The object of research is the process of generating text data corpora using the CorDeGen method. The problem solved in this study is the insufficient efficiency of generating corpora of text data by the CorDeGen method according to the speed criterion.

Based on the analysis of the abstract CorDeGen method – the steps it consists of, the algorithm that implements it – the possibilities of its parallelization have been determined. As a result, two new modified methods of the base CorDeGen method were developed: "naive" parallel and parallel. These methods differ from each other in whether they preserve the order of terms in the generated texts compared to the texts generated by the base method ("naive" parallel does not preserve, parallel does). Using the .NET platform and the C# programming language, the software implementation of both proposed methods was performed in this work; a property-based testing methodology was used to validate both implementations.

The results of efficiency testing showed that for corpora of sufficiently large sizes, the use of parallel CorDeGen methods speeds up the generation time by 2 times, compared to the base method. The acceleration effect is explained precisely by the parallelization of the process of generating the next term – its creation, calculation of the number of occurrences of texts, and recording – which takes most of the time in the base method. This means that if it is necessary to generate sufficiently large corpora in a limited time, in practice it is reasonable to use the developed parallel methods of CorDeGen instead of the base one. The choice of a particular parallel method (naive or conventional) for a practical application depends on whether or not the ability to predict the order of terms in the generated texts is important

Keywords: natural language processing, CorDeGen method, text data corpora, corpora generation

UDC 004.021:004.91
DOI: 10.15587/1729-4061.2024.298670

ACCELERATING THE PROCESS OF TEXT DATA CORPORA GENERATION BY THE DETERMINISTIC METHOD

Yakiv Yusyn
Corresponding author
PhD*
E-mail: yusyn@pzks.fpm.kpi.ua
Tetiana Zabolotnia
PhD*
*Department of Computer Systems Software
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Beresteiskyi ave., 37, Kyiv, Ukraine, 03056

Received date 27.11.2023
Accepted date 14.02.2024
Published date 28.02.2024

***How to Cite**: Yusyn, Y., Zabolotnia, T. (2024). Accelerating the process of text data corpora generation by the deterministic method. Eastern-European Journal of Enterprise Technologies, 1 (2 (127)), 26–34. doi: https://doi.org/10.15587/1729-4061.2024.298670*

1. Introduction

Most problems in the field of natural language processing are related to the analysis and transformation of text data collected into corpora, for example: clustering, classification, training of language models, etc. In this case, the corpus is a set of selected, processed, and annotated texts in a certain way (according to the task) [1].

At the same time, the results of corpora processing can be both valuable in themselves and used only as an intermediate stage. For example, the results obtained on certain reference corpora can be used to compare and evaluate the effectiveness of natural language processing methods. Also, corpora are necessary when solving (for software implementations of natural language processing methods) pure software engineering problems: benchmarking implementations of different methods/implementations of one method, ensuring the quality of developed implementations, etc.

Historically, the first way to build corpora of textual data is manual. In this case, all texts for the corpus are selected, processed, and annotated by a person. However, with the growing need for corpora of different sizes, different thematic focus, intended for different tasks, this approach loses its relevance because the manual approach requires a lot of time and human effort. In addition, with the spread of the Internet and social networks, the amount of unstructured textual information that is generated by humankind and available for its organization into corpora has increased rapidly. Given this, another way to build corpora of textual data has emerged and developed, which is the generation of corpora based on natural unstructured texts or completely artificial generation. Owing to automatic generation, the process of building corpora is greatly simplified, and the time required for this is reduced but there is a need for methods and algorithms of generation. Such methods and algorithms can be both general-purpose and specialized – those that generate corpora for a specific purpose. Considering the wide variety of natural language processing tasks and the possible applications of corpora with them, the need for corpora generation methods does not diminish.

Therefore, research into devising new methods for generating (general and specialized purpose) text data corpora is relevant.

2. Literature review and problem statement

Many studies have been conducted on generating corpora using natural texts for various natural language processing tasks, such as [2–7].

In work [2], the authors describe the method of generating a corpus of texts in the Tunisian dialect of modern standard Arabic. In order to achieve this, it uses an existing corpus of Modern Standard Arabic and the mapping rules

that apply to that corpus. As a result, the authors designed a tool called Tunisian Dialect Translator. The generated corpus of the Tunisian dialect is expected to be used to solve other tasks of processing texts (and not only) written in this dialect, including training of machine learning models. In general, the described approach can be used to generate a corpus of texts of any dialect of any language. To this end, it is only necessary to have a corpus of texts in the original language and a set of rules for conversion. The authors do not provide any data on the performance of the developed method and the TDT software tool. This can be explained by the fact that during the experimental verification of the effectiveness of the developed method, the authors used a small amount of data (150 verbs and 89 sentences), which the ineffective method would also process quickly enough.

In work [3], the authors consider the problem of automatic generation of corpora for multidimensional intellectual analysis (mining) and analytics of social media. The tweet processing algorithm developed by the authors solves such problems as processing slang and non-standard abbreviations, connected words and regional terms. The described problems are typical for the content of social networks. The developed implementation automates the entire process of collecting and cleaning the content of social networks (in particular, tweets). Using the algorithm developed by the authors and its implementation, it is possible to automatically build thematic corpora of content generated by users of social networks. The authors do not provide data on the computational complexity of the proposed algorithm or measurements of the speed of its implementation, which is related to the peculiarities of the latter. The developed implementation uses a mechanism for streaming tweets on the desired topic immediately upon their appearance (the so-called Twitter Streaming API). Provided that the processing time of one tweet is less than the interval between the appearance of tweets (which is performed for unpopular topics), the specific time indicator and its possible reduction is unimportant. However, if it is necessary to process an existing archive of tweets or if they appear quickly in the stream, the algorithm proposed by the authors may show slow results.

In work [4], the authors consider the task of generating a synthetic "question-answer" corpus. To this end, the authors trained three models, each of which is responsible for a certain stage. The first stage involves extracting the answer from the given passage (natural data). The second step is to generate a question using the passage and the extracted answer. The final, third stage involves predicting the answer using the passage and the generated question. If the predicted and extracted answers match, then this "passage-question-answer" triple is added to the generated corpus. The main time expenditure in this case falls on the stage of model training, which is performed only once. Having trained ready-made models, the proposed corpus generation method could be effectively used for various practical tasks, including those where generation speed is important. However, this method is highly specialized, limited to the generation of corpora of only one type – "question-answer".

In [5], the authors describe two approaches to the generation of large parallel corpora for their use in solving the task of correcting grammatical errors. Both approaches use Wikipedia as a source of natural texts (not necessarily in English): the first approach uses page editing history, and the second approach uses two-way machine translation. The idea of the approach of collecting data from the revision history of Wikipedia pages could be used not only for generating corpora to solve the problem of correcting grammatical errors but also for other tasks of natural language processing. For example, it can involve simplifying the text or paraphrasing the sentence. In the cited work, the authors make some efforts to speed up the proposed approaches, for example, reduce the amount of input data by using only a part of the entire editing history. However, the authors do not provide clear data about speed and other acceleration possibilities.

In paper [6], the authors consider their own experience in the automatic generation of a corpus in the Arabic language, intended for the detection of academic plagiarism. As input natural data, the authors used 2,312 dissertations obtained from the depository at the University of Jordan. The method of automatic processing of text data proposed by the authors consists of the following stages: removal of diacritics, punctuation, and special characters; unification of letter forms; tokenization and stemming; division into n-grams; tagging parts of speech. Despite the fact that the authors declare research on three components of work with the corpus (design, generation, and experimentation), most of the work considers the third component – conducting experiments with the finished corpus. This subjective reason (focusing on work with an already generated corpus) can explain the lack of consideration in the cited paper of the issue of the speed of the proposed corpus generation method. It is only indirectly possible to draw a conclusion about the rather moderate effectiveness of the proposed method, caused by the specificity of the input data format and the large number of processing stages.

In work [7], the authors propose a method for generating a thematic corpus of historical texts from newspapers, which are represented in the form of scanned copies. The proposed method is based on a pipeline of the following stages: image processing, optical character recognition (including error correction), and filtering. For the character recognition error correction stage, the authors also propose their own model, formed on the basis of a manually collected data set. The speed of the proposed pipeline may depend on many factors, and primarily on the quality and resolution of the scanned copies used. Such a strong dependence of the speed of work on not only the amount but also other parameters of the input data can be explained by the lack of attempts by the authors to evaluate or measure it. However, it can be argued that the methods that work with textual data (reported in [2–6]) are more effective in terms of speed than the one proposed by the authors, as they do not require work with images.

There are also studies that consider the generation of fully synthetic corpora for their use in solving software engineering tasks (for example, benchmarking or quality assurance). When solving such problems, it may be necessary to generate hundreds or even thousands of different corpora, and the time required for this may be an important parameter. In works [8, 9], the authors proposed and later expanded the deterministic method of corpus generation – CorDeGen. This method has such properties as the determinism of the result and the minimum amount of input data, which simplifies the use of this method in solving software engineering tasks. In [8], the authors show an example of the use of corpora generated by the CorDeGen method when searching for defects in the software implementation of the k-means clustering method. In [9], the CorDeGen method is used to test the effectiveness

of the developed methods of metamorphic testing of software clustering systems. In both works [8, 9], the authors report the results of speed measurements of the developed implementation of the method (on different platforms) but without analyzing the possibilities for its acceleration. Although any possible acceleration is relevant for the results, especially in work [9], in which generation is performed several times for each test in the cloud, with payment for consumed resources.

Based on our review of the literature [2–9], it is possible to conclude that the available works focus only on the generation of corpora itself (development of approaches, methods, algorithms). Available studies leave the question of the speed of the developed methods and algorithms (and their possible acceleration) outside the scope of the research, although it is also important. This can be explained by a combination of both objective and subjective reasons. Among the subjective reasons, it is possible to include the fact that in many works the generated corpus itself is considered as the main scientific achievement, and not the method of its generation, therefore the method is not analyzed much. Objective reasons include conducting experiments on small amounts of data or on such tasks that do not require high speed, which is why its issue is not considered. Separately, it is possible to single out the case of using the corpus generation process when solving software engineering problems. In this case, available works provide speed data but consider it as sufficient, despite the significant potential for acceleration and the possible effect of it.

All this suggests that it is advisable to conduct research into the development of ways to accelerate the existing methods for generating text data corpora (especially the development of parallelized methods).

3. The aim and objectives of the study

The purpose of our study is to identify the possibility of speeding up the process of generating corpora of text data using the CorDeGen method by developing modification(s) of this method that would support parallel execution. This will make it possible to improve the processes of solving software engineering tasks in the field of natural language processing that use the generation of text data corpora, reducing their execution time.

To achieve the goal, the following tasks were set:

– to devise parallel method(s) for deterministic generation of text data corpora based on the basic CorDeGen method;

– to develop a software implementation of the devised parallel method(s) and validate it;

– to analyze the effectiveness of the devised parallel method(s) according to the corpus generation speed criterion, using the developed software implementation.

4. The study materials and methods

4. 1. The object and hypothesis of the study

The object of our study is the process of generating corpora of text data using the CorDeGen method.

The main hypothesis of the research assumes that the corpus generation process using the CorDeGen method could be parallelized and, starting with a sufficiently large corpus size, the speedup effect should exceed the additional cost of parallelization.

The main simplifications adopted in the research process are:

– consideration of only one method for generating corpora of text data, CorDeGen, since the rate of corpus generation, among the considered methods, is the most important for it;

– consideration of only one technique for speeding up the process of generating corpora of text data – parallelization.

4. 2. CorDeGen: deterministic method for generating corpora of texts

The abstract CorDeGen method consists of the steps shown in Fig. 1 [8].

Method 1. Abstract CorDeGen method

1: Input parameter N_{terms} (number of unique terms)

2: Calculation of the number of documents N_{docs} using the function $f(x)$

3: Calculation of the vector $\overline{tf}$ for each term i, containing the number of occurrences of the term in documents, using the calculation of the function $g(x)$

4: Entry to each document of term i, based on the calculation of the number of occurrences

Fig. 1. Abstract CorDeGen method

Fig. 1 demonstrates that the abstract method does not define specific functions $f(x)$, $g(x)$, and the technique of obtaining a linear representation of the term by its index i, but sets certain requirements for them [9]:

– the function $f(x)$ should slow down its growth as x increases;

– the function $g(x)$ must allocate different terms to different documents in different amounts;

– the technique of obtaining a string representation of a term by its index should not require any additional data.

The basic CorDeGen method [8] defines the function $f(x)$ as $\left\lfloor \sqrt[4]{x} \right\rfloor$ and uses the hexadecimal representation of the index i as a way to obtain the string representation of the term. The representation of the function $g(x)$ for calculating the j-th element of the vector $\overline{tf}$ for term i is given in formula (1) [8]:

$$tf_i = \begin{cases} 0, j \notin (c_i - r \ldots c_i + r), \\ \dfrac{1}{2r+2} N_i, j \in (c_i - r \ldots c_i + r), j \neq c_i, \\ \dfrac{2}{2r+2} N_i, j = c_i. \end{cases}$$

In formula (1), c_i is the index of the "central" document for term i; r is the half-length of the range of documents to which the term i is recorded in non-zero quantities. By "central" document, we mean the document in the center of the range to which the term i will be written in twice as much as compared to the others. At the same time, the range (c_i–r...c_i+r) is closed in a ring with respect to the collection of documents.

Thus, the algorithm implementing the basic CorDeGen method is as follows [8]:

1. Calculate the parameters N_{docs} and r from formulas (2) and (3):

$$N_{docs} = \left\lfloor \sqrt[4]{N_{terms}} \right\rfloor,$$

$$r = \left\lfloor \frac{N_{docs}}{5} \right\rfloor + 1.$$

2. For i from 0 to N_{terms} (not inclusive):

a) calculate the linear value of the term that will be recorded in the documents, using the conversion of the number i into the hexadecimal number system;

b) calculate the total number of occurrences of term i in the corpus according to formula (4):

$$N_i = N_{docs}\left(i \bmod N_{docs} + 1\right);$$

c) calculate the indexes of documents that will include term i using formulas (5) and (6):

$$c_i = i \bmod N_{docs},$$

$$i_{range} = \left(c_i - r \ldots c_i + r\right);$$

d) write the i-th term $\frac{2}{2r+2}N_i$ times to the document with index c_i. To all other documents whose indices belong to the i_{range} range, write by $\frac{1}{2r+2}N_i$ occurrence.

The asymptotic computational complexity of this algorithm is $O(N^{1,5})$ [9].

4. 3. Applied hardware and software

To perform all experiments with developed software implementations, a physical machine with the following hardware was used: Intel Core i7-9750H CPU 2.60GHz, 1 CPU (6/12 cores); 16 Gb of RAM (2667 MHz). The described physical machine is running Windows 10 (10.0.19045.3448/22H2/2022Update).

The .NET 8 platform (runtime environment 8.0.0) was used as the main platform for building the software implementation of the CorDeGen method and the devised parallel methods. The .NET platform provides parallel programming capabilities known within the platform as the Task Parallel Library (shown in Fig. 2).

Fig. 2 demonstrates that the TPL provides data parallelism capabilities and an implementation of the task-based asynchronous pattern, which can also be used to parallelize computations, as tasks are executed in different threads from the pool in parallel.

4. 4. Validation of the developed software implementation

The primary user need expected to be satisfied by the software implementation of CorDeGen's parallel method(s) is the equivalence of the results to the results generated by the underlying method.

Given that the CorDeGen method is defined for any positive N_{terms}, and that the execution process of the parallel method(s) may be non-deterministic, traditional oracle-based tests for validating the developed software are inefficient.

The property-based testing (PBT) methodology was used to validate the developed software implementation. PBT is a testing methodology that, instead of testing the exact value at the output of the system under test for a given input, tests whether the resulting value satisfies specified specific properties [11]. In this case, the system under test is a software implementation of the basic and parallel method, and the output is two generated corpora (generated by the basic and parallel method).

Two properties can be defined for such a system under test:

– "weak": for any N_{terms}, for each document $D_{i\in\{1,Ndocs\}}$ generated by the basic and parallel method of generating text data corpora, the set of terms, and the number of their occurrences must match;

– "strong": for any N_{terms}, for each document $D_{i\in\{1,Ndocs\}}$ generated by the basic and parallelized method of generating text data corpus, the set of terms, the number of their occurrences and their order must match.

The description of the properties demonstrates that when a strong property is satisfied, the weak one will also be satisfied automatically, so a certain parallel version of the CorDeGen method can satisfy one or both properties.

The FsCheck library was used for the software implementation of PBT based on the defined properties [12]. An example of property implementation using this library is shown in Fig. 3.

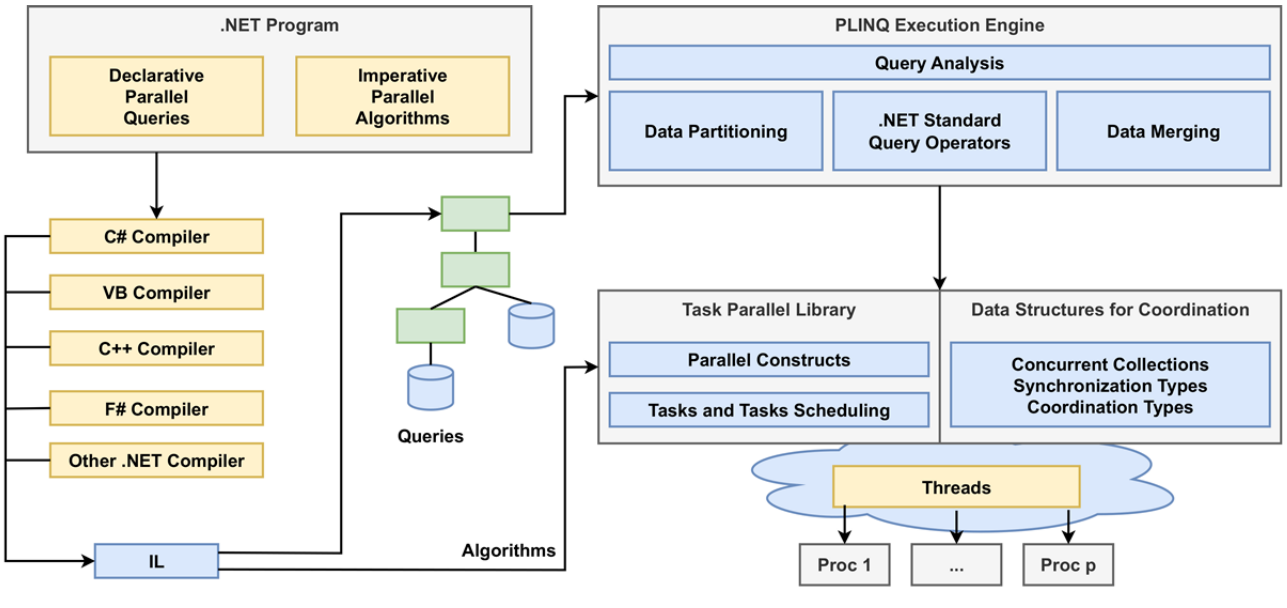

Fig. 2. Parallel programming in .NET [10]

```
[Property]
| 0 references
public Property ParallelGeneratorTest_DefaultPresenter(PositiveInt termCount)
{
    var expectedTexts = new CorpusGenerator(
            termCount.Get,
            ITermPresenter.Default)
        .GetCorpus();
    var actualTexts = new ParallelCorpusGenerator(
            termCount.Get,
            ITermPresenter.Default,
            Environment.ProcessorCount)
        .GetCorpus();

    return actualTexts.SequenceEqual(expectedTexts).ToProperty();
}
```

Fig. 3. An example of implementing a "strong" property using the FsCheck library

Fig. 3 demonstrates that this library provides the ability to generate random input data (in this case, the size of the corpus, which is a positive integer), with support for their compression in the case of failing the test.

4. 5. Designing tests for the developed implementation performance

The purpose of performance tests of the developed implementation is not only to measure the speed of generating corpora of different sizes but also to compare this speed with the speed of generating corpora of the same size using the basic CorDeGen method. Essentially, this means benchmarking should be performed, using the basic method as a baseline.

The BenchmarkDotNet library [13] is the de facto standard of the .NET framework for writing benchmarks, as it is used by the platform developers themselves. This library automates most benchmarking processes (for example, choosing the number of method calls, the number of warm-up and actual iterations), thus providing reliable results and immediately providing their statistical treatment [14].

6 terms of a geometric progression series with a step of 5 and an initial value of 100 are chosen as the corpus size parameter during performance tests. Such parameter values make it possible to evaluate the effectiveness of the developed implementation over the entire range from microcorpora to super-large corpora (312,500 unique terms is comparable to the number of words in the English language [15]).

5. Results of research into the parallelization of the corpus generation process using the CorDeGen method

5. 1. Devising parallelized method(s) for deterministic generation of text data corpora

In the basic CorDeGen generation method, each iteration performed for each term is independent and can be run in parallel. The only issue that arises relates to synchronizing the recording of generated terms to documents.

One of the options for solving this task is to refuse synchronization. In this case, each iteration of the working cycle of the method is performed completely in parallel and immediately writes the term to documents. This version of the CorDeGen parallel method was named "naive" parallel.

Thus, the "naive" parallel CorDeGen method is shown in Fig. 4.

Due to the refusal to synchronize the order of writing terms to generated texts – terms are written immediately after generation – only the "weak" property can be fulfilled for this method. However, this method is easy to implement programmatically, and among all possible ways to parallelize the basic method, this method will show the best results in terms of speed (for large N_{terms}).

Another approach to constructing a parallel CorDeGen method is to split the entire range $0...N_{terms}$ into p separate parts. For each received part of the range, it is possible to generate separate, independent documents that will contain only terms from this part of the range (we shall call these documents sub-documents). After completing the generation process for all parts, the resulting corpus documents can be obtained by combining the received sub-documents of each part in the appropriate order. The method built on the basis of this approach was called simply a parallel method.

Thus, the parallel CorDeGen method is shown in Fig. 5.

This parallel method satisfies the defined "strong" property: terms are written sequentially for each part, and all parts are also combined sequentially.

Method 2. "Naive" parallel CorDeGen method

1: Input parameter N_{terms} (number of unique terms)
2: Calculation of the number of documents N_{docs} using the function $f(x)$
3: In parallel, with a certain degree of parallelism p, for each term i
4: Calculation of the vector $\overline{tf}$, containing the number of occurrences of the term in documents, using the calculation of the function $g(x)$
5: Entry to each document of term i, based on the calculation of the number of occurrences

Fig. 4. The "naive" parallel CorDeGen method

Method 3. Parallel CorDeGen method

1: Input parameter N_{terms} (number of unique terms)
2: Calculation of the number of documents N_{docs} using the function $f(x)$
3: Division of the range $0...N_{terms}$ into p consecutive parts
4: In parallel, with a certain degree of parallelism p, for each part p_j
5: for each term i from this part of the range
6: Calculation of the vector $\overline{tf}$, containing the number of occurrences of the term in subdocuments, using the calculation of the function $g(x)$
7: Entry to each document of term i, based on the calculation of the number of occurrences
8: In parallel, with a certain degree of parallelism p, for each document d
9: Get a document by combining the corresponding subdocuments of all parts of the range $0...N_{terms}$ in the order of division

Fig. 5. Parallel CorDeGen method

5. 2. Development of the software implementation of the devised parallel methods and its validation

The general architecture of the developed software implementation of parallel methods for generating text data corpora is shown in Fig. 6.

In general, the developed software implementation consists of four modules:

– a module containing software implementations of CorDeGen methods;

– a module containing tests based on the properties of software implementations of parallel methods;

– a module containing performance tests (benchmarks);

– an application module with a command line interface for generating corpora of text data.

The software implementations module of CorDeGen methods contains software implementations of three methods: basic, "naive" parallel, and parallel. These software implementations use the abstraction of obtaining a string representation of a term by its index ("Strategy" design template). Such an architectural solution allows us to expand the software implementation in various ways of obtaining a string representation of a term by its index (for researching modifications of the CorDeGen method in this part), without changing the implementation of the methods themselves.

The command-line interface program enables the end user to generate text data corpora and save them to text files using the developed software implementations of the CorDeGen methods.

The test module implements the properties described above in the form of property-based tests to validate the developed software implementations of parallel methods. The "weak" property is used to check the implementation of the "naive" parallel method, the "strong" property is used to check the implementation of the parallel method. The validation results are shown in Fig. 7.

The Performance Tests module contains benchmark implementations of implemented text corpus generation methods, with the base method as a baseline. The results obtained using these benchmarks are represented in the next subchapter.

Test	Duration
CorDeGen.Tests.Integration (6)	677 ms
CorDeGen.Tests.Integration (6)	677 ms
NaiveParallelGeneratorTests (3)	351 ms
ParallelGeneratorTests (3)	326 ms

Fig. 7. Results of validation of the developed software implementations of parallel generation methods

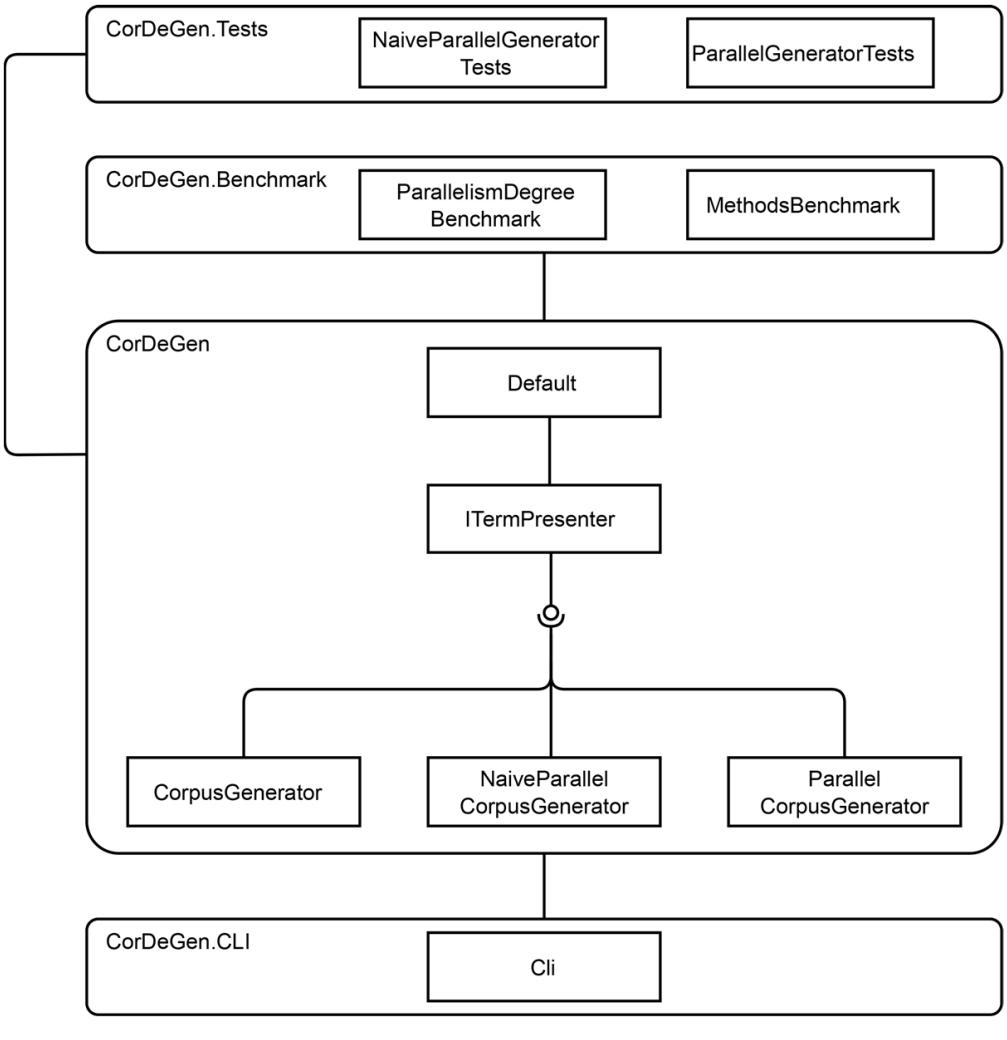

Fig. 6. General software architecture

5. 3. Analyzing performance of the devised parallel methods according to the criterion of the speed of generating corpora

When testing the effectiveness of the devised parallel methods, the choice of the optimal value (the value that will provide the best results in terms of speed) of the degree of parallelism is important for the obtained results. In the general case, the optimal value of the degree of parallelism can depend on many factors but the main ones are two factors: the hardware on which the generation is performed and the size of the corpus that needs to be generated.

This study uses a physical machine with 6 physical/12 logical cores. The selection of the optimal value of the degree of parallelism for this machine was carried out experimentally: by measuring the speed of generating the corpus of the same size by the parallel method but with different values of the degree of parallelism. As the corpus size for this experiment, the size of 312500 terms was chosen (the largest corpus that will be used subsequently when testing the effectiveness of the devised parallel methods).

The average corpus generation time for different values of the degree of parallelism is shown in Fig. 8.

Fig. 8 demonstrates that the average time of generating the corpus from the beginning drops rapidly, then remaining at approximately the same level. The minimum value is reached when the degree of parallelism is equal to the number of logical cores of the physical machine used. Therefore, in further comparative testing of the effectiveness of the devised parallel methods, the value of the degree of parallelism for both methods will be fixed and equal to 12.

The results of testing performance of the devised parallel methods are given in Table 1.

Our results have high accuracy with low variance – the standard error is in the range from 0.06 % to 0.57 % of the average value. Parallel implementations have a higher value of this ratio than the base method implementation because they are more sensitive to random changes in Windows operating system load, while the base method implementation only occupies and runs on one core.

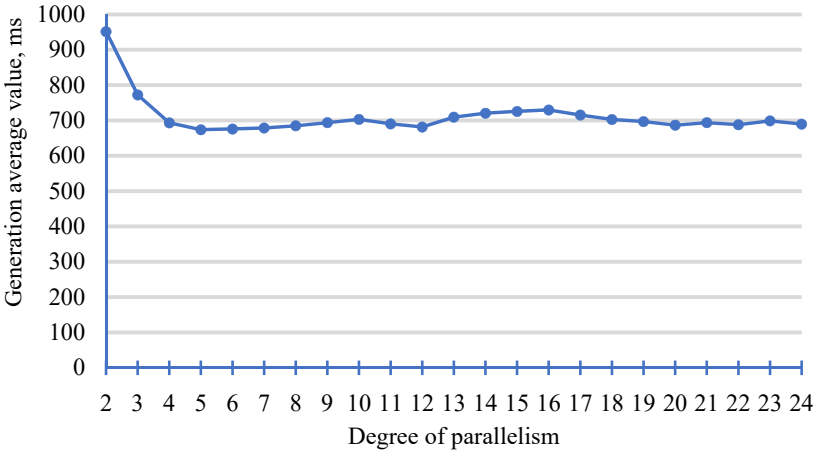

Fig. 8. Average corpus generation time (size: 312500 terms) for different values of the degree of parallelism

Table 1

Results of testing performance of the devised parallel methods

Method	Minimum	Q1	Median	Q3	Maximum
Corpus size: 100 unique terms (μs)					
CorDeGen	13.46	14.012	14.197	14.242	14.424
«Naïve» parallel	39.446	39.509	39.591	39.676	39.912
Parallel	15.717	15.801	15.852	15.878	15.925
Corpus size: 500 unique terms (μs)					
CorDeGen	103.935	110.170	110.871	111.428	112.500
«Naïve» parallel	173.927	174.279	174.329	174.723	175.215
Parallel	128.877	131.012	131.999	133.624	135.282
Corpus size: 2500 unique terms (μs)					
CorDeGen	1.331	1.412	1.422	1.426	1.442
«Naïve» parallel	1.022	1.028	1.029	1.030	1.038
Parallel	0.634	0.644	0.651	0.657	0.662
Corpus size: 12500 unique terms (μs)					
CorDeGen	14.117	14.285	14.349	14.554	14.731
«Naïve» parallel	8.730	8.920	8.983	9.067	9.165
Parallel	14.550	15.264	15.440	15.835	16.474
Corpus size: 62500 unique terms (μs)					
CorDeGen	117.891	119.496	120.616	121.522	123.189
«Naïve» parallel	61.172	62.383	63.827	64.825	67.181
Parallel	64.670	67.344	68.574	71.205	74.863
Corpus size: 312500 unique terms (s)					
CorDeGen	1.254	1.259	1.265	1.270	1.277
«Naïve» parallel	0.587	0.609	0.629	0.636	0.645
Parallel	0.667	0.679	0.694	0.712	0.739

6. Discussion of results of investigating the parallelization of the corpus generation process using the CorDeGen method

The devised parallel methods of CorDeGen generation inherit from the basic method the main features that constitute their advantages over the methods reported in [2–7]. Unlike the methods proposed in [2–7], the methods devised do not use natural text data as input. This makes it possible to significantly simplify the process of generating text data corpus when solving software engineering tasks, due to the absence of the need to store input text data.

Both devised parallel CorDeGen methods have advantages and disadvantages relative to each other and relative to the basic CorDeGen method reported in [8, 9]. These advantages and disadvantages also affect their applicability in specific practical cases.

The main advantage of the "naive" parallel method (Fig. 4) is the simplicity of its algorithmic and software implementation (at the level of the basic CorDeGen method) since most programming languages provide the possibility of parallel execution of cycle iterations. Also, for large corpus sizes, this method could show the best results in terms of speed, as it has no additional overhead.

The disadvantage of this "naive" parallel method, compared to parallel and basic, is that terms are written to documents in a random order – depending on how the iterations of the work cycle were parallelized. In other

words, documents generated by the "naive" parallel method differ from documents generated by the basic CorDeGen method by the order of the terms in the documents.

If the corpus generated by the "naive" parallel method is further processed by methods that ignore the order (for example, clustering methods based on the "bag of words" model), then the above drawback can be neglected. This is explained by the fact that in this case the processing result would completely coincide with the processing result of the corpus of the same size generated by the basic method.

If the corpus is supposed to be treated by methods that do not neglect the order of terms in documents (for example, based on n-gram language models), then the use of a "naive" parallel method may be complicated from the point of view of predicting the processing result.

The main advantage of the parallel method (Fig. 5) over the "naive" parallel one is that the received corpora completely coincide with the corpora obtained by the basic CorDeGen method. Therefore, regardless of how the corpus is treated further, the results for corpora generated by both methods will be the same.

The disadvantage of the parallel method is the need for an additional stage of combining the generated sub-documents into the final corpus. In addition to complicating the algorithm that implements this method, it could also lead to additional overhead when the software implementation of the method is running, compared to the base method.

Validation of the developed implementations of parallel methods (Fig. 7) confirms their validity as for each of the devised methods the corresponding defined property between its output data and the output data of the basic method is performed. In this case, the "weak" property corresponds to the "naive" parallel method and the "strong" property to the parallel method.

Our results (Table 1) of testing performance of implementations of the proposed parallel CorDeGen methods confirm the main hypothesis of the current study. Starting with a sufficiently large size of the corpus (2500 unique terms) to be generated, both parallel methods begin to outperform the basic method reported in [8, 9]. The resulting situation where the parallel method is faster than the base method for a size of 2500 terms, slower for a size of 12500, and faster again for a size of 62500 may be considered an outlier. Such an outlier can be the result of testing on a normal operating system (with crowding out multitasking), as well as possible garbage collector intervention. At the same time, as we can see, the parallel method is faster than the "naive" parallel method on small corpus sizes, and only with the increase of N_{terms}, the "naive" parallel method is faster.

The practical effect of using the proposed parallel methods in comparison with the use of the basic method reported in [8, 9] can be demonstrated using the following example. Let 100 integration tests based on properties, which accept a body of text data as input, be used to validate a conditional information system. Considering that the FsCheck library calls each such test by default 100 times with different inputs, this means in total the need to generate 10,000 corpora when running all the tests once. If the average size of the corpus during such testing is equal to 62,500 terms, then the total effect of acceleration from the use of the devised parallel methods will be about 9 minutes. This is a significant result, considering that with the active development of information systems, integration tests can be run multiple times during one working day.

It should be noted that implementations of parallel methods were tested with only one fixed value of the degree of parallelism, selected by testing on the largest size of experimental data (Fig. 8). For small corpus sizes, reducing the degree of parallelism could significantly speed up the generation process by parallel methods and at least reduce the gap between them and the base method. Analysis of such a two-factor dependence "hardware-corpus size-degree of parallelism" may be of scientific interest for further work on the topic of this study. However, this direction of development may contain difficulties related to the possible variety of hardware that must be taken into account. Another possible continuation of the work on the topic of this research may be the use of other acceleration techniques, for example, memoization or distributed computing.

7. Conclusions

1. Based on the analysis of stages of the basic CorDeGen generation method, approaches to its parallelization have been determined. As a result, two parallel CorDeGen methods have been devised and described – "naive" parallel and parallel, which differ in their approach to solving the task of preserving the order of writing terms in the formed corpus. The "naive" parallel method does not enable preservation of the order of terms in the generated texts relative to the basic method, which limits its applicability. The parallel method preserves the order, so it can be used everywhere instead of the base method.

2. The software implementation of the basic and devised parallel CorDeGen methods is based on a modular architecture. To validate the developed software implementation of parallel methods, a property-based test methodology was used, for which two properties ("weak" and "strong") are defined, which connect the result of generation by basic and abstract parallel methods.

3. The effectiveness of the devised parallel methods was verified using the developed software implementation. To this end, data on the speed of generating corpora of six different sizes (from 100 to 312,500 terms) by basic, "naive" parallel, and parallel methods were collected. The results showed that for large enough corpora, the use of parallel CorDeGen methods accelerates the generation time by 2 times, compared to the basic method.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

All data are available in the main text of the manuscript.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

References

1. Dash, N. S., Arulmozi, S. (2018). Definition of 'Corpus.' History, Features, and Typology of Language Corpora, 1–15. https://doi.org/10.1007/978-981-10-7458-5_1
2. Boujelbane, R., Ellouze Khemekhem, M., Belguith, L. (2013). Mapping Rules for Building a Tunisian Dialect Lexicon and Generating Corpora. Proceedings of the Sixth International Joint Conference on Natural Language Processing. Nagoya, 419–428. Available at: https://aclanthology.org/I13-1048
3. Javed, N., Muralidhara, B. L.(2015). Automating Corpora Generation with Semantic Cleaning and Tagging of Tweets for Multi-dimensional Social Media Analytics. International Journal of Computer Applications, 127 (12), 11–16. https://doi.org/10.5120/ijca2015906548
4. Alberti, C., Andor, D., Pitler, E., Devlin, J., Collins, M. (2019). Synthetic QA Corpora Generation with Roundtrip Consistency. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. https://doi.org/10.18653/v1/p19-1620
5. Lichtarge, J., Alberti, C., Kumar, S., Shazeer, N., Parmar, N., Tong, S. (2019). Corpora Generation for Grammatical Error Correction. Proceedings of the 2019 Conference of the North. https://doi.org/10.18653/v1/n19-1333
6. Al-Thwaib, E., Hammo, B. H., Yagi, S. (2020). An academic Arabic corpus for plagiarism detection: design, construction and experimentation. International Journal of Educational Technology in Higher Education, 17 (1). https://doi.org/10.1186/s41239-019-0174-x
7. Tanaka, K., Chu, C., Kajiwara, T., Nakashima, Y., Takemura, N., Nagahara, H., Fujikawa, T. (2022). Corpus Construction for Historical Newspapers: A Case Study on Public Meeting Corpus Construction Using OCR Error Correction. SN Computer Science, 3 (6). https://doi.org/10.1007/s42979-022-01393-6
8. Yusyn, Y. O., Zabolotnia, T. M. (2021). Text data corpora generation on the basis of the deterministic method. KPI Science News, 3, 38–45. Available at: http://scinews.kpi.ua/article/view/240780
9. Yusyn, Ya. O. (2022). Metody ta prohramni zasoby metamorfichnoho testuvannia prohramnykh system avtomatychnoi klasteryzatsiyi pryrodnomovnykh tekstovykh danykh. Kyiv, 357. Available at: https://ela.kpi.ua/handle/123456789/52417
10. Parallel programming in .NET: A guide to the documentation (2022). Microsoft Learn. Available at: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/
11. Claessen, K., Hughes, J. (2000). QuickCheck. Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming. https://doi.org/10.1145/351240.351266
12. Aichernig, B. K., Schumi, R. (2016). Property-Based Testing with FsCheck by Deriving Properties from Business Rule Models. 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). https://doi.org/10.1109/icstw.2016.24
13. Overview | BenchmarkDotNet. .NET Foundation and contributors. BenchmarkDotNet. Available at: https://benchmarkdotnet.org/articles/overview.html
14. Akinshin, A. (2019). Pro .NET Benchmarking. Apress. https://doi.org/10.1007/978-1-4842-4941-3
15. Soukhanov, A. H. (1992). The American Heritage Dictionary of the English Language. Houghton Mifflin.