{n-1}{k-1} - \frac{1}{n}\binom{n}{k},$$

or

$$\binom{n-1}{k-1} - \frac{1}{n}\binom{n}{k} = \binom{n-1}{k-1} - \frac{1}{n}\binom{n}{k},$$

which is true. Hence the equality in (9) holds, if and only if the hypertournament is regular. □

The next result gives stronger bound for the total scores in k-hypertournaments.

Theorem 10 *If $S = [s_1, s_2, \ldots, s_n]$ is a score sequence in non-decreasing order, $R = [r_1, r_2, \ldots, r_n]$, is a losing score sequence in non-increasing order and $T = [t_1, t_2, \ldots, t_n]$ is the total score sequence in non-increasing order of a k-hypertournament H, then for $1 < p < \infty$*

$$\sum_{i=1}^{l} t_i^p \geq l\left\{\binom{n-1}{k-1} + \frac{2}{l}\binom{n-l}{k} - \frac{2}{l}\binom{n}{k}\right\}^p,$$

with equality holds if and only if $t_1 = t_2 = \ldots = t_n$ and $l = n$.

Proof. Since the total score is $t_i = s_i - r_i$, then

$$\sum_{i=1}^{l} t_i = \sum_{i=1}^{l} s_i - \sum_{i=1}^{l} r_i.$$

Therefore, by using Theorem 4 we obtain,

$$\sum_{i=1}^{l} t_i \geq l\binom{n-1}{k-1} + \binom{n-l}{k} - \binom{n}{k} - \binom{n}{k} + \binom{n-l}{k},$$

or

$$\sum_{i=1}^{l} t_i \geq l\binom{n-1}{k-1} - 2\binom{n}{k} + 2\binom{n-l}{k}. \tag{10}$$

But by the Holder's inequality with $\frac{1}{p} + \frac{1}{q} = 1$ we have,

$$\sum_{i=1}^{l} t_i \leq \left(\sum_{i=1}^{l} t_i^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{l} t_i^q\right)^{\frac{1}{q}}.$$

Using in (10), we get

$$l^{\frac{1}{q}} \left(\sum_{i=1}^{l} t_i^p\right)^{\frac{1}{p}} \geq l\binom{n-1}{k-1} + 2\binom{n-l}{k} - 2\binom{n}{k},$$

or

$$\left(\sum_{i=1}^{l} t_i^p\right)^{\frac{1}{p}} \geq l^{-\frac{1}{q}} \left(l\binom{n-1}{k-1} + 2\binom{n-l}{k} - 2\binom{n}{k}\right).$$

This implies

$$\begin{aligned}
\left(\sum_{i=1}^{l} t_i^p\right)^{\frac{1}{p}} &\geq l^{-\frac{1}{q}} l\binom{n-1}{k-1} + \frac{2}{l} l^{-\frac{1}{q}} l\binom{n-l}{k} - \frac{2}{l} l^{-\frac{1}{q}} l\binom{n}{k} \\
&\geq l^{1-\frac{1}{q}}\binom{n-1}{k-1} + \frac{2}{l} l^{1-\frac{1}{q}}\binom{n-l}{k} - \frac{2}{l} l^{1-\frac{1}{q}}\binom{n}{k} \\
&= l^{1-\frac{1}{q}} \left(\binom{n-1}{k-1} + \frac{2}{l}\binom{n-l}{k} - \frac{2}{l}\binom{n}{k}\right) \\
&= l^{\frac{1}{p}} \left(\binom{n-1}{k-1} + \frac{2}{l}\binom{n-l}{k} - \frac{2}{l}\binom{n}{k}\right),
\end{aligned}$$

(because $\frac{1}{p} + \frac{1}{q} = 1$ or $\frac{1}{p} = 1 - \frac{1}{q}$),
which gives,

$$\sum_{i=1}^{l} t_i^p \geq l\left(\binom{n-1}{k-1} + \frac{2}{l}\binom{n-l}{k} - \frac{2}{l}\binom{n}{k}\right)^p. \tag{11}$$

Now, we show that the equality in (11) holds, if and only if $t_1 = t_2 = \ldots = t_n = t$ and $l = n$. That is, the equality in (11) holds if and only if

$$nt^p = n\left(\binom{n-1}{k-1} + \frac{2}{n}\binom{n-n}{k} - \frac{2}{n}\binom{n}{k}\right)^p.$$

That is, if and only if

$$t^p = \left(\binom{n-1}{k-1} + \frac{2}{n}(0) - \frac{2}{n}\binom{n}{k}\right)^p,$$

or

$$t = \binom{n-1}{k-1} - \frac{2}{n}\binom{n}{k},$$

or

$$t = \frac{k}{n}\binom{n}{k} - \frac{2}{n}\binom{n}{k} = \frac{1}{n}\binom{n}{k}(k-2), \tag{12}$$

which is the total score of a vertex in a regular k-hypertournament, because of the following fact.

By Theorem 5, we have for $l = n$, and $t_1 = t_2 = \ldots = t_n = t$

$$\sum_{i=1}^{n} t = n\binom{n-1}{k-1} - 2\binom{n}{k}.$$

This implies,

$$nt = n\binom{n-1}{k-1} - 2\binom{n}{k},$$

or

$$t = \binom{n-1}{k-1} - \frac{2}{n}\binom{n}{k}.$$

Further, we can write

$$t = \frac{k}{n}\binom{n-1}{k-1} - \frac{2}{n}\binom{n}{k} = \frac{1}{n}\binom{n}{k}(k-2), \tag{13}$$

which shows that, (12) and (13) are same. □

3 Degrees in hypertournaments

It is evident from Theorem 8 that inequalities on degrees play important role in the study of hypertournaments. We shall use the classical inequalities to provide more insight into the behavior of degrees in hypertournaments and hence the structure of hypertournaments. We also discuss the case of equality in detail for the inequalities derived and prove the equality holds if and only if the hypertournament is degree regular. We also obtain the necessary and sufficient conditions for the existence of a degree regular hypertournament.

Theorem 11 *Let* n *and* k *be two positive integers with* $n > k > 1$*. If* $D = [d_1, d_2, \dots, d_n]$ *is a degree sequence of some* k*-hypertournament, then for a real number* p *with* $1 < p < \infty$

$$\sum_{i=1}^{r} d_i^p \geq \frac{r}{2^p}(r-1)^p \binom{n-2}{k-2}^p, \tag{14}$$

where $1 \leq r \leq n$*. In particular, for* $r = n$

$$\sum_{i=1}^{n} d_i^p \geq \frac{n}{2^p}(n-1)^p \binom{n-2}{k-2}^p, \tag{15}$$

with equality in (15) holds if and only if the hypertournament is degree regular.

Proof. By Theorem 8, we have

$$\sum_{i=1}^{r} d_i \geq \binom{r}{2}\binom{n-2}{k-2},$$

or

$$\binom{r}{2}\binom{n-2}{k-2} \leq \sum_{i=1}^{r} d_i.$$

But,

$$\sum_{i=1}^{r} d_i = \sum_{i=1}^{r} d_i.1 \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}}(\sum_{i=1}^{r} 1^q)^{\frac{1}{q}},$$

(because by Holder's inequality with, $\frac{1}{p} + \frac{1}{q} = 1$).
Hence,

$$\binom{r}{2}\binom{n-2}{k-2} \leq \left(\sum_{i=1}^{r} d_i^p)\right)^{\frac{1}{p}} \left(\sum_{i=1}^{r} 1^q\right)^{\frac{1}{q}},$$

(for, $1 \leq r \leq n$ and $\frac{1}{p} + \frac{1}{q} = 1$).
That is,

$$\binom{r}{2}\binom{n-2}{k-2} \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}} r^{\frac{1}{q}},$$

or

$$r^{-\frac{1}{q}}\binom{r}{2}\binom{n-2}{k-2} \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}},$$

or

$$\frac{r^{-\frac{1}{q}} r(r-1)(r-2)!}{2(r-2)!\binom{n-2}{k-2}} \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}},$$

or

$$r^{1-\frac{1}{q}}\frac{(r-1)}{2}\binom{n-2}{k-2} \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}}.$$

This gives,

$$r^{\frac{1}{p}}\frac{(r-1)}{2}\binom{n-2}{k-2} \leq (\sum_{i=1}^{r} d_i^p)^{\frac{1}{p}},$$

(because $\frac{1}{p} + \frac{1}{q} = 1$ or $\frac{1}{p} = 1 - \frac{1}{q}$).
Hence,

$$\sum_{i=1}^{r} d_i^p \geq \frac{r(r-1)^p}{2^p}\binom{n-2}{k-2}^p. \tag{16}$$

For $r = n$, we have by the equality in Theorem 8

$$\sum_{i=1}^{n} d_i = \binom{n}{2}\binom{n-2}{k-2}.$$

So, inequality (16) now becomes

$$\sum_{i=1}^{n} d_i^p \geq \frac{n}{2^p}(n-1)^p\binom{n-2}{k-2}^p. \tag{17}$$

Further, we show that the equality in (17) holds if and only if $d_1 = d_2 = ... = d_n = d$, that is, if and only if the hypertournament is degree regular.
Suppose $d_1 = d_2 = ... = d_n = d$ in (17). Then equality holds if and only if

$$\sum_{i=1}^{n} d_i^p = \frac{n}{2^p}(n-1)^p\binom{n-2}{k-2}^p.$$

That is, if and only if

$$nd^p = \frac{n}{2^p}(n-1)^p\binom{n-2}{k-2}^p,$$

or

$$d^p = \frac{(n-1)^p}{2^p}\binom{n-2}{k-2}^p.$$

That is, if and only if

$$d = \frac{(n-1)}{2}\binom{n-2}{k-2}, \tag{18}$$

which clearly is the degree of a vertex in a regular k-hypertournament, verified as follows. We know by the equality in Theorem 8

$$\sum_{i=1}^{n} d_i = \binom{n}{2}\binom{n-2}{k-2}.$$

Since $d_1 = d_2 = \ldots = d_n = d$.

$$nd = \binom{n}{2}\binom{n-2}{k-2},$$

which gives,

$$nd = \frac{n(n-1)!}{2(n-2)!}\binom{n-2}{k-2},$$

or

$$d = \frac{(n-1)}{2}\binom{n-2}{k-2}. \tag{19}$$

Clearly (18) is same as (19). □

The following result gives the conditions for the existence of a degree regular k-hypertournament on n vertices.

Theorem 12 *Let n and k be two positive integers. For $n > 2$ and $n > k > 1$, there exists a degree regular k-hypertournament on n vertices if and only if n divides $\binom{k}{2}\binom{n}{k}$.*

Proof. Suppose there exists a degree regular k-hypertournament with its degree sequence $[d_1, d_2, \cdots, d_n]$. Then by the inequality (16), we have

$$\sum_{i=1}^{n} d^2 = \frac{n}{2^2}(n-1)^2\binom{n-2}{k-2}^2,$$

(case of equality with $p = 2$, where $d_1 = d_2 = \ldots = d_n = d$).
Thus,

$$nd^2 = \frac{n}{2^2}(n-1)^2\binom{n-2}{k-2}^2,$$

or

$$d^2 = \frac{(n-1)^2}{2^2}\binom{n-2}{k-2}^2.$$

This gives,

$$d = \frac{(n-1)}{2}\binom{n-2}{k-2},$$

(because degree cannot be negative).
Now,

$$2d = (n-1)\binom{n-2}{k-2},$$

or

$$\frac{2d}{k(k-1)} = \frac{(n-1)(n-2)!}{k(k-1)(k-2)!(n-k)!},$$

or

$$2nd = \frac{k(k-1)n!}{k!(n-k)!},$$

or

$$nd = \frac{k(k-1)}{2}\binom{n}{2}.$$

Conversely, suppose that n divides $\binom{k}{2}\binom{n}{k}$.
Set for each $1 \le i \le n$,

$$d_i = \frac{1}{n}\binom{k}{2}\binom{n}{k} = \frac{k(k-1)}{2n}\binom{n}{k}.$$

Then,

$$\begin{aligned} d_i &= \frac{k(k-1)}{2n}\frac{n!}{k!(n-k)!} \\ &= \frac{k(k-1)n(n-1)(n-2)!}{2nk(k-1)(k-2)!(n-k)!} \\ &= \frac{(n-1)}{2}\binom{n-2}{k-2}. \end{aligned}$$

Therefore,

$$\sum_{i=1}^{r} d_i = \sum_{i=1}^{r} \left\{ \frac{n-1}{2} \binom{n-2}{k-2} \right\},$$

which implies, for $1 \le r \le n$

$$\begin{aligned}\sum_{i=1}^{r} d_i &= \frac{r(n-1)}{2} \binom{n-2}{k-2} \\ &\ge \frac{r(r-1)}{2} \binom{n-2}{k-2} \\ &= \frac{r(r-1)(r-2)!}{2(r-2)!} \binom{n-2}{k-2},\end{aligned}$$

(because $n \ge r$ implies $(n-1) \ge (r-1)$).
Hence,

$$\sum_{i=1}^{r} d_i \ge \binom{r}{2} \binom{n-2}{k-2},$$

with equality when $r = n$.

Thus by Theorem 8, $D = [d_1, d_2, \ldots, d_n]$ is the degree sequence of a degree regular k-hypertournament. □

References

[1] R. Assous, Enchainabilite et seuil de monomorphie des tournois n-aires, *Discrete Math.* **62** (1986), 119–125. ⇒201

[2] E. Barbut, A. Bialostocki, A generalization of rotational tournaments, *Discrete Math.* **76** (1989), 81–87. ⇒201

[3] T. A. Chisthi, G. Zhou, S. Pirzada, A. Iványi, On independence numbers of uniform hypergraphs, *Acta Univ. Sapientiae, Informatica* **6,** 1 (2014) 132–158. ⇒204

[4] P. Frankl, What must be contained in every oriented k-uniform hypergraph, *Discrete Math.* **62** (1986), 311–313. ⇒201

[5] G. Gutin, A. Yeo, Hamiltonian paths and cycles in hypertournaments, J. *Graph Theory* **25** (1997), 277–286. ⇒201

[6] K. K. Kayibi, M. A. Khan, S. Pirzada, Uniform sampling of k-hypertournaments, *Linear and Multilinear Algebra* **61,** 1 (2013), 123–138. ⇒201

[7] Y. Koh, S. Ree, On k-hypertournament matrices, *Lin. Alg. Appl.* **373** (2002) 183–195. ⇒202, 203

[8] M. A. Khan, S. Pirzada, K. K. Kayibii, Scores, Inequalities and regular hypertournaments, J. Math. Inequal. Appl. **15,** 2 (2012) 343–351. ⇒203, 204

[9] H. G. Landau, On dominance relations and the structure of animal societies, III, The condition for a score structure, *Bull. Math. Biophys.* **15** (1953) 143–148. ⇒201, 204

[10] S. Pirzada, On scores in multipartite hypertournaments, *Eurasian Math. Journal,* **2,** 1 (2011) 112–119. ⇒201

[11] S. Pirzada, *An Introduction to Graph Theory*, Universities Press, Orient Blackswan, 2012. ⇒201

[12] S. Pirzada, Score lists in bipartite multi hypertournaments, *Math. Vesnik* **64,** 4 (2012) 286–296. ⇒201

[13] S. Pirzada, hypertournaments-Scores, losing scores, total scores and degrees, *J. Comp. Math. Comb. Comp.* **84** (2013) 95–108. ⇒201

[14] S. Pirzada, T. A. Naikoo, Score sets in tournaments, *Vietnam J. Math.* **34,** 2 (2006) 157–161. ⇒201

[15] S. Pirzada, U. Samee, Mark sequences in digraphs, *Seminaire Lotharingien de Combinatoire* **55** (2006) B55c. ⇒201

[16] S. Pirzada, T. A. Naikoo, N. A. Shah, Score sequences in oriented graphs, *J. Appl. Math. Comp.* **23,** 1–2 (2007) 257–268. ⇒201

[17] S. Pirzada, G. Zhou, Score lists in (h,k)-bipartite hypertournaments, *Appl. Math. J. Chinese Universities, Series B* **22,** 4 (2007) 485–489. ⇒201

[18] S. Pirzada,, T. A. Naikoo, G. Zhou, Score lists in tripartite hypertournaments, *Graphs and Combinatorics* **23,** 4 (2007) 445–454. ⇒201

[19] S. Pirzada, T. A. Naikoo, Score sets in oriented graphs, *Appl. Analysis and Discrete Math.* **2,** 1 (2008) 107–113. ⇒201

[20] S. Pirzada, G. Zhou, On k-hypertournament losing scores, *Acta Univ. Sapientiae, Informatica* **2,** 1 (2010) 5–9. ⇒201

[21] S. Pirzada, G. Zhou, A. Iványi, Score lists in multipartite hypertournaments, Acta Univ. Sapientiae, Informatica **2,** 2 (2010) 184–193. ⇒201

[22] S. Pirzada, G. Zhou, Degree lists in k-bipartite hypertournaments, *TWMS J. Pure and Appl. Math.* **5,** 2 (2014), 89–116. ⇒201

[23] S. Pirzada, M. A. Khan, G. Zhou, K. K. Kayibi, On scores, losing scores and total scores in hypertournaments, *Electronic J. Graph Theory Appl.* **3,** 1 (2015) 8–21. ⇒201

[24] C. Wang, G. Zhou, Note on the degree sequences of k-hypertournaments, *Discrete Math.* **308,** 11 (2008) 2292–2296. ⇒204

[25] G. Zhou, S. Pirzada, Degree sequences in oriented k-hypergraphs, *J. Appl. Math., Comp.* **27,** 1-2 (2008) 149–158. ⇒201

[26] G. Zhou, Y. Tianxing, Z. Kemin, On score sequences of k-hypertournament, *Europ. J. Comb.* **21** (2000), 993–1000. ⇒201

[27] G. Zhou, K. Zhang, On the degree sequences of k-hypertournaments, *Chinese Annals of Math., Series A* **22,** 1 (2001) 115–120. ⇒204

Received: November 4, 2015 • Revised: December 22, 2015

DE GRUYTER OPEN Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 216–238

DOI: 10.1515/ausi-2015-0020

Chaotic behavior of the lattice Yang-Mills on CUDA

Richárd FORSTER
Eötvös University
Faculty of Informatics
email: `forceuse@inf.elte.hu`

Ágnes FÜLÖP
Eötvös University
Faculty of Informatics
email: `fulop@caesar.elte.hu`

Abstract.
The Yang-Mills fields plays important role in the strong interaction, which describes the quark gluon plasma. The non-Abelian gauge theory provides the theoretical background understanding of this topic.The real time evolution of the classical fields is derived by the Hamiltonian for SU(2) gauge field tensor. The microcanonical equations of motion is solved on 3 dimensional lattice and chaotic dynamics was searched by the monodromy matrix. The entropy-energy relation was presented by Kolmogorov-Sinai entropy. We used block Hessenberg reduction to compute the eigenvalues of the current matrix. While the purely CPU based algorithm can handle effectively only a small amount of values, the GPUs provide enough performance to give more computing power to solve the problem.

1 Introduction

In particle physics there are more fundamental questions which demand the GPU, both of theoretical and experimental point of view.

Computing Classification System 1998: I.1.4
Mathematics Subject Classification 2010: 81T25
Key words and phrases: lattice gauge field, chaos in Hamiltoinan systems, parallel computing, GPU, CUDA

In the CERN NA61 collaboration one of most important research field is the quark-gluon examination. The aspects of theoretical physics includes the next current topics in the lattice field theory using GPU: strongly interacting Higgs sector, QCD hadron spectrum (eigenvalue distribution of he overlap Dirac operator).

We present a parallel algorithm, which enables to study the chaotic behaviour as Lyapunov spectrum of SU(2) Yang-Mills fields and the entropy-energy relation utilizing the Kolmogorov-Sinai entropy. We uses this method for large number of element of matrices to apply the CUDA platform particularly the eigenvalue of the monodromy matrix, which is an $\mathsf{f x} \times \mathsf{f}$ sparse matrix ($\mathsf{f} = 24\mathsf{N}$).

The first step the non-Abelian gauge fields equation of motions is written by the lattice Hamiltonian SU(2) [2]. This system was solved by lattice process developed on the GPU [5]. The algorithm satisfies the constraint of the total energy and the unitarity, orthogonality of the suitable link variable on the 3 dimensional space.

In the next step we use the block Hessenberg reduction [11] to compute the required eigenvalues to determine the chaotic behaviour [8]. As it is described in [10] we are working with a hybrid system, that utilizes both the CPU and GPU for the most optimal performance. Thanks to this system it is possible to reach 2-3 times higher performance compared to the simple CPU based implementation of the same block Hessenberg reduction.

In the section 2 we introduce the basic concept of the non-Abelian gauge field. We describe the lattice regularization of Yang-Mills fields and SU(2) Hamiltonian to achieve the equations of motion in the section 3. We consider the chaotic description of the dynamics in the section 4, which contains chaos in the Hamiltonian systems and the lattice monodromy matrix method. The Hessenberg method is introduced in the section 5 to determine the eigenvalues of the monodromy matrix. The parallel hyprid Hessenberg algorithm is investigated in the section 6. The eigenvalue spectrum is determined by this parallel method and the numerical result is summarized in this section.

2 Gauge fields

The non-Abelian gauge[13] field plays important role in the theoretical particle physics. This theory based on principles of gauge invariance, which is derived from the Abelian gauge field to consider the principle of invariance under local gauge transformation.

In the electrodynamics field of charge e undergoes the local gauge transformation but derivative of this field does not transform like as the field itself, therefore we must introduce a field $A_\mu(x)$ ($\mu = 0, 1, 2, 3$) with gauge transformation property $A_\mu(x) \to A_\mu(x) + \partial_\mu \Lambda(x)$, with arbitrary $\Lambda(x)$ then we use it to construct a gauge invariant derivative of the original field of e, which transform just like this field. The gauge-invariant Lagrange can be constructed by these quantities. A dynamics for the gauge field is introduced by means of the Yang-Mills action:

$$S_{YM} = \frac{1}{4} \int d^4x F^a_{\mu\nu} F^a_{\mu\nu}, \tag{1}$$

where the $F^a_{\mu\nu}$ form is a component of an antisymmetric gauge field tensor in Minkowski space:

$$F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f^{abc} A^b_\mu A^c_\nu, \tag{2}$$

where $\mu\nu = 0, 1, 2, 3$ are space-time coordinates, the symmetry generators are labeled by $a, b, c = 1, 2, 3$ and g is the bare gauge coupling constant and f^{abc} are the structure constants of the continuous Lie group. The generators of this group fulfill the following relationship $[T^b, T^c] = i f^{bcd} T^d$. The equation of motion can be expressed by covariant derivative in the adjoint representation:

$$\partial^\mu F^a_{\mu\nu} + g f^{abc} A^{b\mu} F^c_{\mu\nu} = 0. \tag{3}$$

The Yang-Mills action contains cubic and quartic self-interaction terms. The original article[14] was published by Yang and Mills in 1954.

3 Lattice Yang-Mills fields

We will describe the lattice regularization on the Euclidean continuum to the hyper-cubic lattice [9].

The shortest non-zero distance on a hyper-cubic lattice is the lattice spacing a.

These are group elements which are related to the Yang-Mills potential A^c_i:

$$U_{x,i} = \exp(a A^c_i(x) T^c), \quad \text{where} \quad T^c \quad \text{is a group generator.} \tag{4}$$

For SU(2) symmetry group these links are given by the Pauli matrices τ, where $T^c = -(ig/2)\tau^c$. The indices x, i denote the link of the lattice which starts at the 3 dimensional position x and pointing into the nearest neighbour in direction i, $x+i$. We consider the collection of all link variables as the lattice gauge field.

3.1 Wilson action

We consider the construction of a gauge invariant action for the gauge field.

The non-Abelian gauge field strength can be expressed by the oriented plaquette i.e. product of four links on an elementary box with corners $(x, x+i, x+i+j, x+j)$:

$$U_{x,ij} = U_{x,i}U_{x+i,j}U_{x+i+j,-i}U_{x+j,-j}, \tag{5}$$

where $U_{x,-i} = U^{\dagger}_{x-i,i}$ and we will use this notation $U_p \equiv U_{x,ij}$.

The Wilson action is defined for pure lattice gauge theory [4]

$$S[U] = \sum_p S_p(U_p) \tag{6}$$

with the plaquette term:

$$S_p(U) = \beta(1 - \frac{1}{N}\mathrm{ReTr}U), \tag{7}$$

for SU(2) and β is a constant. Here the sum over all plaquettes p is meant to include every plaquette only with one orientation.

The Wilson action is gauge invariant since $\mathrm{Tr}U'_p = \mathrm{Tr}U_p$ and it is real and positive.

We consider the question, in which sense Wilson action for SU(2) is related to the Yang-Mills action for gauge fields on the continuum. Because $A_\mu(x)$ a Lie algebra values vector field. The expression (4) is extended by a, then the Wilson action is the following:

$$S = -\frac{\beta}{4N}\sum_x a^4 \mathrm{Tr}F_{\mu\nu}(x)F^{\mu\nu}(x) + O(a^5). \tag{8}$$

Thus the leading term for small 'a' coincidences with the Yang-Mills action if we set $\beta = \frac{2N}{g^2}$, where g identifies the bare coupling constant of the lattice theory.

The coupling on space-like and time-like plaquettes are no longer equal in the action:

$$S = \frac{2}{g^2}\sum_{p_t}(N - \mathrm{tr}(U_{p_t})) - \frac{2}{g^2}\sum_{p_s}(N - \mathrm{tr}(U_{ps})). \tag{9}$$

The time like plaquette is denoted by U_{p_t} and space like U_{p_s}.

Consider the path is a closed contour i. e. Wilson loop, it is invariant under gauge changes and independent of the starting point. The product of such group elements along the closed line is a gauge covariant quantity, the trace over such products are invariant. Because the U_{p_t} can be series expansion by a_t:

$$U_{p_t} = U(t)U^\dagger(t+a_t) = UU^\dagger + a_t U\dot{U}^\dagger + \frac{a_t^2}{2}U\ddot{U}^\dagger + \dots \tag{10}$$

$N - \mathrm{tr}(U_{p_t}) = -\frac{a_t^2}{2}\mathrm{tr}(U\ddot{U}^\dagger)$ up to $O(a_t^3)$ correction, where $UU^\dagger = 1$.

Therefore the homogenous non-Abelian gauge action:

$$\Delta S_H = \frac{2}{g^2}\left(\frac{a_t^2}{2}\sum_i \mathrm{tr}(\dot{U}_i\dot{U}_i^\dagger) - \sum_{ij}(N - \mathrm{tr}(U_{ij}))\right). \tag{11}$$

The Scaled Hamiltonian was derived in the next form:

$$a_t H = \frac{2}{g^2}\left(\frac{a_t^2}{2}\sum_i \mathrm{tr}(\dot{U}_i\dot{U}_i^\dagger) + \sum_{ij}(N - \mathrm{tr}(U_{ij}))\right). \tag{12}$$

The indices x, i denotes the link of the lattice starting at the 3 dimensional position x and pointing into the i-th direction.

3.2 SU(2) Hamiltonian

We study the real time classical evolution of the next Hamiltonian for SU(2) [3] [2]:

$$H = \sum_{x,i}\left(\frac{1}{2}\langle \dot{U}_{x,i}, \dot{U}_{x,i}\rangle + \left(1 - \frac{1}{4}\langle U_{x,i}, V_{x,i}\rangle\right)\right). \tag{13}$$

The complement variable $V_{x,l}(U)$ is constructed from triple product of links, which is complete to considere every link x, i to an elementary plaquette:

$$V_{x,l} = \frac{1}{4}\sum_{\left(\substack{(l,s):\{(i,j),(k,j),\\(-i,j),(-k,j)\}}\right)} U_{x+l,s}U^\dagger_{x+l+s,-l}U^\dagger_{x+l,-l}, \quad \text{where} \tag{14}$$

i, j, k note the unit vectors of three dimensional lattice. The canonical variable assigns by $P_{x,i} = \dot{U}_{x,i}$. We will denote the single link $U_{x,i}$ with U. Quaternion representation (for one link):

$$U = u_0 + i\tau^a u^a \qquad U = \begin{pmatrix} u_0 + iu_3, & iu_1 + u_2 \\ iu_1 - u_2, & u_0 - iu_3 \end{pmatrix}, \tag{15}$$

where, τ^a is the Pauli matrix. The lattice equation of motion is derived by canonical variable from this Hamiltonian.

3.3 Lattice equations of motion

The Hamiltonian equation of motion is solved with dt discrete time steps. This algorithm satisfies the Gauss law and the constraint of total energy[1]. We will denote single link $U_{x,i}$ in time t with U_t.

$$\begin{aligned} U_{t+1} - U_{t-1} &= 2\Delta t(P_t - \varepsilon U_t) \\ P_{t+1} - P_{t-1} &= 2\Delta t(V(U_t) - \mu U_t + \varepsilon P_t), \quad \text{where} \end{aligned} \tag{16}$$

$$\varepsilon = \frac{\langle U_t, P_t \rangle}{\langle U_t, U_t \rangle}, \quad \mu = \frac{\langle V(U_t), U_t \rangle + \langle P_t, P_t \rangle}{\langle U_t, U_t \rangle} \tag{17}$$

The ε, μ means the Lagrange multipliers and the symmetry SU(N) is fulfilled by the next expressions: $\langle U_t, U_t \rangle = 1$ (unitarity) and $\langle U_t, P_t \rangle = 0$ (orthogonality).

3.3.1 Implicit-Explicit-Endpoint algorithm

We apply these notions

$$P' = P_{t+1} \qquad P = P_t.$$

The Implicit-Explicit-Endpoint recursion algorithm:

$$P' = P + (V - \mu U + \varepsilon P') \tag{18}$$
$$U' = U + (P' - \varepsilon U), \tag{19}$$

where μ, ε are the Lagrange multipliers.

In the next section we study the nonlinearity of the Yang-Mills fields, which is described by the chaotic theory[12]. Instead of the classical rescaling solution we apply the monodromy matrix method, which can describe the gauge field evolution, in this case the short and long time behaviour. The question of ergodization is addressed via the Kolmogorov-Sinai entropy.

4 Measurement of chaos

We consider the description of dynamical systems. One of these is the Poencare map.

$$x_{i+1} = P(x_i) \quad i = 0, 1, 2 \ldots, \tag{20}$$

where we assign a hyper-surface $n-1$ dimension in the phase space n dimension and the trayectories are cutting it. The successive points of intersection can be given by this expression (20). Lyapunov exponent is defined by the next form:

$$\frac{d}{dt} x_i = F_i(x_1 \ldots x_n) \quad i = 1 \ldots N. \tag{21}$$

Let us introduce the quantity $\delta x_i(t)$:

$$\delta x_i(t) = x_i - \tilde{x_i}, \tag{22}$$

where $\delta x_i(t)$ means the distance between the two paths. The evolution of this notion is the following:

$$\frac{d}{dt}(\delta x_i) = \sum_{i=1}^{N} \delta x_k(t) \left(\frac{\partial F_i}{\partial x_i} \right)_{x_i = \tilde{x}_k(t)}. \tag{23}$$

The value of the distance $\delta x_i(t)$ is calculated by this expression:

$$D(t) = \left(\sum_{i=1}^{N} \delta(x_i(t))^2 \right)^{\frac{1}{2}}. \tag{24}$$

This quantity indicates the changing of the track distance $\delta\tilde{x}_i(t)$, where the paths were near at the beginning ($t = 0$). The maximal Lyapunov exponent follows:

$$L = \lim_{t \to \infty} \lim_{d(0) \to 0} \frac{1}{t} \ln \frac{D(t)}{D(0)}. \tag{25}$$

If $L > 0$, then the motion becomes chaoticity. We mention the rescaling method [7] briefly. Let us suppose there is a point q(0) in the phase space and the vectors $\nu_i, i = 1 \ldots \nu_K$ in the tangent space $T_{q(0)}$, we solve the equations of motion in phase space. We obtain q(t) and $\nu_i \in T_{q(t)}$ under the parallel evolution of the paths for a small perturbation of the initial condition in the tangent space. The Gram-Schmidt ortogonalisation is applied to determinate the tangent vector ν_i on the interval $k\tau$. The scaling vectors s_i are obtained by this procedure to calculate the Lyapunov exponents:

$$L_i = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{\ln s_i^k}{\tau}, \tag{26}$$

where n is the number of iterations. One condition is to determine the tangent space at different points of phase space in this procedure. It is easy in the Euclidean space, but more difficult on the lattice gauge theory, because we need to perform the rescaling frequently.

The monodromy matrix method follows only one gauge field path for a long time evolution. This matrix is a linear stability matrix along a trayectory, which is solved by classical equation of motion and it provides the Lyapunov spectrum in the time evolution of field configuration on lattice.

4.1 Chaos in Hamiltonian system

An important part of the dynamical process is to provide the Hamiltonian function. This depends on the coordinate of space and momentum $H(q_i, p_i), i = 1 \dots n$. The canonical conjugates variables are following

$$\dot{p_i} = -\frac{\partial H}{\partial q_i} \quad \dot{q_i} = \frac{\partial H}{\partial p_i}. \tag{27}$$

Let us given $q_i + \nu_i$, $p_i + \zeta_i$ nearby trajectories up to the linear approximation. Then the modified Hamilton function:

$$H^{'} = H + \left(\zeta_j \frac{\partial H}{\partial p_j} + \nu_j \frac{\partial H}{\partial q_j}\right). \tag{28}$$

The canonical variables are introduced:

$$\dot{\zeta}_i = -\frac{\partial}{\partial q_i}\left(\zeta_j \frac{\partial H}{\partial p_j} + \nu_j \frac{\partial H}{\partial q_j}\right), \tag{29}$$

$$\dot{\nu}_i = \frac{\partial}{\partial p_i}\left(\zeta_j \frac{H}{p_j} + \nu_j \frac{\partial H}{\partial q_j}\right). \tag{30}$$

The equations of motion can be written in the next form:

$$\begin{pmatrix} \dot{\zeta} \\ \dot{\nu} \end{pmatrix} = \begin{pmatrix} -\partial^2_{pq} H & -\partial^2_q H \\ \partial^2_p H & \partial^2_{pq} H \end{pmatrix} \begin{pmatrix} \zeta \\ \nu \end{pmatrix}. \tag{31}$$

If there exist at least one positive eigenvalue of this stability matrix, then the (ξ, ν) distance grows exponentially and the system is chaoticity.

The Lyapunov spectrum L_i is expressed in terms of the monodromy matrix's eigenvalues Λ_i [6]:

$$L_i = \lim_{T\to\infty} \frac{\int_0^T \Lambda_i(t)dt}{T}, \quad i = 1, \ldots, f, \tag{32}$$

where $\Lambda_i(t)$ are the solutions of the characteristic equation:

$$\det[\Lambda_i(t)\mathbf{1} - M(t)] = 0. \tag{33}$$

at a given time t. Here M is the linear stability matrix, and f is the number of degrees of freedom. The discrete definition of the Lyapunov spectrum:

$$L_i' = \langle \Lambda_i \rangle = \frac{1}{n}\sum_{j=1}^{n} \ln \Lambda_i(t_{j-1}), \quad i = 1, \ldots, f, \tag{34}$$

where t_j's are subsequent times along an evolutionary path of the gauge field configurations. In the conservative dynamics the Liouville's theorem is fulfilled:

$$\sum_{i=0}^{f} L_i = 0. \tag{35}$$

In the Hamiltonian system due to the conservation of the energy $L_i = -L_{f-i+1}$ is satisfied for every i. The Kolmogorov-Sinai entropy is expressed by Pesins formula:

$$h^{KS} = \sum_i L_i \Theta(L_i), \tag{36}$$

where $\Theta(x)$ being 1 for positive arguments and 0 otherwise. The dimension of h^{KS} is a rate (1/time) estimating the entropy:

$$S = \frac{h^{KS}}{\mathrm{Re}(L_0)N^3}. \tag{37}$$

4.2 Lattice monodromy matrix

We explain the elements of the matrix in this section. The monodromy matrix is the following:

$$M(t) = \begin{pmatrix} \frac{\partial \dot{U}}{\partial U} & \frac{\partial \dot{U}}{\partial P} \\ \frac{\partial \dot{P}}{\partial U} & \frac{\partial \dot{P}}{\partial P} \end{pmatrix}. \tag{38}$$

The matrix's elements of the lattice Hamiltonian for SU(2) are expounded:

$$\frac{\partial \dot{U}^a}{\partial U^b} = 0, \tag{39}$$

$$\frac{\partial \dot{U}^a}{\partial P^b} = \delta^{ab}, \tag{40}$$

$$\frac{\partial \dot{P}^a}{\partial U^b} = \frac{\partial V^a}{\partial U^b} - \left(\sum_{c=1}^{N} U_c \frac{\partial V^c}{\partial U^b}\right) U^a - V^b U^a - \sum_{c=1}^{N} (U_c V^c + P_c P^c)\delta^{ab}, \tag{41}$$

$$\frac{\partial \dot{P}^a}{\partial P^b} = -2P^b U^a, \quad \text{where} \tag{42}$$

$$\frac{\partial V_k^{\alpha_q}}{\partial U^{\beta_q}} = \sum_{l=1}^{\mathcal{N}} \frac{\partial V_k^{\alpha_q}(U_1,\ldots,U_{\mathcal{N}})}{\partial U_l^{\beta_q}}, \quad \text{ahol } \mathcal{N} = 12, \quad \alpha_q, \beta_q = 0,1,2,3. \tag{43}$$

The over-dots assign the derivative with respect to the scaled time t/a. They are providing information about the stability of trajectories in the neighbourhood of any point of an orbit in the (U, P) phase space. A small perturbation $(\delta U, \delta P)$, evolves in time governed by the monodromy matrix M. It is written by this form:

$$M(t) = \begin{pmatrix} 0 & 1 \\ \frac{\partial \dot{P}}{\partial U} & \frac{\partial \dot{P}}{\partial P} \end{pmatrix}. \tag{44}$$

The eigenvalues of this matrix can be classified as follows: for real and positive eigenvalues, neighbouring trajectories part exponentially and the motion is unstable. In the limit of large time we obtain the Lyapunov components from these eigenvalues to use the expression (34).

5 The eigenvalues of the monodromy matrix

The stability matrix is very rare and the number of element of matrix is very large. We applied the Hessenberg method [11] for the determination of the eigenvalues of the monodromy matrix, because the convergence of this method is very fast.

5.1 Balancing

The balancing procedure enables to take into account the sensitivity of the eigenvalues to rounding errors. The errors is proportional to the Euclidean norm of the matrix. The idea of balancing applies the similarity transformations to make corresponding rows and columns of the matrix have comparable norms, while leaving the eigenvalues unchanged. Balancing is a procedure of order N^2 operations.

We used the algorithm of Osborn. This process contains a sequence of similarity transformations by diagonal matrices D. The rounding errors was investigated during the balancing method, the elements of D are restricted to the powers of the radix, which base applied for floating-point arithmetic (i.e. 2 for most machines). The output is a matrix that is balanced in the norm, which is given by summing the absolute magnitudes of the matrix elements.

5.2 Reduction to Hessenberg form

First we reduced the matrix to a simplified form, it is called Hessenberg form, and the we applied an iterative procedure on the simpler matrix. Such structure can be accomplished by a sequence of Householder transformations, or other method, which is similar to Gaussian elimination with pivoting. Accordingly, the actual elimination procedure used is little bit different from Gauss elimination process.

Before the r-th stage, the original matrix $A \equiv A_1$ has become A_r, which was upper Hessenberg form in its first $r-1$ rows and columns. The r-th stage then contained the following operations:

- Search for the element of maximum magnitude in the r-th column below the diagonal. If it is zero, drop the next two "bullets" and the stage is done. Otherwise, suppose the maximum element was in row r'.
- The rows r' and $r+1$ are swapped (the pivoting procedure). To perform the permutation a similarity transformation, also swapped columns r' and $r+1$.
- For $i = r+2, r+3, \dots N$ compute the multiplier
$$n_{i,r+1} \equiv \frac{a_{ir}}{a_{r+1,r}}.$$
Subtract $n_{i,r+1}$ times row $r+1$ from row i. To perform the elimination of the similarity transformation, we also add $n_{i,r+1}$ times column i to column $r+1$.

A total of $N-2$ such stages are carried out.

When the magnitudes of the matrix elements changed by many orders, we rearranged the matrix so that the largest elements is situated in the top left-hand corner. This decreased the roundoff error (the reduction proceeds from left to right). The operation count is about $5N^3/6$ for large N.

5.3 The QR algorithm for real Hessenberg matrices

We took the following relations for the QR algorithm with shifts:

$$Q_s \cdot (A_s - k_s\mathbf{1}) = R_s \tag{45}$$

where Q is orthogonal and R is upper triangular matrix, and

$$A_{s+1} = R_s \cdot Q_s^T + k_s\mathbf{1} = Q_s \cdot A_s \cdot Q_s^T. \tag{46}$$

The QR transformation keeps the upper Hessenberg form of the original matrix $A \equiv A_1$ and the workload is $O(n^2)$ per iteration on such matrix. As $s \to \infty$, A_s converges to a term, where the eigenvalues are either isolated on the diagonal elements or they are eigenvalues of 2×2 submatrix on the diagonal. This shows a rapid convergence. The basic difference in this situation is that a nonsymmetric real matrix can have complex eigenvalues. This means that the eigenvalues may be complex for a good choices of the shifts k_s.

The complex arithmetic can be used in this process. We need here states that if B is a nonsingular matrix such that

$$B \cdot Q = Q \cdot H, \tag{47}$$

where Q is orthogonal and H is upper Hessenberg, then Q and H are fully determined by the first column of Q.

We used two step of the QR algorithm, either with two real shifts k_s and k_{s+1}, or with complex conjugate values k_s and $k_{s+1} = k^*$. This gives a real matrix A_{s+2}, where

$$A_{s+2} = Q_{s+1} \cdot Q_s \cdot A_s \cdot Q_s^T \cdot Q_{s+1}^T. \tag{48}$$

The Q's are calculated by the next expression:

$$A_s - k_s\mathbf{1} = Q_s^T \cdot R_s \tag{49}$$

$$A_{s+1} = Q_s \cdot A_s \cdot Q_s^T \tag{50}$$

$$A_{s+1} - k_{s+1}\mathbf{1} = Q_{s+1}^T \cdot R_{s+1}. \tag{51}$$

Let us used the equation (50), and the expression (51) can be written:

$$A_s - k_{s+1}\mathbf{1} = Q_s^T \cdot Q_{s+1}^T \cdot R_{s+1} \cdot Q_s. \tag{52}$$

Therefore, if we define

$$M = (A_s - k_{s+1}\mathbf{1}) \cdot (A_s - k_s\mathbf{1}) \tag{53}$$

equations (49) and (52) give

$$R = Q \cdot M, \tag{54}$$

where

$$Q = Q_{s+1} \cdot Q_s \tag{55}$$

$$R = R_{s+1} \cdot R_s. \tag{56}$$

The equation (48) can be rewritten:

$$A_s \cdot Q^T = Q^T \cdot A_{s+2}. \tag{57}$$

We search for an upper Hessenberg matrix H such that

$$A_s \cdot \overline{Q}^T = \overline{Q}^T \cdot H, \tag{58}$$

where $\overline{Q}$ is orthogonal. If $\overline{Q}^T$ has the same first column as Q^T, then $\overline{Q} = Q$ and $A_{s+2} = H$.

The first row of Q is determined as follows. The equation (54) presents that Q is orthogonal matrix and it triangularizes the real matrix M. Any real matrix can be triangularized with a sequence of Householder matrices P_1, P_2, ... P_{n-1}. Thus the matrix Q can expressed by $Q = P_{n-1} \dots P_2 \cdot P_1$.

We need search for $\overline{Q}$, which is satisfying equation (58) whose first row is that of P_1. The Householder matrix P_1 is determined by the first column of M. Because A_s is upper Hessenberg, the equation (53) presents that the first column of M has the form $[p_1, q_1, r_1, 0, \dots 0]^T$, where

$$\begin{aligned} p_1 &= a_{11}^2 - a_{11}(k_s + k_{s+1}) + k_s k_{s+1} + a_{12}a_{21} \\ q_1 &= a_{21}(a_{11} + a_{22} - k_s - k_{s+1}) \\ r_1 &= a_{21}a_{32}. \end{aligned} \tag{59}$$

Therefore

$$P_1 = \mathbf{1} - 2w_1 \cdot w_1^T,$$

where w_1 has only its first 3 elements nonzero. Proceeding in this way up to P_{n-1}, we can see that at each stage the Householder matrix P_r has a vector w_r, which is nonzero only in elements $r, r+1$ and $r+2$. These elements are calculated by the elements $r, r+1$, and $r+2$ in the $(r-1)$-st column of the current matrix.

The result is the next

$$P_{n-1} \cdots P_2 \cdot P_1 \cdot A_s \cdot P_1^T \cdot P_2^T \cdots P_{n-1}^T = H,$$

where H is upper Hessenberg matrix. Thus

$$\overline{Q} = Q = P_{n-1} \cdots P_2 \cdot P_1$$

and

$$A_{s+2} = H.$$

The shifts of the beginning at each stage are formed to the eigenvalues of the 2×2 matrix in the bottom right-hand corner of the current A_s.

This gives

$$\begin{aligned} k_s + k_{s+2} &= a_{n-1,n-1} + a_{nn} \\ k_s k_{s+1} &= a_{n-1,n-1} a_{nn} - a_{n-1,n} a_{n,n-1}. \end{aligned} \qquad (60)$$

Substituting the expression (60) in the equation (59) we get

$$\begin{aligned} p_1 &= a_{21}\{[(a_{nn} - a_{11})(a_{n-1,n-1} - a_{11}) - a_{n-1,n}a_{n,n-1}]/a_{21} + a_{12}\} \\ q_1 &= a_{21}[a_{22} - a_{11} - (a_{nn} - a_{11}) - (a_{n-1,n-1} - a_{11})] \\ r_1 &= a_{21}a_{32}. \end{aligned} \qquad (61)$$

We reduce possible roundoff, when there are small off-diagonal elements. Finally, we perform a double QR step we constructed the Householder matrices P_r $r = 1, \ldots n-1$.

For P_1 we applied p_1, q_1 and r_1, which were given by expressions (61). The remaining matrices, p_r, q_r and r_r were calculated by the $(r, r-1)$, $(r+1, r-1)$, and $(r+2, r-1)$ elements of the current matrix. The number of arithmetic operations can be decreased by writing the nonzero elements of the $2w \cdot w^T$ part of the Householder matrix in the form

$$2w \cdot w^T = \begin{bmatrix} (p \pm s)/(\pm s) \\ q/(\pm s) \\ r/(\pm s) \end{bmatrix} \cdot [1 \;\; q/(\pm s) \;\; r/(p \pm s)],$$

where

$$s^2 = p^2 + q^2 + r^2.$$

If we proceed in this way, convergence is usually very fast, which is need to control from step to step. There are two possible ways of terminating the iteration for an eigenvalue. First, if $a_{n,n-1}$ becomes 'negligible', then a_{nn} is an eigenvalue. We can then delete the n-th row and column of matrix and find the next eigenvalue. Otherwise $a_{n-1,n-2}$ may become negligible. Then the eigenvalues of the 2×2 matrix in the lower right-hand corner may be taken to be eigenvalues. We delete the n-th and $(n-1)$-th rows and column of the matrix and continue the process. The operation count for the QR algorithm described here is $\sim 5k^2$ per iteration, where k is the current size of the matrix.

In the next (6.) Section the significant question is the parallelisation of the Hessenberg method in rare matrix.

6 Parallel hybrid Hessenberg method

In [5] we used the CUDA platform to develop a parallel version of the Yang-Mills algorithm for lattice calculations. Here we give the details how we have moved forward from there by applying parallelism to calculate the eigenvalues of the monodromy matrix (4.1. subsection), which helps to show the chaos in the non-Abelian gauge field theory.

6.1 Main idea

Examining the Hessenberg method we can easily differentiate parts that has more computational intensive tasks, while others are not so performance sensitive. Hence the Block Hessenberg Algorithm is used. In this instead of taking the whole matrix as the input of the transformation process we divide it into smaller blocks. We take these blocks and calculate the Householder vector for each column in that block and with it update the consecutive columns. When finished we use the accumulated Householder transformations to update the rest of the whole matrix. We repeat this until we update all the blocks. This way with the accumulated Householder transformations in overall less matrix multiplications will be used compared to the original Hessenberg Algorithm in which case we always have to update every column with the calculated Householder vector.

6.1.1 Block householder algorithm

To calculate the Householder vectors we use the following [10]:

$$v = (\frac{1}{A_{k+1,k+\sigma}}) \tag{62}$$

$$\sigma = sign(A_{k+1,k}\|x\|_2) \tag{63}$$

$$V_k = [v_1, v_2, ..., v_k] \tag{64}$$

Compact-WY representation of the k Householder transformations:

$$\begin{gathered}(I - v_1 v_1^T)...(I - v_k v_k^T) = I - V_k T_k V_k^T \\ A := A(I - V_L T_L V_L^T) = A - Y_L V_L^T \\ A := (I - V_L T_L V_L^T)A\end{gathered}$$

$$T_k = \begin{bmatrix} T_{k-1} & -\tau_k T_{k-1} V_{k-1}^T v_k \\ 0^T & \tau_k \end{bmatrix} \tag{65}$$

$$Y_k = AV_k T_k = \begin{bmatrix} Y_{k-1} & \tau_k(-Y_{k-1} V_{k-1}^T v_k + A v_k) \end{bmatrix} \tag{66}$$

We initialize V, T and Y as follows:

$$\begin{gathered}V_1 = [v_1] \\ T_1 = [\tau_1] \\ Y_1 = [AV_1 T_1]\end{gathered}$$

Update formula for one column of a block

$$A_{*,k} := A_{*,k} - Y_k((V_k)_{k,*})^T \tag{67}$$

$$A_{*,k} := (I - V_k T_k^T V_k^T)A_{*,k} \tag{68}$$

Update formula for the rest of the matrix

$$A_{*,L+1:n} := A_{*,L+1:n} - Y_L((V_L)_{L+1:n,*})^T \tag{69}$$

$$A_{*,L+1:n} := (I - V_L T_L^T V_L^T)A_{*,L+1:n} \tag{70}$$

With the hybrid implementation we extend the algorithm to the GPU as much as reasonably possible. The GPU will need a high amount of data to be able to achieve high utilization [15] and thus high performance, so the low on data parts of the calculation are kept on the CPU while the intensive matrix multiplications are pushed to the GPU.

Before commencing any calculations we upload the matrix into the GPUs memory, after that we follow the next steps for the k^{th} block:
For every column (i) in the block:

1. Compute the Householder vector (v), the i^{th} column of V (CPU) [eq. 62,63,64]

2. Update T and Y matrices (CPU) [eq. 65,66]

3. Update the next column (CPU) [eq. 67,68]

   After updating the block:

4. Update the rest of the matrix with V,Y,T (GPU) [eq. 69,70]

5. Copy over the next block to the CPU as it has been updated on the GPU

6. Continue the reduction

6.2 Restrictions

The monodromy matrix can become very big as we increase the N parameter of the lattice, thus requiring a lot of memory, which can go up to the TB range. Because of this right now reasonable results can be achieved only up to $N = 6$.

6.3 Implementation

For implementation and testing we have used a GeForce GTX 980M with compute capability 5.2 (Table 1) and an Intel Core i7-4710HQ CPU that does not have an IGP (Table 2).

As the starting point the original matrix for which we would like to compute the Hessenberg form will be uploaded to the GPU. After this as we compute the new Hessenberg vectors in the V matrix trough the blocks and update the T, Y matrices we are providing every element for the GPU to update our

	GeForce GTX 980M
Technical Specifications	Compute Capability 5.2
Transistors (Million)	5200
Memory (GB)	4
Memory Bandwidth (GB/s)	160
GFLOPs	3189
TDP (watts)	125

Table 1: GeForce GTX 980M technical specifications.

	i7-4710HQ
Transistors (Million)	1400
Connected memory (GB)	24
Memory Bandwidth (GB/s)	25.6
GFLOPs	422
TDP (watts)	47

Table 2: Core i7-4170HQ technical specifications.

original matrix. We copy the V, T, Y matrices to the GPU and using matrix-matrix multiplication we do the update. After this as the matrix has been changed the next block will be copied over to the CPU side. We choose the blocks to be 32 columns wide.

For doing the matrix multiplications in parallel on the GPU we use the NVIDIA developed CUBLAS library's *cublasdgemm* function, while on the CPU we use LAPACK with Intel MKL BLAS.

6.3.1 Comparison of the theoretical performance

Here we provide a comparison between the GPU's and CPU's achievable performance based on the GFLOPS and TDP values.

If we look at the computational power of the used processors we can see that the GPU is 7.5 times higher than what the CPU can provide. The GFLOPS of the CPU was calculated using the following formula:

$$\mathsf{GFLOPS} = \mathsf{cores} * \mathsf{clock} * \frac{\mathsf{FLOPs}}{\mathsf{cycle}} / 1000$$

The clock rate of a Core i7-4710HQ on full load with all 4 cores activated is

3.3 GHz and the maximum FLOPs/cycle is 32 [16], thus the maximum performance number in Table 2. If we would like to reach the GPU's performance with the actual CPU architecture, this means we will need 7.5 times more power. This means if we stay on the Haswell architecture and we will just try to increase the throughput we will reach a TDP of 352.5 watts. This leads us to the conclusion if we would like to have the same performance on the CPU that we have on the GPU will need 2.82 times more power for the CPU. This will also mean that the physical limitations will not hold back the increase of clock rate and power which can never be true, leaving the GPUs the most efficient processors.

6.4 Numerical results

To calculate the eigenvalues of the monodromy matrix we used the Hessenberg method. To make the process more efficient we applied the block Hessenberg model to be able to utilize parallelization.

For overall testing the following system was used (Table 3):

CPU	GPU	OS	Compiler	CUDA version
Intel Core i7 4710HQ	GeForce GTX 980M	Windows 10 Pro	Visual C++ 2013	7.0

Table 3: The used system's specification.

The numerical results fulfill the physical principle, the constraint value of the physical quantity remains constant during the time evolution of the equation of motion. The Lyapunov Spectrum (Figure 2) justifies the existence of the chaotic motion in the Yang-Mills fields.

We compared the runtime of the CPU to the GPU (Figure 1), the GPU gives substantially better results as we increase the available work. Evaluating the same lattice size the runtime on the GPU shows acceleration of a magnitude of 3.

The Kolmogorov-Sinai entropy (Figure 3) is obtained from the evolution eigenvalues of the monodromy matrix as functions of the scaled energy. These results gives good approximation for an ideal gas (S logE).

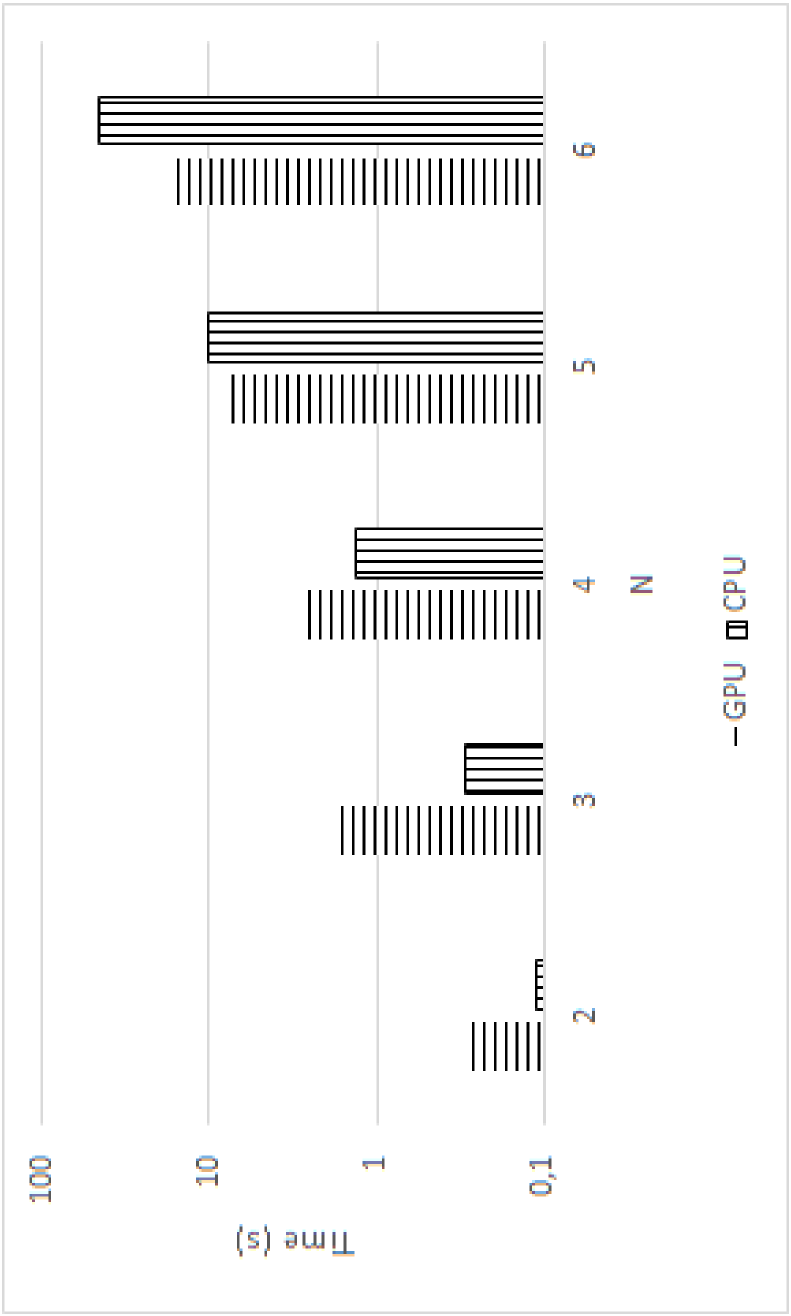


Figure 1: Runtime on the CPU and on the GPU with $\mathsf{N} = 2, 3, 4, 5, 6$.

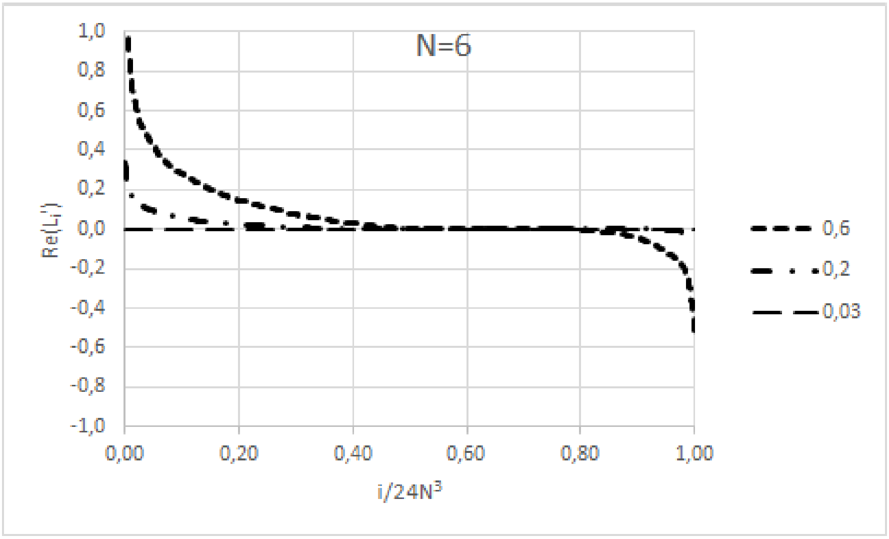


Figure 2: The Lyapunov spectrum on N=6.

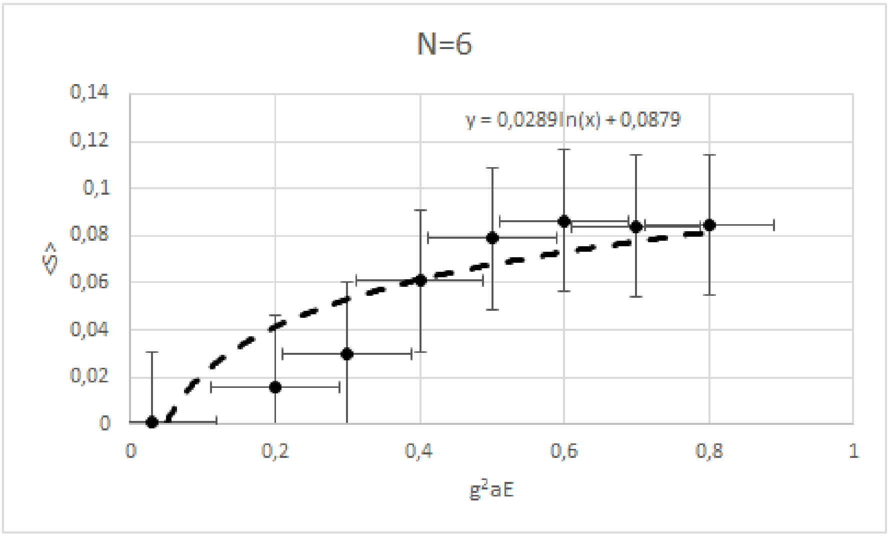


Figure 3: The Kolmogorov-Sinai entropy on N=6.

7 Summary

As the GPUs are becoming more powerful with each new architecture it becomes easier to modify the existing applications and algorithms to be parallel. In our case the Block Hamilton algorithm was able to achieve a 3 fold speed up compared to the CPU version, while computing the eigenvalues of the monodromy matrix.

By moving from CPU to GPU the eigenvalues are the same, thus keeping the physical principles valid. Physical constant quantities remains constraint while solving the equation of motion by parallel algorithm, such as the total energy.

The performance of the GPUs make it possible to calculate eigenvalues of the monodromy matrix to evince the chaotic behaviour of Yang-Mills system.

References

[1] T. S. Biró, Conserving algorithms for real-time non-Abelian lattice gauge theories, *Int. Journ. of Modern Phys. C* **6** (1995) 327–344. ⇒221

[2] T. S. Biró, A. Fülöp, C. Gong, S. Matinyan, B. Müller, A. Trajanov, Chaotic dynamics in classical lattice field theories,*165th WE-Heraeus Seminar on Theory of Spin Lattices and Lattice Gauge Models*, 14–19 Oct 1996. Bad Honnef, Germany, *Lec. Notes in Physics* **494** (1997) 164–176. ⇒217, 220

[3] T. S. Biró, C. Gong, B. Müller, A. Trayanov, Hamiltonian dynamics of Yang-Mills fields on a lattice, *Int. Journ. of Modern Phys. C* **5** (1994) 113–149. ⇒ 220

[4] M. Creutz, *Quarks, Gluons and Lattices*, Cambridge University Press, Cambridge CB2 1RP, 1983. ⇒219

[5] R. Forster, A. Fülöp, Yang-Mills lattice on CUDA, *Acta Univ. Sapientiae, Informatica*, **5**, 2 (2013) 184–211. ⇒217, 230

[6] A. Fülöp, T. S. Biró, Towards the equation of state of a classical SU(2) lattice gauge theory, *Phys. Rev. C* **64** (2001) 064902(5). *arxiv.org.* ⇒224

[7] C. Gong, Lyapunov spectra in SU(2) lattice gauge theory, *Phys. Rev D* **49** (1994) 2642–2645. ⇒222

[8] B. Müller, A. Trayanov, Deterministic chaos on non-abelian lattice gauge theory, *Phys. Rev. Letters* **68,** 23 (1992) 3387–3390. ⇒217

[9] I. Montvay, G. Münster, *Quantum fields on a lattice*, Cambridge University Press, Cambridge CB2 1RP, 1994. ⇒218

[10] J. Muramatsu, T. Fukaya, S. Zhang, Acceleration of Hessenberg Reduction for Nonsymmetric Eigenvalue Problems in a Hybrid CPU-GPU Computing Environment, *Intern. J. of Networking and Computing* **1,** 2 (2011) 132–143. ⇒217, 231

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipies in C*, Cambridge University Press, 2002. ⇒217, 225

[12] L. E. Reichl, *The Transition to Chaos*, Springer-Verlag, 1992. ⇒221

[13] S. Weinberg, *The Quantum Theory of Fields*, Cambridge University Press CB2 1RP, 1996 ⇒217

[14] C. N. Yang, R. Mills, Conservation of isotropic spin and isotopic gauge invariance, *Phys. Rev.* **96** (1954) 191–195. ⇒218

[15] CUDA C Programming Guide NVIDIA Corp., 2013, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. ⇒232

[16] Technology Insight: Intel Next Generation Microarchitecture Code Name Haswell, IDF2012. ⇒234

Received: September 30, 2015 • Revised: December 29, 2015

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English.
Information about each series can be found at
http://www.acta.sapientia.ro.

Acta Universitatis Sapientiae, Informatica

Each volume contains two issues.

Sapientia University DE GRUYTER OPEN Scientia Publishing House

ISSN 1844-6086
http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: http://www.acta.sapientia.ro.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (acta-inf@acta.sapientia.ro) only, using the LaTeX style and sample file at the address http://www.acta.sapientia.ro. Beside the LaTeX source a pdf format of the paper is necessary too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (http://www.acm.org/about/class/1998) and AMS Mathematics Subject Classification codes (http://www.ams.org/msc/).

References should be listed alphabetically based on the Intructions for Authors given at the address http://www.acta.sapientia.ro.

Illustrations should be given in Encapsulated Postscript (eps) format.

One issue is offered each author free of charge. No reprints will be available.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by Idea Printing House
Director: Péter Nagy

ISSN 1844-6086
http://www.acta.sapientia.ro