Parallel Approach to Context Transformations

Michal Vašinek, Jan Platoš

Department of Computer Science, FEECS, VŠB-Technical University of Ostrava,
17.listopadu, 708 33, Ostrava - Poruba,
michal.vasinek@vsb.cz,jan.platos@vsb.cz,

Abstract. Context transformation is a process that turns input data into one with lower zero order entropy. The main weakness of algorithms presented sofar is the complexity of replacing procedure. In this paper we describe properties related to the parallelization of replacing procedure and we propose a modified version of a basic context transformation algorithm, that uses a parallel approach.

Keywords: compression, context, transformation, entropy, parallel computing

1 Introduction

There are two main classes of algorithms employed in the data compression. The first class of algorithms deals directly with the compression and their purpose is to decrease the size of the input message. Examples [3] of such algorithms are Huffman coding, Arithmetic coding, Lempel Ziv algorithms family, PPM and many others. The second class of algorithms behaves more like preprocessors for the first class, these algorithms are usually called transformations, examples are Burrows-Wheeler transformation [1] or MoveToFront [2] transformation that are used in bzip2 file compressor.

The purpose of transformations is not to decrease message size but to change the internal message structure that could be more easily handled by some of the first class algorithms. In [5] and [6] we propose a reversible transformation method called a 'Context transformation', that we use to reduce zero-order entropy of input messages. Transformed data are then compressed using entropy coding algorithms, like Huffman coding. In this paper we describe properties related to the parallelization of context transformation algorithms.

We use several notation conventions: we use Σ to denote set of message symbols, characters in equations are greek symbols from Σ and unless stated otherwise, they can represent any member of Σ, i.e. $\alpha = \beta$ as well as $\alpha \neq \beta$ it should be clear from the context. When we present examples of transformations we use symbols from english alphabet to denote particular transformations like $ab \rightarrow ac$, then each character $a, b, c, \ldots$ are distinct characters $a \neq b \neq c \neq \ldots$.

The rest of the paper is organized as follows. Section 2 contains description of the proposed Context transformations and their properties. Section 3 analyses complexity of the transformation with respect to the number of symbols in the alphabet for both proposed transformation types. Section 4 describes the ability of the transformations to work in parallel. Section 5 contains design of the parallel algorithm and presents results achieved on the selected dataset. Last Section 6 concludes the paper and discusses achieved results.

2 Context Transformations

The context transformation is a method that takes two digrams, the digram $\alpha\beta$ that is present in the input message and the digram $\alpha\gamma$ that is not present in the input message. Transformation replaces all occurences of $\alpha\beta$ for $\alpha\gamma$. The symbol α is called a context symbol and it is present in both digrams.

Definition 1 *Context transformation(CT) is a mapping $CT(\alpha\beta \rightarrow \alpha\gamma, w) : \Sigma^n \rightarrow \Sigma^n$, Σ is the alphabet of the input message w and n is the length of the input message, that replaces all digrams $\alpha\beta$ for digram $\alpha\gamma$, where $p(\alpha, \gamma) = 0$ and $\beta \neq \gamma$.*

We may consider also the transformation of the form $\beta\alpha \rightarrow \gamma\alpha$, such transformation would correspond to the same transformation like in Definition 1 if we perform replacement on the mirror message. There is one rule that must be followed and it is that such transformation must be applied on the input message from right to left, respectively from left to right in the case of $\beta\alpha \rightarrow \gamma\alpha$. This rule ensures that for each context transformation an inverse transformation that transforms the new message back to the input message exists. The proof of this rule can be found in [5] and [7]. Context transformations are a subset of more general set of generalized context transformations:

Definition 2 *Generalized context transformation(GCT) is a mapping $GCT(\alpha\beta \leftrightarrow \alpha\gamma, w) : \Sigma^n \rightarrow \Sigma^n$, Σ is the alphabet of the input message w and n is the length of the input message, that exchanges all digrams $\alpha\beta$ for digram $\alpha\gamma$ and vice-versa.*

GCT transformation can be applied in any direction, but $CT \subset GCT$ only when they are applied in the same direction. In this paper we describe GCT transformations applied from right so it is consistent with the former notion of context transformations.

The main difference between a context and a generalized context transformation is that we can swap any two digrams beginning with *alpha*, hence we are no more restricted on cases when one of digrams is not present in a message. Example of each transformation is presented in Fig. 1.

The following paragraphs contains brief discussion of the context transformation process and its implication to zero order Shanonn entropy [4]. The reader can find the detailed description in [5] and [7]. When we will speak about entropy we mean the Shannon's entropy over alphabet Σ of individual symbols defined by:

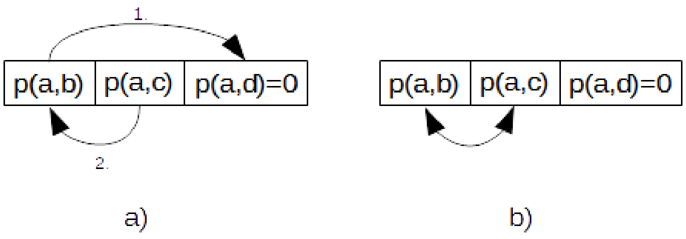

Fig. 1. Examples of two different types of transformations, $p(b) > p(c) > p(d)$, $p(a,c) > p(a,b)$, $p(b,c) > p(a,b)$: a) Sequence of context transformations $ab \rightarrow ad, ac \rightarrow ab$, b) Generalized context transformation $ab \leftrightarrow ac$

$$H = -\sum_{x \in \Sigma} p(x) log(p(x)) \tag{1}$$

where $p(x)$ is probability of symbol x. Context transformations are based on the fact that the structured data, like a human-written texts or programming codes has its inner structure and the most of the information is not hidden in the entropy of input data alphabet but it depends more on occurences of digrams, trigrams, words, ...

Suppose example string 'kakaoo', frequencies of its digrams can be represented by a matrix, we call such matrix a 'Context matrix'. Entry of a context matrix is non-negative integer that represents frequency of digram represented by row symbol followed by column symbol. For our example string, the context matrix is shown in Table 1.

	k	a	o
k	0	2	0
a	1	0	1
o	0	0	1

Table 1. The context matrix that represents the string 'kakaoo'

Since the probability distribution of symbols is uniform, the entropy of our example string is for given alphabet maximal. From context matrix we see that there are several accessible transformations, first we select a context transformation that replaces all digrams 'ka' for a digram 'kk', so the condition of zero and non-zero entry is fulfilled.

The resulted string is 'kkkkoo' and its context matrix is shown in Table 2. We can see two different consequences of this transformation:

– the alphabet size decreased,

– the entropy decreased.

	k	a	o
k	3	0	1
a	0	0	0
o	0	0	1

Table 2. The context matrix that represents the transformed string 'kkkkoo'

When we use a generalized context transformation instead of a simple context transformation we arrive in two different strings based on which transformation direction was applied. When GCT is applied from left then $GCT_{\rightarrow}(w) = kkaaoo$ and from right like in a context transformation case $GCT_{\leftarrow}(w) = kkkkoo$.

We may select another transformation, for example 'oo' for 'ok', leaving w in the state 'kkkkok', that has even lower entropy. What we examine in this paper is, if we can run these two transformations simultaneously, respectively under what conditions these can be perfomerd in parallel.

2.1 Transformations and their consequences on Huffman coding

Compression schema based on CT or GCT consist of two phases, in the first phase the input message is transformed using (G)CT, and finaly in the second phase, the message is compressed using some zero-order entropy coder.

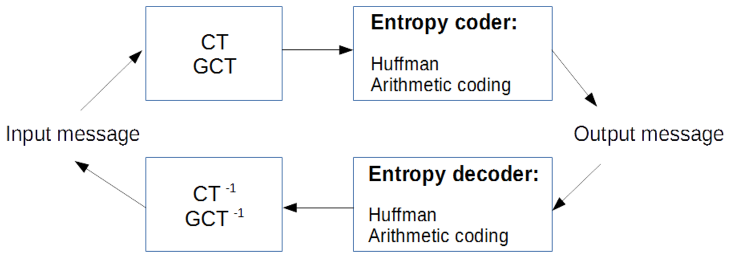

Fig. 2. Compession schema based on context transformations.

Suppose static Huffman coding as a zero-order entropy coder, if $p(\beta) > p(\gamma)$ then for lengths of their corresponding Huffman codes holds that $|c(\beta)| \leq |c(\gamma)|$. If $p(\alpha, \gamma) \neq 0$ and $p(\alpha, \beta) = 0$ (resp. $p(\alpha, \beta) < p(\alpha, \gamma)$) and transformation $CT(\alpha\gamma \rightarrow \alpha\beta)$ (resp. $GCT(\alpha\beta \leftrightarrow \alpha\gamma)$) is applied, then all symbols γ that are part of digrams $\alpha\gamma$ will be coded by the code of length $|c(\beta)|$ instead of the code of length $|c(\gamma)|$.

3 Complexity analysis

Presented algorithms are all greedy approaches, algorithms iteratively search for the best entropy reducing transformations and then apply transformation on the source. Formally we divide each iteration in two steps:

1. Search procedure - find the best transformation according the actual context matrix,
2. Replacing procedure - apply selected transformation on the input string w.

```
n ← input size
w ← input string
compute_matrix()
**repeat**
   transformation = search_procedure()
   H0 = entropy()
   H1 = entropy(transformation)
   **if** $n * (H0 - H1) < LIMIT$ **then**
      **return**
   **end if**
   replacing_procedure(transformation,w)
**until true**
```

Fig. 3. Basic structure of context transformation algorithms

The infinite loop in Fig. 3 terminates when there are no more transformations that are able to reduce entropy more, than by a limit given by $LIMIT$ variable. The $LIMIT$ variable is set up to the size of transformation description. In our algorithms $LIMIT = 24$ because we store description of each transformation as a triplet of byte values $\alpha\beta\gamma$.

Both parts of the algorithm can be parallelized, but the operation that is the most time consuming is the replacing procedure as is depicted in Table 3. In the Section 5 we present a modified version of the basic algorithm, that uses the fact, that some transformations can be performed simultaneously and we try to parallelize the replacing procedure.

Search(ms)	Replacement(ms)
1.144	28.813

Table 3. Average time per file needed for search and replacement procedures. Dataset assasin(6097 files).

In the next sections we analyse complexities of both parts and we show that for source string w of length $|w| \to \infty$ the only significant part remains to be replacing procedure.

3.1 Search procedure

Algorithm in Fig. 3 is a basic greedy algorithm that searches for and eventually performs context transformations in such a way that the resulted entropy of alphabet Σ is lower than in the untransformed case Search procedure operates on the context matrix, where rows and columns are sorted according their frequencies. The algorithm iterates through rows of the matrix from the most probable one to the least probable one.

The simplified version of the search procedure is presented in the following pseudocode:

```
$cm \leftarrow context\ matrix$
$\Delta H = 0 \leftarrow the\ change\ of\ zero\ order\ entropy$
$T \leftarrow searched\ transformation$
**for** $row = 1$ to $dim(cm)$ **do**
   **for** $col\_1 = 1$ to $dim(cm)$ **do**
      **for** $col\_2 = col\_1 + 1$ to $dim(cm)$ **do**
         $\Delta H_{temp} = compute\_dH(row, col\_1, col\_2)$
         **if** $\Delta H_{temp} - \Delta H < 0$ **then**
            $\Delta H = \Delta H_{temp};$
            $T = [row, col\_1, col\_2]$
         **end if**
      **end for**
   **end for**
**end for**
```

Fig. 4. Outline of the search procedure

The function *compute_dH* has constant complexity because it computes only change of entropy, probabilities will be modified only for symbols β and γ, so we don't need to recompute entropy of all symbols, but is is sufficient to recompute entropies of β and γ. The complexity of the search procedure is dependent only on the size of the alphabet and its worst case complexity is $O(t|\Sigma|^3)$, because we have to perform $1/2|\Sigma|^3$ searches. In our algorithm design the maximum allowable size of the alphabet is $|\Sigma| = 256$, since we interpret as a letter only a one byte values. That concludes that the upper limit of operations needed for one search is fixed and in the worst case it is given by $|\Sigma|^3$.

There are several techniques how the search procedure can be designed with lower complexity, i.e. precomputation and storage of n best transformations at the beginning and then only updating the modified ones, leads to the number of entropy recomputation given by $|\Sigma|^3 + 2t|\Sigma|^2$.

3.2 Replacing procedure

The second step, the replacing procedure is very simple, it passes data t times and exchanges both digrams. Let t be a number of performed GCTs and n is the length of the input, then the complexity of replacing procedure is $O(tn)$.

The inverse transformation consists only from replacing procedure and so it also has the complexity $O(tn)$. If the generalized context transformation is of the form $\alpha\beta \leftrightarrow \alpha\gamma$ then its inverse transformation is of the same form, but is taken in the opposite direction. The experimentally observed t was in range 100-1000 of transformations.

3.3 Comparison

When we let the complexitities functions to be equal we arrive at the limit when one computation of a replacement procedure becomes more significant than the one computation of a search procedure. We have to arrive at the limit because the search procedure is independent of the input size:

$$c_1 tn = c_2|\Sigma|^3 + c_3 t|\Sigma|^2 \tag{2}$$

Constants c_i represents implementation specific coefficients. The number of transformations t on both sides of the equation can be rearranged leaving us with:

$$n = \frac{c_2|\Sigma|^3 + c_3 t|\Sigma|^2}{c_1 t} = \frac{c_2|\Sigma|^3}{c_1 t} + \frac{c_3|\Sigma|^2}{c_1} \tag{3}$$

as a number of transformations $t \to \infty$ the first term on the right side becomes zero:

$$\lim_{t\to\infty} n = \frac{c_3|\Sigma|^2}{c_1} \tag{4}$$

when $|\Sigma|$ is finite, then for all source strings of length m, where $m > n$, the replacing procedure will be more computationally intensive than the search procedure.

$$c_1 tm > c_2|\Sigma|^3 + c_3 t|\Sigma|^2 \tag{5}$$

4 Parallel transformations

In this section we describe conditions needed for parallelization of the context and generalized context transformations.

4.1 Commuting transformations

Let's consider two different transformations T_1 and T_2, we say that these two transformations commute, if they satisfy following definition:

Definition 1. *Two different transformations T_1 and T_2 commute if:*

$$T_1(T_2(w)) = T_2(T_1(w)) \tag{6}$$

In our model example of the string $w = kakaoo$ the two presented transformations commute, since if T_1 is 'ko' to 'kk' transformation and T_2 is 'oo' to 'ok' transformation then $T_2(T_1(\text{`}kakaoo')) = T_1(T_2(\text{`}kakaoo')) = kkkkok$.

As an example of non-commuting transformations let's consider transformation T_1 again and transformation T_3 that replaces digrams 'ao' to 'aa'. Applying these two transformations in both ways will lead to different results. $T_3(T_1(w)) =' kkkkao'$ but $T_1(T_3(w)) =' kkkkoo'$ so we see that $T_1(T_3(w)) \neq T_3(T_1(w))$

4.2 Parallel transformations

Commutativity is not sufficient property to decide if two transformations could be run in parallel. As may be seen along with property that they have an inverse transformation to them.

Let's consider again our example word $w = kakaoo$ and two transformations $T_{4,}$ representing $ak \to aa$ and T_5 representing $ao \to aa$. These two transformations commute and transform together the word 'kakaoo' into the word 'kaaaao', but when we perform inverse transformation, we can get different results, since we don't know which of two inverse transformations will replace digram 'aa' first.

Before we show how to handle inverse transformations, we introduce two sets, an abstract set D_i and set K_i, that will be later used to prove a theorem about parallel transformations:

Definition 2. *Let D_i be a set of all unique digrams that are modified(created or destroyed) by transformation $T_i(\alpha\beta \leftrightarrow \alpha\gamma)$ and let the set K_i be a set of all transformation digrams $K_i = \{\alpha\beta, \alpha\gamma\}$.*

The set D_i contains digrams $\alpha\beta$, $\alpha\gamma$ and all digrams of type βX and γX, where $X \in \Sigma$. Suppose two sets D_1 and D_2 and let $D_1 \cap D_2 \neq \emptyset$, these two sets share at least one common digram d, suppose that $d \in K_1 \cup K_2$, it means that transformations T_1 and T_2 will interfere on this particular digram i.e. when transformation T_1 modifies digram $\alpha\beta$ on $\alpha\gamma$ then the second transformation won't be able to modify digram $\alpha\beta$ as it would do in the case when there is no other transformation T_1. From the above reasoning we form a lemma about parallel transformations:

Lemma 1. *Two transformations T_1 and T_2 can be run in parallel if:*

$$D_1 \cap D_2 = \emptyset \tag{7}$$

Proof. Since $D_1 \cap D_2 = \emptyset$ then no digrams that are modified by both transformations T_1 and T_2 exist and so these two transformations can be applied together. □

Lemma 1 gives us a simple condition that we use in our algorithms to construct a set of parallel transformations, but it is a weak condition, because there still exists parallel transformations, but $D_1 \cap D_2 \neq \emptyset$, i.e. suppose $T_1(ab \leftrightarrow ac)$ and $T_2(db \leftrightarrow dc)$, for these two transformations $D_1 \cap D_2 = \{bX, cX\}$.

Theorem 1. *Two transformations T_1 and T_2 can be run in parallel if:*

$$D_1 \cap K_2 = \emptyset \tag{8}$$

and

$$D_2 \cap K_1 = \emptyset \tag{9}$$

Proof. Suppose that transformation T_1 is of the form $T_1(\alpha\beta \leftrightarrow \alpha\gamma)$, it has correspoding sets $D_1 = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$ and $K_1 = \{\alpha\beta, \alpha\gamma\}$.

If T_2 contains in its K_2 one of the elements from D_1 then it means that the transformation T_1 can modify some of the elements that would be otherwise transformed(replaced) by T_2, so the two transformations can be run in parallel only if $D_1 \cap K_2 = \emptyset$.

Similar reasoning can be used to prove theorem for equation (9). If T_2 contains in its D_2 one of the elements from K_1, i.e. $\alpha\beta$ then if T_2 will change any occurence of $\alpha\beta$ first, the first transformation won't be able to modify it to $\alpha\gamma$, so such transformations cannot be parallelized and for parallel transformations must hold $D_2 \cap K_1 = \emptyset$.

Now suppose that $D_1 \cap K_2 = \emptyset$, but $D_2 \cap K_1 \neq \emptyset$, then some element i.e. $\alpha\beta$ is in the set D_2, we know that transformations are not parallel when $\alpha\beta \in K_1 \wedge \alpha\beta \in K_2$, now we prove that they cannot be parallelized also if $\alpha\beta \in D_2 \backslash K_2$, bacause then T_2 is of the form $X\alpha \leftrightarrow XY$, but when α is modified on Y then instead of $\alpha\beta$ will be $Y\beta$ and transformation T_1 cannot modify it. The same is valid in the case when $D_1 \cap K_2 \neq \emptyset$, but $D_2 \cap K_1 = \emptyset$. So both conditions in Theorem 1 must be valid together. □

Because our parallel algorithm is based on Lemma 1, we show several other properties that parallel transformations based on Lemma 1 have.

Theorem 2. *Two different transformations T_1 and T_2 can be run in parallel if:*

$$T_1(T_2(w)) = T_2(T_1(w)) = w_T \tag{10}$$

and

$$T_1^{-1}(T_2^{-1}(w_T)) = T_2^{-1}(T_1^{-1}(w_T)) = w \tag{11}$$

Proof. Suppose the transformation $T_{\leftarrow}(\alpha\beta \leftrightarrow \alpha\gamma)$, where the arrow at the index is a label for transformation direction i.e. from right to left $\leftarrow$; it has its coresponding set of modified digrams $D_{\leftarrow} = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$, next suppose the transformation $T_{\rightarrow}(\alpha\beta \leftrightarrow \alpha\gamma)$ and its coresponding set of digrams $D_{\rightarrow} = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$, we see that $D_{\leftarrow} = D_{\rightarrow}$, but we know that $T_{\rightarrow}(T_{\leftarrow}(w)) = w$, so transformations T and T^{-1} share the same set D.

If $T_1(T_2(w)) = T_2(T_1(w)) = w_T$ then it means that there are no interfering digrams, because if there would be such digrams then the equality would not hold, so $D_{\leftarrow,1} \cap D_{\leftarrow,2} = \emptyset$, but we showed that $D_{\leftarrow} = D_{\rightarrow}$ so also intersect of inverse transformations sets is empty and T_1^{-1} and T_2^{-1} will also recover w correctly. □

Lemma 1 and Theorem 2 describes the same phenomenom and they can be generalized for the arbitrary set of parallel transformations:

Theorem 3. *The set of transformations T can be run in parallel if for all pairs of transformations T_i and T_j holds that:*

$$D_i \cap D_j = \emptyset \tag{12}$$

Proof. Suppose the set T can be run in parallel, and suppose two transformations $T_i, T_j \in T$ such that $D_i \cap D_j \neq \emptyset$, then it means that there exist some digram $\alpha\beta$ common for T_i and T_j and these transformations cannot be run in parallel, but this is in contradiction with hypothesis that T is parallel, so the set $D_i \cap D_j = \emptyset$. □

There is one important fact to emphasize that emerged as a consequence of Theorem 2, it is a statement about inverse transformations:

Corollary 1. *Let $T = \{T_1, T_2, \dots, T_n\}$ is a set of parallel transformations, then the set $T^{-1} = \{T_1^{-1}, T_2^{-1}, \dots, T_n^{-1}\}$ is parallel as well.*

Proof. In Theorem 2 we proved that $D_i = D_i^{-1}$ and because for the set T holds that for all sets D_i, D_j is $D_i \cap D_j = \emptyset$, then also $D_i^{-1} \cap D_j^{-1} = \emptyset$ and the transformation set T^{-1} can be run in parallel.

Corollary 1 is very important result because it tells us that we may parallelize not only the set T but also its inverse T^{-1} so an inverse transformation algorithm is parallelizable as well.

With the knowledge of Lemma 1 we know how to construct set T, now we explore how large the set possibly can be for particular alphabet Σ:

Theorem 4. *The maximal size M_T of the set of parallel transformations T for particular alphabet Σ is:*

$$M = \lfloor \frac{|\Sigma|}{2} \rfloor \tag{13}$$

Proof. There are two basic types of the set D, one type coresponds to the transformation of the form $\alpha\alpha \leftrightarrow \alpha\beta$ and the second type coresponds to the transformation $\alpha\beta \leftrightarrow \alpha\gamma$. The first set $D_1 = \{\alpha\beta, \alpha\alpha, \alpha X, \beta X\} = \{\alpha X, \beta X\}$ influences two rows of the context matrix meanwhile the second set $D_2 = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$ influences three rows. So when only transformations of the first type are selected into T then at most $\lfloor |\Sigma|/2 \rfloor$ transformations can be run in parallel. □

Theorem 5. *Relation to be parallel between transformations is not transitive.*

Proof. Suppose a sets of digrams $D_{i\in\{0,1,2\}}$. Let $D_0 \cap D_1 = \emptyset$ and $D_1 \cap D_2 = \emptyset$, if transitivity holds then $D_0 \cap D_2 = \emptyset$, but this is not generally true. Consider following example, let $T_0(ab \leftrightarrow ac)$,$T_1(ad \leftrightarrow ae)$, transformations are clearly parallelizable. Let $T_2(ac \leftrightarrow af)$, $D_1 \cap D_2 = \emptyset$, but $D_0 \cap D_2 \neq \emptyset$. □

5 Parallel version of the basic context transformation algorithm

As we saw in the section about parallel transformations, there is a distinct number of rows affected by each transformation. This knowledge allows us, based on Lemma 1, collect parallel transformations in the first step and afterwards perform them simultaneously. Individual transformations are perfomed simultaneously in shared memory. The process is outlined in Fig.5.

```
$cm \leftarrow$ $context\ matrix$
$t\_array = [] \leftarrow parallel\ transformations$
$t\_size \leftarrow array\_size$
for $row = 1$ to $dim(cm)$ do
  for $col = 1$ to $dim(cm)$ do
    $i = 0$
    while $parallel\_transformation(t\_array, row, col)$ $and$ $i < t\_size$ do
      $append\_transformation(t\_array, get\_transformation())$
      $i = i + 1$
    end while
    for $transformation$ in $t\_array$ do
      $perform\_parallel(transformation)$
    end for{This section is run in parallel}
  end for
end for
```

Fig. 5. Parallel modification of basic transformation algorithm

We tested algorithm on the computer machine equipped by four proccessors. Parallel algorithm was more then three times faster than the serial one. The results are shown in Table 4.

Table 4. Average processing time per file. Dataset assasin.

Serial(ms)	Parallel(ms)	Speed-up
45.383	13.397	3.387

Serial and parallel algorithms can have different transformation paths(i.e. different transformations or the order in which transformations are performed).

In the serial version of the algorithm, there can be also performed transformations inaccessible to the parallel algorithm, so the resulted entropy is lower in the serial case as is presented in Table 5.

No transformation	Serial	Parallel
4.938	4.054	4.106

Table 5. Entropy comparison. Dataset assasin.

6 Conclusion

We showed a basic properties that have to be fulfilled to run replacing procedure of context transformation algorithms in parallel. The presented parallel version of the basic context transformation algorithm significantly increases the speed of processing of individual data files. On the other hand, usage of parallel algorithm can lead to minor increase of the resulted entropy.

There are two directions in which we would like to continue our research, the first direction is to prepare parallel algorithms according Theorem 1 instead of Lemma 1 and since algorithms presented sofar performs transformations using only digrams, in the future work we will also focus on development of algorithms operating on different context lengths.

Acknowledgment

This work was supported by the SGS in VSB - Technical University of Ostrava, Czech Republic, under the grant No. SP2015/146.

References

1. Burrows, M. and Wheeler D.J., *A block sorting lossless data compression algorithm*, 1994
2. Bentley, J, L. and Sleator, D. D. and Tarjan, R. E. and Wei, Victor K. *A locally adaptive data compression scheme* Commun. ACM, vol 4, pp. 320-330, 1986
3. Salomon, D., *Data Compression: The Complete Reference*, Springer, NewYork, 2007.
4. Shannon, C. E., *A mathematical theory of communication* Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, 1948.
5. Vašinek, M., *Kontextové mapy a jejich aplikace* Master Thesis, VŠB - Technical University of Ostrava, 2013
6. Vašinek, M. and Platoš, J., *Entropy reduction using context transformations* Data Compression Conference(DCC), pp. 431-431, 2014
7. Vašinek, M., *Context Transformations* Ph.D. Workshop of Faculty of Electrical Engineering and Computer Science, 2014