Modification of Valiant's Parsing Algorithm for the String-Searching Problem*

Yuliya Susanina[1][0000−0003−3904−3764], Anna Yaveyn[2,3][0000−0001−9733−5429], and Semyon Grigorev[3][0000−0002−7966−0698]

[1] Saint Petersburg State University, 7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
[2] JetBrains Research, Universitetskaya emb., 7-9-11/5A
St.Petersburg, Russia
jsusanina@gmail.com, anya.yaveyn@yandex.ru, semen.grigorev@jetbrains.com

Abstract. Some string-matching problems can be reduced to parsing: verification whether some sequence can be derived in the given grammar. To apply parser-based solutions to such area as bioinformatics, one needs to improve parsing techniques so that the processing of large amounts of data was possible. The most asymptotically efficient parsing algorithm that can be applied to any context-free grammar is a matrix-based algorithm proposed by Valiant. This paper presents a modification of the Valiant's algorithm, which facilitates efficient utilization of modern hardware in highly-parallel implementation. Moreover, the modified version significantly decreases the number of excessive computations, accelerating the search of substrings.

Keywords: Context-free grammar · Parsing · Valiant's algorithm · String-matching · Secondary structure.

1 Introduction

The secondary structure of RNA's is tightly related to biological functions of organisms. So its analysis plays an important role in organisms classification and recognition problems.

Specific features of secondary structure can be described by some context-free grammar (CFG). There is a number of approaches to sequences analysis based on parsing: verification whether the given sequence can be derived in the specified grammar. Some approaches to secondary structure analysis model a string with correlated symbols with probabilistic formal grammars [4, **?**]. For some problems, it is necessary to find all derivable substrings of the given string [1]. This case is the string-matching problem also known as string-searching problem.

Most CFG-based approaches suffer the same issue: the computational complexity is poor. Traditionally used CYK [3, **?**] runs with a cubic time complexity

* The research was supported by the Russian Science Foundation, grant No. 18-11-00100.

and demonstrates poor performance on long strings or big grammars [5]. We argue that more efficient algorithms are needed in such field as bioinformatics where a large amount of data is common. In some cases, context-free grammars are not expressive enough. Fortunately, some features can be expressed with more exotic grammar classes: for example, pseudoknots can be described by using conjunctive grammars [8], while it is impossible by using context-free one.

Asymptotically most efficient parsing algorithm is Valiant's algorithm [7] which is based on matrix multiplication. Okhotin generalized this algorithm to conjunctive and Boolean grammars which are the natural extensions of CFG which have more expressive power [6]. Valiant's algorithm simplifies the utilization of parallel techniques to improve performance by offloading critical computations onto matrices multiplication. However, this algorithm is not suitable for the string-matching problem.

In this paper we present the modification of Valiant's algorithm, which improves utilization of GPGPU and parallel computations by computing some matrices products concurrently. Also, the proposed algorithm can be easily utilized for the string-matching problem.

2 Background

In this section we introduce some basic definitions from the formal language theory, and describe Valiant's parsing algorithm on which we base our solution.

2.1 Formal languages

An alphabet Σ is a finite nonempty set of symbols. Σ^* is a set of all finite strings over Σ. A context-free grammar G_S is a quadruple (Σ, N, R, S), where Σ is a finite set of terminals, N is a finite set of nonterminals, R is a finite set of productions of the form $A \to \beta$, where $\Sigma \cap N = \varnothing$, $A \in N, \beta \in V^*$, $V = \Sigma \cup N$ and $S \in N$ is a start symbol. Context-free grammar $G_S = (\Sigma, N, R, S)$ is said to be in Chomsky normal form if all productions in R are of the form: $A \to BC$, $A \to a$, or $S \to \varepsilon$, where $A, B, C \in N, a \in \Sigma, \varepsilon$ is an empty string. For each context-free grammar G of length N we can find an equivalent grammar in Chomsky normal form with length N^2 [2].

$L_G(S) = \{\omega | S \xrightarrow[G_S]{*} \omega\}$ is a language specified by the grammar $G_S = (\Sigma, N, R, S)$, where $A \xrightarrow[G_S]{*} \omega$ means that ω can be derived in a finite number of rules applications from the start symbol S.

2.2 Parsing and bioinformatics

...

2.3 Valiant's parsing algorithm

Tabular parsing algorithms construct a matrix T, cells of which are filled with nonterminals from which the corresponding substring can be derived. These algorithms usually work with the grammar in Chomsky normal form. For $G_S = (\Sigma, N, R, S)$, $T_{i,j} = \{A | A \in N, a_{i+1} \dots a_j \in L_G(A)\} \quad \forall i < j$.

The elements of T are filled successively starting with $T_{i-1,i} = \{A | A \to a_i \in R\}$. Then, $T_{i,j} = f(P_{i,j})$, where $P_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$ and $f(P) = \{A | \exists A \to BC \in R : (B, C) \in P\}$. Finally, the input string $a_1 a_2 \dots a_n$ belongs to $L_G(S)$ iff $S \in T_{0,n}$.

If all cells are filled sequentially, the time complexity of this algorithm is $O(n^3)$. Valiant proposed to offload the most intensive computations to the Boolean matrix multiplication. The most time-consuming is computing $\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$ and Valiant's idea is to compute $T_{i,j}$ by multiplication of submatrices of T.

Listing 1: Parsing by Matrix Multiplication: Valiant's Version

Input: Grammar $G = (\Sigma, N, R, S), w = a_1 \dots a_n, n \geq 1, a_i \in \Sigma$, where $n + 1 = 2^k$

main():
compute(0, n + 1);
accept if and only if $S \in T_{0,n}$

compute(l, m):
if $m - l \geq 4$ **then**
    compute(l, $\frac{l+m}{2}$*);*
    compute($\frac{l+m}{2}$, m*)*
complete(l, $\frac{l+m}{2}$, $\frac{l+m}{2}$, m*)*

complete(l, m, l', m'):
if $m - l = 4$ *and* $m = l'$ **then** $T_{l,l+1} = \{A | A \to a_{l+1} \in R\}$;
else if $m - l = 1$ *and* $m < l'$ **then** $T_{l,l'} = f(P_{l,l'})$;
else if $m - l > 1$ **then**
    $leftgrounded = (l, \frac{l+m}{2}, \frac{l+m}{2}, m)$, $rightgrounded = (l', \frac{l'+m'}{2}, \frac{l'+m'}{2}, m')$,
    $bottom = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2}), left = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$,
    $right = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m'), top = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$;
    complete(bottom);
    $P_{left} = P_{left} \cup (T_{leftgrounded} \times T_{bottom})$;
    complete(left);
    $P_{right} = P_{right} \cup (T_{bottom} \times T_{rightgrounded})$;
    complete(right);
    $P_{top} = P_{top} \cup (T_{leftgrounded} \times T_{right})$;
    $P_{top} = P_{top} \cup (T_{left} \times T_{rightgrounded})$;
    complete(top)

Multiplication of two submatrices of parsing table T is defined as follows. Let $X \in (2^N)^{m \times l}$ and $Y \in (2^N)^{l \times n}$ be two submatrices of the parsing table T. Then, $X \times Y = Z$, where $Z \in (2^{N \times N})^{m \times n}$ and $Z_{i,j} = \bigcup_{k=1}^{l} X_{i,k} \times Y_{k,j}$.

Note that the computation of $X \times Y$ can be replaced by the multiplication of $|N|^2$ Boolean matrices (for each nonterminal pair). Denote the matrix corresponding to the pair $(B, C) \in N \times N$ as $Z^{(B,C)}$, then $Z^{(B,C)}_{i,j} = 1$ iff $(B, C) \in Z_{i,j}$. It should also be noted that $Z^{(B,C)} = X^B \times Y^C$. Each Boolean matrix multiplication can be computed independently. Following these changes, time complexity of this algorithm is $O(|G|\text{BMM}(n)log(n))$ for an input string of length n, where $\text{BMM}(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$.

Valiant's algorithm written as proposed by Okhotin is presented in Listing 1. All elements of T and P are initialized by empty sets. Then, the elements of these two table are successively filled by two recursive procedures.

The procedure $compute(l, m)$ computes correct values of $T_{i,j}$ for all $l \leq i < j < m$.

The procedure $complete(l, m, l', m')$ constructs the submatrix $T_{i,j}$ for all $l \leq i < m$, $l' \leq j < m'$. This procedure assumes $T_{i,j}$ for all $l \leq i < j < m, l' \leq i < j < m'$ are already constructed and the current value of $P_{i,j} = \{(B, C) | \exists k, (m \leq k < l'), a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$ for all $l \leq i < m, l' \leq j < m'$. The submatrix partition during the procedure call is shown in Figure 1.

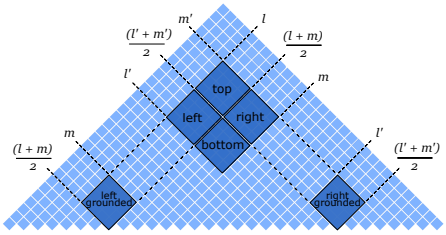


Fig. 1. Matrix partition used in procedure *complete(l, m, l', m')*

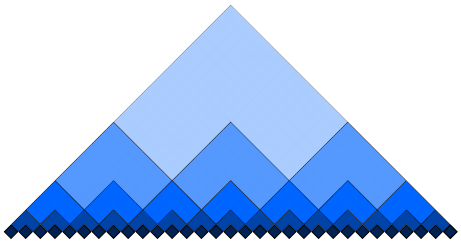


Fig. 2. Matrix partition on V-shaped layers used in modification

A simple example of parsing with the Valiant's algorithm is presented in Figure 3. Only several steps are shown, but it is enough to compare our version with the original algorithm.

3 Modified Valiant's algorithm

In this section, we propose how to rearrange the order in which submatrices are processed in the algorithm. The different order improves the independence of submatrices handling and facilitates the implementation of parallel submatrix processing.

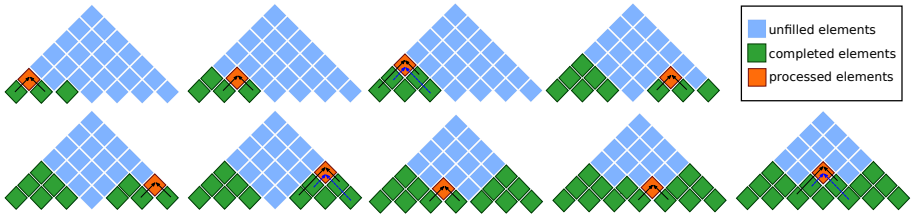


Fig. 3. An example of beginning of Valiant's algorithm

3.1 Layered submatrices processing

We propose to divide the parsing table into layers of disjoint submatrices of the same size (see Figure 2). Such division is possible because the derivation of a substring of the fixed length does not depend on either left or right contexts. An appropriate order of substrings processing guarantees the disjointness of submatrices which form a layer. Each layer consists of square matrices which size is a power of 2. The layers are computed successively in the bottom-up order. Each matrix in the layer can be handled independently, which facilitates parallelization of layer processing.

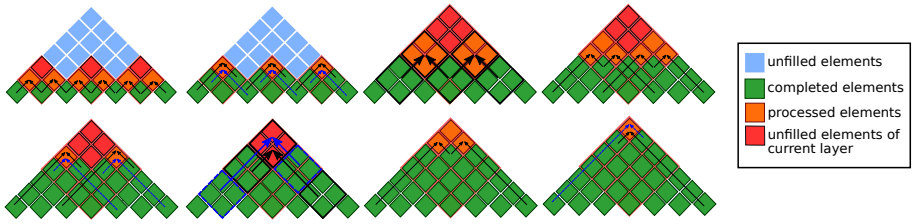


Fig. 4. An example of the modification of Valiant's algorithm

Figure 4 demonstrates the modified algorithm. The lowest layer (submatrices of size 1) has already been computed. The second layer is filled in in the steps 1-2. Although the original algorithm computes the same matrix, the modified one needs only two steps using parallel computation of submatrix products.

The modified version of Valiant's algorithm is presented in Listing 2. The procedure *main()* computes the lowest layer ($T_{l,l+1}$), and then divides the table into layers, and computes them with the *completeVLayer()* function. Thus, *main()* computes all elements of parsing table T.

We define *left(subm)*, *right(subm)*, *top(subm)*, *bottom(subm)*, *rightgrounded(subm)* and *leftgrounded(subm)* functions which return the submatrices for matrix $subm = (l, m, l', m')$ according to the original Valiant's algorithm (Figure 2).

The procedure *completeVLayer(M)* takes an array of disjoint submatrices M which represents a layer. For each *subm = (l, m, l', m')* $\in M$ this procedure computes *left(subm), right(subm), top(subm)*. The procedure assumes that the

Listing 2: Parsing by Matrix Multiplication: Modified Version

Input: $G = (\Sigma, N, R, S), w = a_1 \dots a_n, n \geq 1, n+1 = 2^p, a_i \in \Sigma$

main():
for $l \in \{1, \dots, n\}$ **do** $T_{l,l+1} = \{A | A \to a_{l+1} \in R\}$;
for $1 \leq i < p-1$ **do**
    layer = constructLayer(i);
    completeVLayer(layer)
accept if and only if $S \in T_{0,n}$

constructLayer(i):
$\{(k2^i, (k+1)2^i, (k+1)2^i, (k+2)2^i) \mid 0 \leq k < 2^{p-i} - 1\}$

completeLayer(M):
if $\forall (l, m, l', m') \in M \quad (m - l = 1)$ **then**
    for $(l, m, l', m') \in M$ **do** $T_{l,l'} = f(P_{l,l'})$;
else
    completeLayer($\{$*bottom(subm)* $\mid$ *subm* $\in M\}$);
    completeVLayer(M)

completeVLayer(M):
multiplicationTasks$_1$ =
  $\{$*left(subm), leftgrounded(subm), bottom(subm)* $\mid$ *subm* $\in M\} \cup$
  $\{$*right(subm), bottom(subm), rightgrounded(subm)* $\mid$ *subm* $\in M\}$;
multiplicationTask$_2$ = $\{$*top(subm), leftgrounded(subm), right(subm)* $\mid$ *subm* $\in M\}$;
multiplicationTask$_3$ = $\{$*top(subm), left(subm), rightgrounded(subm)* $\mid$ *subm* $\in M\}$;
performMultiplications(multiplicationTask$_1$*)*;
completeLayer($\{$*left(subm)* $\mid$ *subm* $\in M\} \cup \{$*right(subm)* $\mid$ *subm* $\in M\}$*)*;
performMultiplications(multiplicationTask$_2$*)*;
performMultiplications(multiplicationTask$_3$*)*;
completeLayer($\{$*top*$(subm)$ $\mid$ *subm* $\in M\}$*)*

performMultiplication(tasks):
for $(m, m1, m2) \in$ *tasks* **do** $P_m = P_m \cup (T_{m1} \times T_{m2})$;

elements of *bottom(subm)* and $T_{i,j}$ for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$ are already constructed. Also it is assumed that the current value of $P_{i,j} = \{(B, C) | \exists k, (m \leq k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}$ for all i and j such that $l \leq i < m$ and $l' \leq j < m'$.

The procedure *completeLayer(M)* also takes an array of disjoint submatrices M, but unlike the previous one, it computes $T_{i,j}$ for all $(i, j) \in subm$. This procedure requires exactly the same assumptions on $T_{i,j}$ and $P_{i,j}$ as in the previous case.

In other words, *completeVLayer(M)* computes the entire layer M and *completeLayer*(M_2) is a helper function which is necessary for computation of smaller square submatrices $subm_2 \in M_2$ inside of M.

Finally, the procedure *performMultiplication(tasks)*, where *tasks* is an array of triples of submatrices, performs the basic step of the algorithm: matrix multiplication. It is worth mentioning that $|tasks| \geq 1$ and each task can be computed independently, while the original algorithm handles one *task* per step sequentially. So, the practical implementation of this procedure can easily utilize different techniques of parallel array processing.

3.2 Correctness and complexity

We provide the proof of correctness and time complexity for the proposed modification in this section. To do it we should prove correctness of subprocedure *completeLayer*.

Lemma 1. *Let M be a layer. If for all $(l, m, l', m') \in M$:*

1. $T_{i,j} = \{A | a_{i+1} \dots a_j \in L_G(A)\}$ *for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$;*
2. $P_{i,j} = \{(B, C) | \exists k, (m \leq k < l') : a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}$ *for all $l \leq i < m$ and $l' \leq j < m'$.*

Then the procedure completeLayer(M), returns correctly computed sets of $T_{i,j}$ for all $l \leq i \leq m$ and $l' \leq j \leq m'$ for all $(l, m, l', m') \in M$.

Proof. Proof by induction on $m - l$.

Theorem 1. *Algorithm from listing 2 correctly computes $T_{i,j}$ for all i and j, thus an input string $a = a_1 a_2 \dots a_n \in L_G(S)$ if and only if $S \in T_{0,n}$.*

Proof. Primarily to prove the theorem, we show by induction that all layers of the parsing table T are computed correctly.

Basis: layer of size 1×1. Parsing table T consists of one layer of size 1 and its elements are correctly computed in lines 2-3 in listing 2.

Inductive step: assume any layer of size less than or equal to $2^{p-2} \times 2^{p-2}$ are computed correctly.

Define layer of size $2^{p-1} \times 2^{p-1}$ as M. Hereinafter *subm = (l, m, l', m')* is a typical element of layer M.

Consider *completeVLayer(M)* call.

Firstly, *performMultiplications(multiplicationTask$_1$)* adds to each $P_{i,j}$ all pairs (B, C) such that $\exists k$, $(\frac{l+m}{2} \leq k < l')$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i, j) \in leftsublayer(M)$ and (B, C) such that $\exists k$, $(m \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i, j) \in rightsublayer(M)$. Now *completeLayer(leftsublayer(M) ∪ rightsublayer(M))* can be called and it returns correctly computed *leftsublayer(M) ∪ rightsublayer(M)*.

Then *performMultiplications* called with arguments *multiplicationTask$_2$* and *multiplicationTask$_3$* adds pairs (B, C) such that $\exists k$, $(\frac{l+m}{2} \leq k < m)$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ and (B, C) such that $\exists k$, $(l' \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ to each $P_{i,j}$ for all $(i, j) \in topsublayer(M)$. So as

$m = l'$ (from the construction of the layer), condition for elements of matrix P are fulfilled. Now *completeLayer(topsublayer(M))* can be called and it returns correctly computed *topsublayer(M)*.

All $T[i, j]$ $\forall (i, j) \in M$ are computed correctly.

Thus, *completeVLayer(M)* returns correct $T_{i,j}$ for all $(i, j) \in M$ for any layer M of parsing table T and lines 4-6 in listing 2 return all $T_{i,j} = \{A | A \in N, a_{i+1} \dots a_j \in L_G(A)\}$.

Lemma 2. *Let $calls_i$ is a number of the calls of completeVLayer(M) where for all $(l, m, l', m') \in M$ with $m - l = 2^{p-i}$.*

- *for all $i \in \{1, .., p-1\}$ $\sum_{n=1}^{calls_i} |M|$ is exactly $2^{2i-1} - 2^{i-1}$;*
- *for all $i \in \{1, .., p-1\}$ products of submatrices of size $2^{p-i} \times 2^{p-i}$ are calculated exactly $2^{2i-1} - 2^i$ times.*

Proof. Prove the first statement by induction on i.

Basis: i = 1. $calls_1$ and $|M| = 1$. So, $2^{2i-1} - 2^{i-1} = 2^1 - 2^0 = 1$.

Inductive step: assume that $\sum_{n=1}^{calls_i} |M|$ is exactly $2^{2i-1} - 2^{i-1}$ for all $i \in \{1, .., j\}$.

Let us consider $i = j + 1$.

Firstly, note that function *costructLayer(i)* returns $2^{p-i} - 1$ matrices of size 2^i, so in the call of *completeVLayer(costructLayer(k - i)) costructLayer(k - i)* returns $2^i - 1$ matrices of size 2^{p-i}. Secondly, *completeVLayer(M)* is called 3 times for the left, right and top submatrices of size $2^{p-(i-1)}$. Finally, *completeVLayer(M)* is called 4 times for the bottom, left, right and top submatrices of size $2^{p-(i-2)}$, except $2^{i-2} - 1$ matrices which were already computed.

Then, $\sum_{n=1}^{calls_i} |M| = 2^i - 1 + 3 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) + 4 \times (2^{2(i-2)-1} - 2^{(i-2)-1}) - (2^{i-2} - 1) = 2^{2i-1} - 2^{i-1}$.

Now we know that $\sum_{n=1}^{calls_{i-1}} |M|$ is $2^{2(i-1)-1} - 2^{(i-1)-1}$ and we can calculate the number of products of submatrices of size $2^{p-i} \times 2^{p-i}$. During these calls *performMultiplications* run 3 times, $|multiplicationTask1| = 2 \times 2^{2(i-1)-1} - 2^{(i-1)-1}$ and $|multiplicationTask2| = |multiplicationTask3| = 2^{2(i-1)-1} - 2^{(i-1)-1}$. So, the number of products of submatrices of size $2^{p-i} \times 2^{p-i}$ is $4 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) = 2^{2i-1} - 2^i$.

Theorem 2. *Let $|G|$ be a length of the description of the grammar G and let n be a length of an input string. Then algorithm from listing 2 calculates matrix T in $\mathcal{O}(|G| BMM(n) \log n)$ where BMM(n) is the number of operations needed to multiply two Boolean matrices of size $n \times n$.*

Proof. The proof is almost identical with the proof of theorem 1 given by Okhotin [6], because, as shown in the last lemma, the Algorithm 1 has the same number of products of submatrices.

To summarize, the correctness of the modification was proved and it was shown that the time complexity remained the same as in Valiant's version.

3.3 Algorithm for substrings

Next, we show how our modification can be applied to the string-matching problem.

To find all substrings of size s, which can be derived from a start symbol for an input string of size $n = 2^p$, we need to compute layers with submatrices of size not greater than $2^{l'}$, where $2^{l'-2} < s \leq 2^{l'-1}$.

Let $l' = p - (m-2)$ and consequently $(m-2) = p - l'$. For any $m \leq i \leq p$ products of submatrices of size 2^{p-i} are calculated exactly $2^{2i-1} - 2^i$ times and each of them imply multiplying $\mathcal{O}(|G|)$ Boolean submatrices. Now we estimate the number of operations needed to find all substrings:

$$C \cdot \sum_{i=m}^{p} 2^{2i-1} \cdot 2^{\omega(p-i)} \cdot f(2^{p-i}) = C \cdot 2^{\omega l'} \sum_{i=2}^{l'} 2^{(2-\omega)i} \cdot 2^{2(p-l')-1} \cdot f(2^{l'-i}) \leq$$
$$C \cdot 2^{\omega l'} f(2^{l'}) \cdot 2^{2(p-l')-1} \sum_{i=2}^{l'} 2^{(2-\omega)i} = \mathrm{BMM}(2^{l'}) \cdot 2^{2(p-l')-1} \sum_{i=2}^{l'} 2^{(2-\omega)i}$$

Thus, time complexity for searching all substrings is $O(|G|\mathrm{BMM}(2^{l'})(l'-1))$, while time complexity for the full input string is $O(|G|\mathrm{BMM}(2^p)(p-1))$. The Valiant's algorithm completely calculate at least 2 triangle submatrices of size $\frac{n}{2}$, as shown in Figure 5, thus the minimum asymptotic complexity is $O(|G|\mathrm{BMM}(2^{p-1})(p-2))$. Thus we can conclude that the modification is asymptotically faster than the original algorithm for substrings of size $s \ll n$.

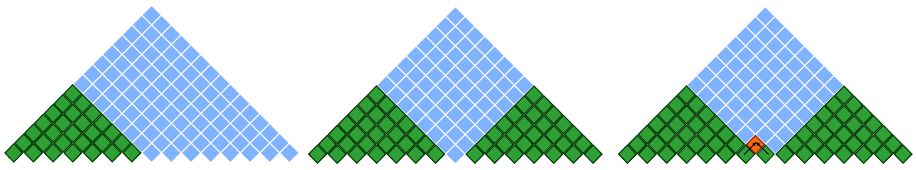


Fig. 5. The number of elements necessary to compute in Valiant's algorithm. It is necessary to calculate at least 2 triangle submatrices of size $\frac{n}{2}$.

4 Evaluation

5 Conclusion

We presented a modification of Valiant's algorithm, which makes it possible to use parallel computations more efficiently. Also we showed its applicability to the string-matching problem with which the original algorithm was not able to work.

The directions for future research is to extend the proposed algorithm for conjunctive and boolean grammars handling. It will be useful for complex secondary structure features processing.

```
s: s s | ( s ) | [ s ] | ε
```

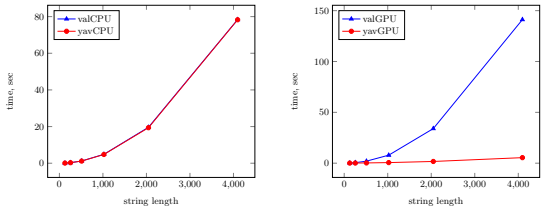
```
s: stem<s0>
any_str: any_smb*[2..10]
s0: any_str | any_str stem<s0> s0
any_smb: A | U | C | G
stem1<s1>: A s1 U | G s1 C | U s1 A | C s1 G
stem2<s1>: stem1<stem1<s1>>
stem<s1>:
    A stem<s1> U
    | U stem<s1> A
    | C stem<s1> G
    | G stem<s1> C
    | stem1<stem2<s1>>
```

References

1. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press (1996)
2. Hopcroft, J.E.: Introduction to automata theory, languages, and computation. Pearson Education India (2008)
3. Kasami, T.: An efficient recognition and syntax-analysis algorithm for context-free languages. Coordinated Science Laboratory Report no. R-257 (1966)
4. Knudsen, B., Hein, J.: Rna secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics (Oxford, England) **15**(6), 446–454 (1999)
5. Liu, T., Schmidt, B.: Parallel rna secondary structure prediction using stochastic context-free grammars. Concurrency and Computation: Practice and Experience **17**(14), 1669–1685 (2005)
6. Okhotin, A.: Parsing by matrix multiplication generalized to boolean grammars. Theor. Comput. Sci. **516**, 101–120 (Jan 2014), http://dx.doi.org/10.1016/j.tcs.2013.09.011
7. Valiant, L.G.: General context-free recognition in less than cubic time. J. Comput. Syst. Sci. **10**(2), 308–315 (Apr 1975), http://dx.doi.org/10.1016/S0022-0000(75)80046-8
8. Zier-Vogel, R., Domaratzki, M.: Rna pseudoknot prediction through stochastic conjunctive grammars. Computability in Europe 2013. Informal Proceedings pp. 80–89 (2013)

Table 1.

N	Gramma $D2$				Grammar BIO			
	valCPU	yavCPU	valGPU	yavGPU	valCPU	yavCPU	valGPU	yavGPU
127	78	76	195	105	1345	1339	193	106
255	289	292	523	130	5408	5488	525	140
511	1212	1177	1909	250	21969	22347	1994	256
1023	4858	4779	7878	540	88698	90318	7890	598
2047	19613	19379	33508	1500	363324	374204	34010	1701
4095	78361	78279	140473	4453	1467675	1480594	141104	5472
8191	315677	315088	-	13650	-	-	-	18039



(a) (b)

Fig. 6. (grammar $D2$).

Table 2.

s	N	adpCPU	adpGPU
250	1023	2996	242
	2047	6647	255
	4095	13825	320
	8191	28904	456
510	2047	12178	583
	4095	26576	653
	8191	56703	884
1020	4095	48314	1590
	8191	108382	1953
2040	4095	197324	5100