AUTHOR'S GUIDELINES FOR CIBB PAPERS

First Author(1), Second Author(2)

(1) First Institute
Affiliation, address, email

(2) Second Institute
Affiliation, address, email

Keywords: keyword1, keyword2, keyword3, keyword4, keyword5.

Abstract. Abstract is abstract.

1 Scientific Background

1.1 Introduction

Since the theory of context-free grammars was developed by Noam Chomsky, it has been extensively studied [**?**, **?**]. The classic application of context-free grammars is describing natural and programming languages. Recent research has shown that the theory of formal languages and, in particular, context-free languages can be used in bioinformatics [**?**, **?**, **?**, **?**].

A good example of this usage is the recognition and classification problems in bioinformatics, some of them are based on the research claiming that the secondary structure of the DNA and RNA nucleotide sequence contains important information about the organism species. The specific features of the secondary structure can be described by some context-free grammar, and therefore the recognition problem can be reduced to parsing—verification if some nucleotide sequence can be derived in this grammar.

Such field of application as bioinformatics requires working with a large amount of data, so it is necessary to find highly efficient parsing algorithm. Moreover, this algorithm needs to be easily adaptive to such computing techniques as GPGPU (General-Purpose computing on Graphics Processing Units) or CPU parallel computing which is now a fairly widespread method to accelerate the computation.

The majority of parsing algorithms either has the cubic-time complexity (Kasami [**?**], Younger [**?**], Earley [**?**]) or could work only with sub-classes of context-free grammars (Bernardy, Claussen [**?**]), but still asymptotically more efficient parsing algorithm that can be applied to any context-free grammar is algorithm based on matrix multiplication proposed by Leslie Valiant [**?**]. It computes the parsing table for a linear input, where each element of this table is responsible for deriving a particular substring. By offloading the most time-consuming computations on a Boolean matrix multiplication, Valiant has achieved an improvement in time complexity, which is O(BMM(n)log(n)) for an input string of size n, where BMM(n) is the number of operations needed to multiply two Boolean matrices of size $n \times n$. In addition, Okhotin generalized the matrix-based algorithm to conjunctive and Boolean grammars which are the natural extensions of context-free grammars with more expressive power and also improved its performance and understandability [**?**]. In spite of the fact that Valiant's algorithm allows us to use parallel techniques, for example, compute matrix products on GPUs, it seems like a large part of matrix multiplications can be performed concurrently.

In this paper, we show how to reorganize the matrix multiplication order in Valiant's algorithm to divide the parsing table into successively computed layers of disjoint submatrices where each submatrix of the layer can be processed independently.

We make the following contributions:

- We propose the modification of Valiant's algorithm which allows to compute some matrix products concurrently and improve the performance through parallel techniques.
- We prove the correctness of the modification and provide its time complexity estimation which 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 $n \times n$.
- We show the applicability of our approach in bioinformatics research, especially in addressing the string-matching problem.
- We have implemented the proposed algorithm and our evaluation shows that ... parallel techniques improve the performance...

1.2 Preliminaries

An alphabet Σ is a finite nonempty set of symbols. Σ^* is a set of all finite strings over Σ. A grammar 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 $\alpha \to \gamma$, where $\alpha \in V^*NV^*$, $\gamma \in V^*$, $V = \Sigma \cup N$ and $S \in N$ is a start symbol.

Grammar $G = (\Sigma, N, R, S)$ is called context-free, if $\forall r \in R$ are of the form $A \to \beta$, where $A \in N, \beta \in V^*$.

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.

$L_G(A)$ is a language of the grammar $G_A = (\Sigma, N, R, A)$, which means all the sentences that can be derived in a finite number of rules applications from the start symbol A.

1.3 Parsing by matrix multiplication

The main problem of parsing is to verify if the input string belongs to the language of some given grammar G. We will describe the Cocke-Younger-Kasami algorithm and the most asymptotically efficient parsing algorithm, which works for all context-free grammars, Valiant's parsing algorithm, based on matrix multiplication. In this paper we use the rewritten version of Valiant's algorithm proposed by Alexander Okhotin.

The CYK algorithm is a basic parsing algorithm. Its main idea 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

$$P_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$$

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 ∈ T_{0,n}

4  compute(l, m):
5  if m − l ≥ 4 then
6      compute(l, (l+m)/2);
7      compute((l+m)/2, m)
8  complete(l, (l+m)/2, (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 → a_{l+1} ∈ 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, (l+m)/2, (l+m)/2, m), rightgrounded = (l', (l'+m')/2, (l'+m')/2, m'),
16     bottom = ((l+m)/2, m, l', (l'+m')/2), left = (l, (l+m)/2, l', (l'+m')/2),
17     right = ((l+m)/2, m, (l'+m')/2, m'), top = (l, (l+m)/2, (l'+m')/2, m');
18     complete(bottom);
19     P_left = P_left ∪ (T_leftgrounded × T_bottom);
20     complete(left);
21     P_right = P_right ∪ (T_bottom × T_rightgrounded);
22     complete(right);
23     P_top = P_top ∪ (T_leftgrounded × T_right);
24     P_top = P_top ∪ (T_left × T_rightgrounded);
25     complete(top)
```

$$f(P) = \{A | \exists A \to BC \in R : (B, C) \in P\}$$

The input string $a_1 a_2 \dots a_n$ belongs to $L_G(S)$ if and only if $S \in T_{0,n}$.

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.

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 $Z_{i,j} = \bigcup_{k=1}^{l} X_{i,k} \times Y_{k,j}$.

In listing 1 full pseudo-code of Valiant's algorithm is written in the terms proposed by Okhotin, is presented. 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.

In figure 1 is presented a simple example of Valiant's algorithm. Only the beginning of the work is shown, because later we point out at this version and our approach differences.

The procedure $compute(l, m)$ constructs the 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 $P[i, j] = \{(B, C) | \exists k, (m \leq k <$

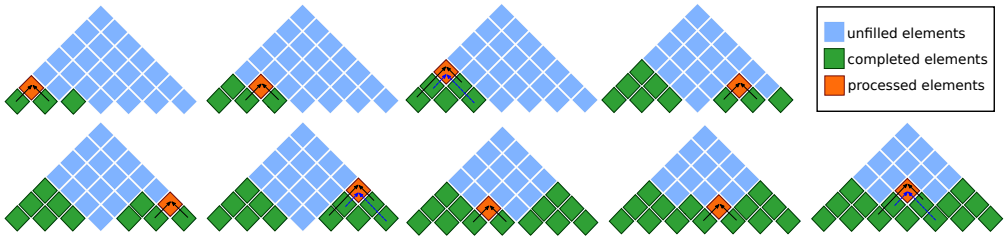

Figure 1: An example of beginning of Valiant's algorithm

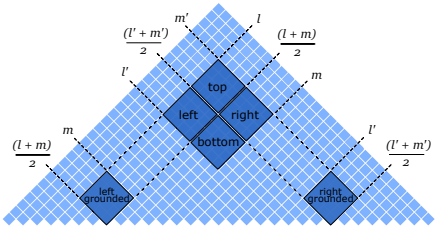

Figure 2: Matrix partition used in *complete(l, m, l', m')* procedure.

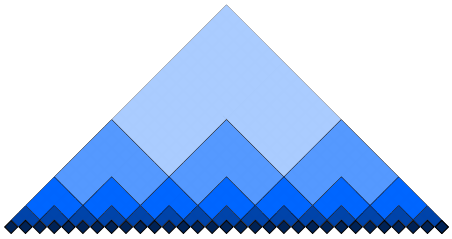

Figure 3: Matrix partition on V-shaped layers.

$l'), a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$ for all $l \leq i < m, l' \leq j < m'$. The submatrix division during the procedure call is shown in figure 2.

Then Valiant described that product of multiplying of two submatrices of parsing table T can be provided as $|N|^2$ Boolean matrices (for each pair of nonterminals). 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 $Z^{(B,C)} = X^B \times Y^C$. So, matrix multiplication in definition 25 can be replaced by Boolean matrix multiplication, each of which can be computed independently. Following these changes, time complexity of algorithm in listing 1 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 $n \times n$.

2 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.

2.1 New approach

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 3. 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 (sub-

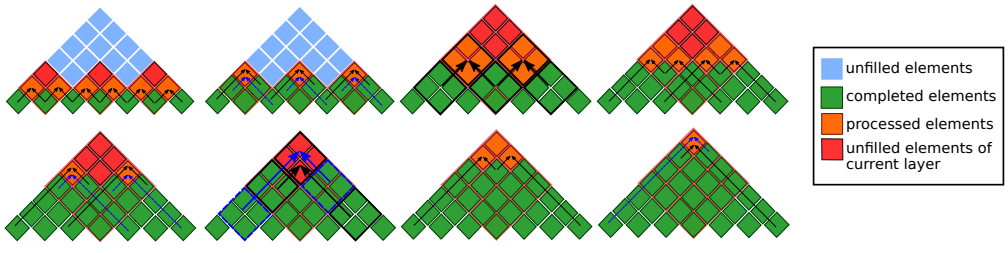

Figure 4: An example of the modification of Valiant's algorithm

matrices 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 1, 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 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.)

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 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)* $= \{bottom(subm) \,|\, subm \in M\}$,
- *leftsublayer(M)* $= \{$*left(subm)* $\,|\, subm \in M\}$,
- *rightsublayer(M)* $= \{$*right(subm)* $\,|\, subm \in M\}$,

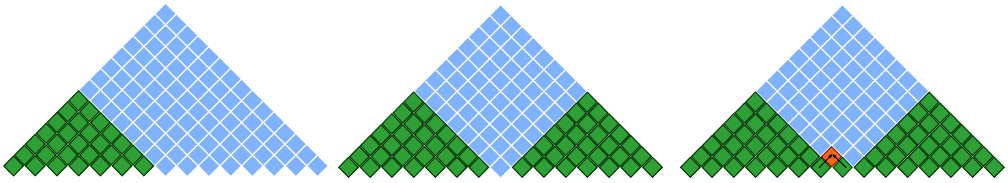

Figure 5: The number of elements necessary to compute in Valiant's algorithm. That means it is nessesary to calculate at least 2 triangle submatrices of size $\frac{n}{2}$.

- *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 $l \leq i < j < m$ 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 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 $|tasks| \geq 1$ 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 **??**.

2.2 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 $2^{l'-2} < s \leq 2^{l'-1}$.

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.

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

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.

2.3 Evaluation

After demonstrating the theoretical value of the proposed solution, the next step of our work was to show its applicability on real data. Both algorithms (original Valiant's version and the modification) were compared on context-free grammar G ...?picture?...

which is used to approximate the secondary structure of the biological sequences. As it was mentioned before, the secondary structure is a very powerful instrument for species classification and identification problem. Parsing algorithms based on matrix multiplication helps efficiently find subsequences with features specific to the secondary structure.

The algorithms were implemented using a library for fast Boolean matrix multiplication M4RI [**?**]. The biological sequences were taken from this dataset[].

All tests were run on a PC with the following characteristics:

- OS:
- CPU:
- System Type:
- RAM:

The results of experiments which are presented ?...? show that our modification can be efficiently applied to the string matching problem as it demonstrates good time on real data.

3 Conclusion

Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work. Conclusion is future work

Acknowledgments

Example of the Acknowledgments section.

References