MODIFICATION OF VALIANT'S PARSING ALGORITHM FOR SUBSEQUENCES HANDLING

Yuliya Susanina[(1),(2)], Anna Yaveyn[(1)], Semyon Grigorev[(1),(2)]

(1) Saint Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, 199034 Russia
jsusanina@gmail.com, anya.yaveyn@yandex.ru

(2) JetBrains Research, Universitetskaya emb., 7-9-11/5A, St.Petersburg, Russia
semen.grigorev@jetbrains.com

Keywords: Context-free grammar, string-matching, Valiant's algorithm, secondary structure, parsing.

Abstract. The string-matching problem can be reduced to parsing—verification if some subsequence can be derived in this grammar. Bioinformatics requires working with a large amount of data, so it is necessary to improve the existing parsing techniques. 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 Valiants algorithm modification, which main advantage is the possibility to divide the parsing table into successively computed layers of disjoint submatrices where each submatrix of the layer can be processed independently. Moreover the modified version decreases a large amount of excessive computations and accelerates the substrings searching.

1 Introduction

Secondary structure of genomic sequences prediction plays important role in classification and recognition problems. It comes from the idea that secondary structure is a powerful source of information related with different biological functions of various organisms.

Some approaches connected with secondary structure analysis based on using formal grammars, as they are successful in modeling strings with correlated symbols [1, 2]. That means the specific features of secondary structure can be described by some context-free grammar (CFG) and the prediction problem can be reduced to parsing—verification if some sequence can be derived in this grammar. But checking the derivability is not frequently the main problem, sometimes all the derivable subsequences must be found [3].

The main disadvantage of CFG-based approaches is considerable problems with computational complexity. Traditional parsing method which is used in these approaches is CYK [4, 5] with cubic-time complexity, but this algorithm demonstrates poor performance on long strings or big grammars [6]. And so, as such field of application as bioinformatics requires working with a large amount of data, it is necessary to find more efficient parsing algorithms.

Still asymptotically most efficient parsing algorithm is based on matrix multiplication Valiant's algorithm [7]. Moreover, Okhotin generalized this algorithm to conjunctive and Boolean grammars which are the natural extensions of CFG with more expressive power [8]. For example, it is possible to express pseudoknots by using conjunctive grammars [9], while it is impossible by using context-free one. Valiants algorithm allows to simply utilize parallel techniques to improve performance by offloading critical computations onto matrices multiplication. However, this algorithm is not appropriate for finding substrings problem.

In this paper we present the modification of Valiant's algorithm, which increase the power of using GPGPU and parallel computations by computing some matrices products concurrently. Also proposed algorithm can be easily adopted for the string-matching, or substring finding, problem.

2 Formal languges

An alphabet Σ is a finite nonempty set of symbols. Σ^* is a set of all finite strings over Σ. A contex-free grammar G 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 $A \in N, \beta \in V^*, V = \Sigma \cup N$ and $S \in N$ is a start symbol. Context-free grammar $G = (\Sigma, N, R, S)$ is said to be in Chomsky normal form if all productions in R are of the form: $A \to BC$, $A \to a$, $S \to \varepsilon$, where $A, B, C \in N, a \in \Sigma, \varepsilon$ is an empty string. $L_G(A) = \{\omega | A \xrightarrow[G_A]{*} \omega\}$ is a language specified by the grammar $G_A = (\Sigma, N, R, A)$, where $A \xrightarrow[G_A]{*} \omega$ means that ω can be derived in a finite number of rules applications from the start symbol A.

2.1 *Valiant's parsing algorithm*

Main idea of all tabular parsing methods is to construct a parsing table T of size $(n+1) \times (n+1)$ for an input string $a_1 a_2 \dots a_n$ and context-free grammar $G = (\Sigma, N, R, S)$ which is in Chomsky normal form, where $T_{i,j} = \{A | A \in N, a_{i+1} \dots a_j \in L_G(A)\} \quad \forall i < j$.

The elements of T are filled successively beginning 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 $L_G(S)$ if and only if $S \in T_{0,n}$.

If all elements 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. As the most time-consuming is computing $\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$, Valiant rearranged computation of $T_{i,j}$, in order to use multiplication of submatrices of T.

Firstly, define multiplication of two submatrices of parsing table T. Let $X \in (2^N)^{m \times l}$ and $Y \in (2^N)^{l \times n}$ be two submatrices of 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 thta computation of $X \times Y$ can be replaced by multiplication of $|N|^2$ Boolean matrices (for each nonterminal pair). Denote matrix corresponding to pair $(B, C) \in N \times N$ as $Z^{(B,C)}$, then $Z^{(B,C)}_{i,j} = 1$ if and only if $(B, C) \in Z_{i,j}$. 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 $O(|G| BMM(n) log(n))$ for an input string of length n, where $BMM(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$.

Valiant's algorithm is written in the terms 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 $l \le i < j < m$.

The procedure $complete(l, m, l', m')$ constructs the submatrix $T_{i,j}$ for all $l \le i < m$, $l' \le j < m'$. This procedure assumes $T_{i,j}$ for all $l \le i < j < m, l' \le i < j < m'$ are already constructed and the current value of $P[i, j] = \{(B, C) | \exists k, (m \le k < l'), a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$ for all $l \le i < m, l' \le j < m'$. The submatrix division during the procedure call is shown in figure 2.

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 is a power of two

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

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

9 complete(*l, m,* l'*,* m'):
10 **if** $m - l = 4$ *and* $m = l'$ **then**
11 $T_{l,l+1} = \{A | A \to a_{l+1} \in R\}$;
12 **else if** $m - l = 1$ *and* $m < l'$ **then**
13 $T_{l,l'} = f(P_{l,l'})$;
14 **else if** $m - l > 1$ **then**
15 $leftgrounded = (l, \frac{l+m}{2}, \frac{l+m}{2}, m), rightgrounded = (l', \frac{l'+m'}{2}, \frac{l'+m'}{2}, m')$,
16 $bottom = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2}), left = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$,
17 $right = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m'), top = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$;
18 complete(bottom);
19 $P_{left} = P_{left} \cup (T_{leftgrounded} \times T_{bottom})$;
20 complete(left);
21 $P_{right} = P_{right} \cup (T_{bottom} \times T_{rightgrounded})$;
22 complete(right);
23 $P_{top} = P_{top} \cup (T_{leftgrounded} \times T_{right})$;
24 $P_{top} = P_{top} \cup (T_{left} \times T_{rightgrounded})$;
25 complete(top)

A simple example of Valiant's algorithm is presented in figure 3. Only the small number of first steps is shown, because later we point out at this version and our approach differences.

3 Modified Valiant's algorithm

In this section we describe the reorganization of submatrix processing order in the Valiant's algorithm which simplify independent handling of submatrices. As a result, proposed modification can facilitate implementation of parallel submatrix processing.

3.1 *Layered submatrices processing*

The main change of this modification is the possibility to divide the parsing table into layers of disjoint submatrices of the same size. The idea of division we have made from the reorganization of the matrix multiplication order is presented in figure 2. Each layer consists of square matrices which size is power of 2. The layers are computed successively in the bottom-up order. Each matrix in the layer can be handled independently, which can help to implement parallel version of layer processing function.

A simple example of the modification is shown in figure 4. The lowest layer (submatrices which size is 1) is already computed and filling of the matrix starts with the second layer (subfigures 1-2). Note that the same process is presented in figure 3, but here it can be done only in 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 divide the table into layers, described

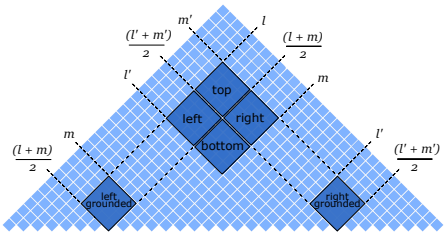

Figure 1: Matrix partition used in *complete(l, m, l', m')* procedure.

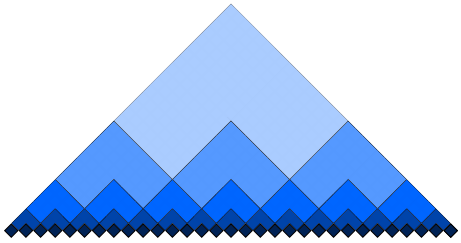

Figure 2: Matrix partition on V-shaped layers used in modification.

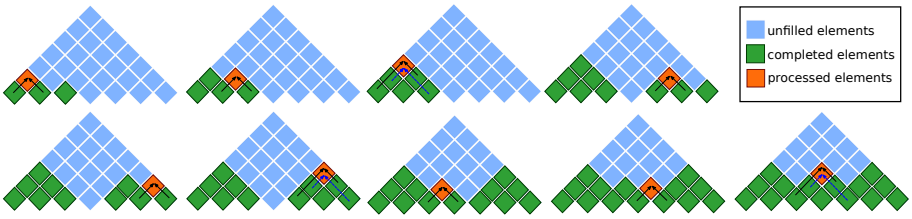

Figure 3: An example of beginning of Valiant's algorithm

earlier, and computes them through the *completeVLayer()* call. Thus, *main()* computes all elements of parsing table T. (Hereinafter, we use layer to mean set of submatrices.)

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

Also denote some subsidiary functions for matrix layer M:

- *bottomsublayer(M)* $= \{bottom(subm) \mid subm \in M\}$,
- *leftsublayer(M)* $= \{$*left(subm)* $\mid subm \in M\}$,
- *rightsublayer(M)* $= \{$*right(subm)* $\mid subm \in M\}$,
- *topsublayer(M)* $= \{top(subm) \mid subm \in M\}$.

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

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 same assumptions on $T_{i,j}$ and $P_{i,j}$ as in the previous case.

In the other words, *completeVLayer(M)* computes the entire layer *M* and *completeLayer(*M_2*)* is a support 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 a triple of submatrices, perform basic step of algorithm: matrix multiplication. It is worth mentioning that, as distinct from the original algorithm, here $|tasks| \ge 1$ and each task can be computed independently. So, practical implementation of this procedure can easily involve different techniques of parallel array processing, such as OpenMP **??**.

3.2 *Algorithm for substrings*

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

So if we want 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 $2^{l'-2} < s \le 2^{l'-1}$.

Let $l' = p - (m - 2)$ and consequently $(m - 2) = p - l'$.

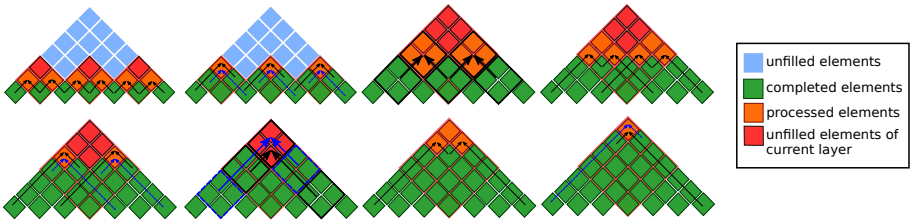

Figure 4: An example of the modification of Valiant's algorithm

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):

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

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(bottomsublayer(M));
    completeVLayer(M)

comleteVLayer(M):

multiplicationTasks$_1$ =
$\{left(subm), leftgrounded(subm), bottom(subm) \,|\, subm \in M\} \cup$
$\{right(subm), bottom(subm), rightgrounded(subm) \,|\, subm \in M\}$;

multiplicationTask$_2$ = $\{top(subm), leftgrounded(subm), right(subm) \,|\, subm \in M\}$;

multiplicationTask$_3$ = $\{top(subm), left(subm), rightgrounded \,|\, subm \in M\}$;

performMultiplications(multiplicationTask$_1$*)*;
completeLayer(leftsublayer(M) $\cup$ *rightsublayer(M))*;
performMultiplications(multiplicationTask$_2$*)*;
performMultiplications(multiplicationTask$_3$*)*;
completeLayer(topsublayer(M))

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

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.

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

Thus, time complexity for searching all substrings is $O(|G|BMM(2^{l'})(l'-1))$, while time complexity for the full input string is $O(|G|BMM(2^p)(p-1))$. In contract to the modification, Valiant's algorithm completely calculate at least 2 triangle submatrices of size $\frac{n}{2}$ (as shown in figure 5) which mean minimum asymptotic complexity $O(|G|BMM(2^{p-1})(p-2))$. Make a conclusion that the modification is asymptotically faster for substrings of size $s \ll n$ than the original algorithm.

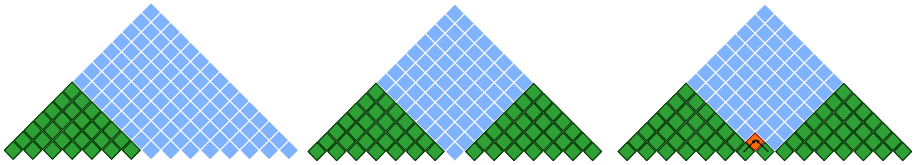

Figure 5: The number of elements necessary to compute in Valiant's algorithm. That means it is nessesary to calculate at least 2 triangle submatrices of size $\frac{n}{2}$.

4 Conclusion

In this paper we presented the modification of Valiant's algorithm, which made it possible to use parallel computations more effectively. Also we show its applicability for the string-matching problem the original version could not deal with.

The priority direction for our future research are high-performance implementation using GPGPU or other parallel techniques and its evaluation on real-world data.

Acknowledgments

The research was supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

References

[1] B. Knudsen and J. Hein, "Rna secondary structure prediction using stochastic context-free grammars and evolutionary history.," *Bioinformatics (Oxford, England)*, vol. 15, no. 6, pp. 446–454, 1999.

[2] R. D. Dowell and S. R. Eddy, "Evaluation of several lightweight stochastic context-free grammars for rna secondary structure prediction," *BMC bioinformatics*, vol. 5, no. 1, p. 71, 2004.

[3] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis*. Cambridge University Press, 1996.

[4] T. Kasami, "An efficient recognition and syntax-analysis algorithm for context-free languages," *Coordinated Science Laboratory Report no. R-257*, 1966.

[5] D. H. Younger, "Context-free language processing in time n3," in *Proceedings of the 7th Annual Symposium on Switching and Automata Theory (Swat 1966)*, SWAT '66, (Washington, DC, USA), pp. 7–20, IEEE Computer Society, 1966.

[6] T. Liu and B. Schmidt, "Parallel rna secondary structure prediction using stochastic context-free grammars," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 14, pp. 1669–1685, 2005.

[7] L. G. Valiant, "General context-free recognition in less than cubic time," *J. Comput. Syst. Sci.*, vol. 10, pp. 308–315, Apr. 1975.

[8] A. Okhotin, "Parsing by matrix multiplication generalized to boolean grammars," *Theor. Comput. Sci.*, vol. 516, pp. 101–120, Jan. 2014.

[9] R. Zier-Vogel and M. Domaratzki, "Rna pseudoknot prediction through stochastic conjunctive grammars," *Computability in Europe 2013. Informal Proceedings*, pp. 80–89, 2013.