IMPLEMENTATION OF PARALLEL AHO-CORASICK ALGORITHM IN PYTHON

M.U.F. Rushda[1], J. Niruthika[2], S. Pranavan[3] & D.R.V.L.B Thambawita[4]

Correspondence: rushdauwaiz@gmail.com

ABSTRACT

Python is a programming language that has experienced tremendous growth in its developer community over recent years. Several reasons can be attributed to this including the ease of use and readability. Data processing is one crucial area where python is popular. It is the second most used language for data analysis. Currently, many computer science applications are becoming more and more data-oriented, especially with the advent of machine learning. Therefore, performance improvement in the data processing field should be looked upon keenly. Multithreading is one crucial way to performance improvement in the data processing. Hence, many languages, including python, support multithreading. However, the multithreading module of CPython, the reference implementation of python, suffers controversy due to GIL (Global interpreter lock) which prevents two threads from entering the same section of code simultaneously. While this is advantageous for I/O bound programs, it is considered as the performance hit for CPU bound programs. In this research, the performance of the parallel version of the Aho-Corasick algorithm is compared against its serial version. Aho-Corasick is a widely used algorithm which addresses the problem of exact string matching, which is one of a significant problem in the computer science domain. The parallelising technique used is the division of input text among threads. Here, we have implemented the serial and parallel versions of AhoCorasick in python and found the real-time elapsed for the parallel portion of the algorithm. The input text size is varied, and graphs are drawn to interpret the result. Also, we have used benchmark implementations of parallel and serial versions of the algorithm used for python. The results show that the time performance of parallel Aho-Corasick algorithm is lower than the serial due to GIL (Global interpreter lock), while the time performance of the benchmark implementation of the parallel version is higher than its serial version.

Keywords: Python, Aho-Corasick, parallel processing, multithreading

1. INTRODUCTION

Many computer science applications nowadays, both in entrepreneurial and scientific domains, are driven by data processing. This is especially true as large amounts of data coming into usage with the advent of machine learning. Volume, velocity, variety and variability can be identified as four important properties of data, according to Gartner's definition of big data. Data processing methods should be able to handle these properties appropriately. Two main considerations of data processing in this regard are:

1. Algorithm
2. Implementation

Both algorithm and implementation should be able to support parallelizing, to gain performance advantage for high volume high

[1] Department of Computer Science and Technology, Uva Wellassa University of Sri Lanka.
[2] Department of Computer Science and Technology, Uva Wellassa University of Sri Lanka.
[3] Department of Computer Science and Technology, Uva Wellassa University of Sri Lanka.
[4] Department of Computer Science and Technology, Uva Wellassa University of Sri Lanka.

velocity data. Data parallelism, which is used in this research, is one major method of parallelizing algorithms. Similarly, Multithreading, which is using different execution lines within a process, is one major method used for parallelized implementations. Thus, programming languages used to implement the algorithms should possess efficient threading concepts.

Python is the most popular programming language used to implement data analysis applications, second only to R, and the growth rate of python is greater than R. There are many reasons behind this; most important ones are,

1. Ease of use allows for rapid prototyping and iteration. Hence different development options can be explored quickly.
2. Python has low LOC (lines of code), which is now considered as one important measure of performance in software industry as the community strive to deny premature optimizations. This is compliant with python's own philosophy, "there should be one and only one- obvious way to do it"
3. Due to the above reasons, python have an active community of great developers. It has an extensive range of purpose-built modules and libraries.

While these make python fit for implementing data processing algorithms, there is a need to analyze the multithreading performance of python for common important algorithms dealing with large amounts of data.

Aho-Corasick is one such algorithm used for string matching problem. In computer science, string matching problem is one fundamental problem, facing great challenge because of the rapid growth of information searching. Aho-Corasick algorithm is one major string matching algorithm used in, intrusion detection, plagiarism detection, bioinformatics, digital forensics, text mining etc.

This research analyses the speed performance metric of parallelized Aho-Corasick algorithm implemented in python. C language implementation of the same algorithm is considered as the bench mark implementation.

1.1. Background

1.1.1. Python as a growing language

Python is an extremely popular, high level, general-purpose programming language. Python is the number one ranking language reported in the IEEE spectrum of top ten programming languages of 2017. Also, Python is the language that shows consistent highest growth index in the TIOBE index for programming languages in 2017.

Python is an interpreted language. The reference implementation of python is written in C, hence the name CPython, while there are other implementations as Jython, Cython and the current PyPy. CPython remains the default and widely used implementation of Python.

Python's design support high extensibility. Python is often used as a glue language, integrating with other languages to extend their functionalities. Its high extensibility also allows for large number of libraries.

a. Parallel processing in CPython

Python threads are real system threads that are they are completely managed by the operating system. The default Python interpreter was designed to have and a thread-safe mechanism, called "GIL" (Global Interpreter Lock). That is, only one thread can execute at a time. This prevents python "threading" module, the POSIX thread module for threads in python from gaining performance advantage by multiple CPU cores.

The reasons behind GIL can be considered as,

1. Increased speed of single-threaded programs
2. Easy integration which C libraries that are usually not thread-safe
3. Implementations that require explicit locking are easier

While these may be advantages for I/O bound programs, GIL is a factor of controversy among python community due it's hindrance to parallel CPU bound programs.

1.1.2. Aho-Corasick

As opposed to traditional string-matching algorithm, Aho-Corasick algorithm can find occurrences of all patterns in each input string in one traversal.

Aho-Corasick algorithm consist 2 distinct steps as follows,

1. Constructing finite state machine
   Constructing a Trie with input pattern words
   Traversing the text over the finite state machine to spell out the patterns

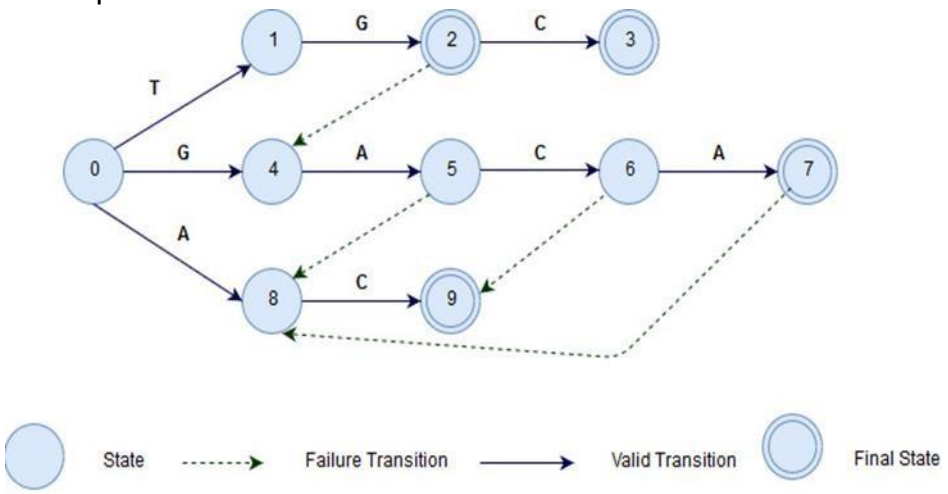

Figure 1. State machine of Aho-Corasick algorithm

2. Traversing the Finite state machine and reading each character from input, storing the result.

a. Parallelizing Aho-Corasick algorithm

Aho-Corasick Algorithm can be parallelized in various ways. Some such ways are,

1. Dividing the state matching building by the division of patterns among threads
2. Dividing the input text file among threads
3. Allocating single thread to each character in the input string (no failure links needed)

In our implementation, we have divided the text file equally among the threads.

1.2. Motivation

Currently there is a mass movement of developer community toward the programming language of Python. This is even more encouraged with Google's decision to go with python in recent years. There are many reasons behind this movement, even though the performance of python is no superior to other languages as Java, C, C# etc. One of such reason is its extensibility. While this is the case there are controversies about the core implementation of Python. GIL (Global Interpreter Lock) is one area of CPython arguments where there are many suggestions and proposals arise for change. But still the question remains as to whether GIL is harmful to your multithreaded programs. As, this is the scenario among python community, we developed interest in knowing the performance of python multithreading for some important I/O bound computational problems.

Such our research idea is on implementing a python version of parallel Aho-Corasick algorithm, as there are no such known implementations, and gaining the idea of its performance improvement over its serial implementation. String-matching algorithms are basic components used in implementations of practical software existing under most operating systems and Aho-Corasick is one commonly used algorithm for string matching.

1.3. Goals

- Implementation serial version of Aho-Corasick in Python, suitable for processing large input files
- Implementation parallel version of Aho-Corasick in Python, suitable for processing large input files
- Comparing the performance of parallel Aho-Corasick and serial Aho-Corasick in python

1.4. Achievement in brief

We have developed python serial and parallel programs for the AhoCorasick algorithm. The programs were tested with files of length ranging from 15208907 characters to 76041088 characters. We have plotted the real time spent in the parallelized portion of the serial and parallel versions. The results show that the time performance of parallel Aho-Corasick algorithm is lower than the serial, while the time performance of the benchmark implementation of parallel version is higher than its serial version. The results show that the time performance of parallel Aho-Corasick algorithm is lower than the serial,

while the time performance of the benchmark implementation of parallel version is higher than its serial version.

1.5. Literature Survey

1.5.1. Python and parallel processing

Python have found great resurgence in recent years. Recent trends of python have reached Google's App Engine (Google Cloud Platform, 2017), Drop box desktop client (mkennedy, 2017) etc. It shows high growth index in the TIOBE index for programming languages (Tiobe.com, 2017).

Though python is gaining popularity as a general-purpose language its major popularity lies in data processing. Many researches have been conducted based on python's data processing libraries. The research on geographic information systems (Sharma, 2017) explores the automatic conversion of data models into python language. Also, research is conducted to using programming to collect, create, or repurpose computerized data to investigate diverse parts of Facebook long range informal communication benefit (Sharma, 2017).

The usage of python in data processing domain attracts attention towards its multithreading capabilities. Python's threads are real threads that are managed by host operating system (GIL, 2017). The reference implementation of python, CPython possess global interpreter lock, or GIL, a mutex that protects access to Python objects (Wiki.python.org, 2017), preventing multiple threads from executing Python bytecodes at once. The presence of the lock is often debated in the python community (reddit, 2017). The major negative concern is that it prevents multithreaded CPython programs from taking full advantage of multiprocessor systems. The positive argument is that GIL is an interpreter level lock rendering advantages to the CPython interpreter by, protecting the interpreter's memory by keeping garbage collection working and maintains the thread safety of memory (jesse noller, 2017).

Though this is the situation with the CPython's threading library called "threading", multicore usage can be achieved in CPython via multiprocessing, spawning several processes within which threads can be operated. This is done via python's "multiprocessing" library. But this approach of parallelizing must combat the overhead of managing shared memory between processes (Toptal Engineering Blog, 2017).

Also, C extensions aren't bound by the GIL; many libraries for Python (such as the math-and-stats library Numpy) can run across multiple

cores. But opting out for C to avoid GIL will drive more programmers away from Python and toward C (Yegulalp, 2017).

1.5.2. Aho-Corasick algorithm

Many of complex computer applications are becoming more and more data oriented. String matching is one important part of such applications as it helps to find patterns and derive conclusions based on them. AhoCorasick is a standard string matching algorithm. It is used in spell checkers, spam filters, intrusion detection, search engines, plagiarism detection, bio informatics, Digital forensics and etc. (Soni, Vyas and Sinhal, 2014). It is an exact set matching algorithm that can match multiple patterns simultaneously, the time complexity does not depend on the number of keywords (Aho and Corasick, 1975) (Bhukya and Somayajulu, 2011). While the naive approach of string matching algorithms has a time complexity of O (n * m), most implementations of Aho-Corasick have O (n + m + z) time complexity, where n is the size of text string, m is the input size of pattern string and z is the number of matches. Aho-Corasick algorithm combines the Knuth-Morris-Pratt algorithm with those of finite state machine the most important aspect of this algorithm is the amount of improvement the finite state algorithm gives over more conventional approaches (Aho and Corasick, 1975). The Aho-Corasick algorithm is one of the most widely used algorithms. The algorithm allows the matching of multiple patterns in a single pass over a text. This is achieved by using the failure links created during the construction phase of the algorithm. The Aho-Corasick presents a major drawback when parallelized, as some patterns may be split over two different chunks of text. Each thread is required to overlap the next chunk of data by the length of the longest pattern -1 (chana S. Vaidya, 2015). Lin et al presented an alternative method for multi-pattern matching on GPU (Lin et al., 2010). Here, each thread is assigned to a single letter in the text. If a match is recorded, the thread continues the matching process until a mismatch. When a mismatch occurs, the thread is terminated. As the number of computational units is large this algorithm is most suited for GPU applications (Nehemia and Lim., 2017). The algorithm also allows for coalesced memory access during the first memory transfer, and early thread termination.

2. METHODOLOGY

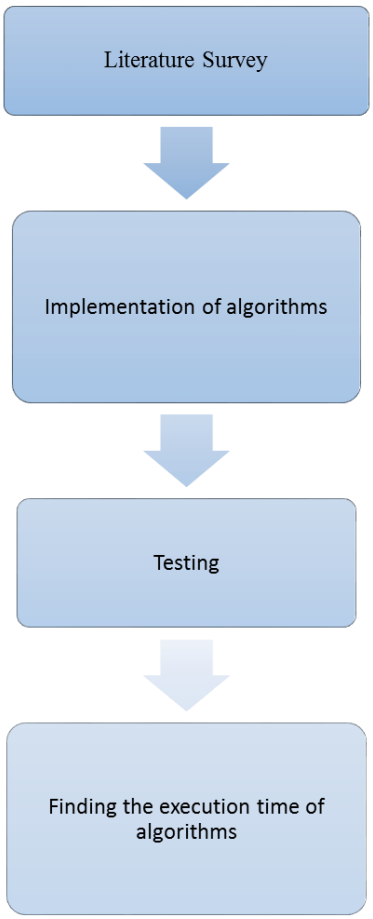

Figure 2. Flow of Methodology

- Literature Survey
  A systematic literature survey was conducted in the areas of Python, its multithreading approach and its multithreading related libraries. Also systematic literature survey on Aho-Corasick, its parallel algorithms and different implementations. Research papers, articles as well as internet forums were considered.

- Implementation of algorithms
  There were four implementations of the Aho-Corasick algorithm as,
  1. Serial Aho-Corasick algorithm in C
  2. Serial Aho-Corasick algorithm in Python
  3. Parallel Aho-Corasick algorithm in C
  4. Parallel Aho-Corasick algorithm in Python

  Here the implementations in C were to act as a benchmark implementation for to find the performance improvement of parallelizing Aho-Corasick algorithm in the absence of GIL. Testing
  Manual testing for the correctness of the results obtained was done with the aid of file comparison software.

- Finding the execution time of algorithms
  All algorithms were executed on same platform (See Appendix). Different length text files were input to the same state machine. Real time spent by a core was calculated after checking the correctness of the results. Graphs were plotted for all executions as number of text lines vs the time of execution

3. DISCUSSION AND RESULTS

3.1. DISCUSSION

Final implementations with large input, pattern files are checked by,

1. Implementing the algorithms by limiting the characters at random points and applying traditional string matching algorithm to find match
2. Sparse some special character randomly between the text (not in nucleotide sequence) so that the results are limited. Then checking at the result index.
3. Results of all the different implementations are obtained and compared at random indices.
4. Using software as gEdit and Araxis to compare the results

Also, after plotting the results the shapes of the graphs were analyzed to see compliance with the time complexity of the portion under timing. Regression analysis was used on the graphs of all 4 implementations the graphs showed linear property, which is in compliance with the complexity (O (n)))

3.2. RESULTS

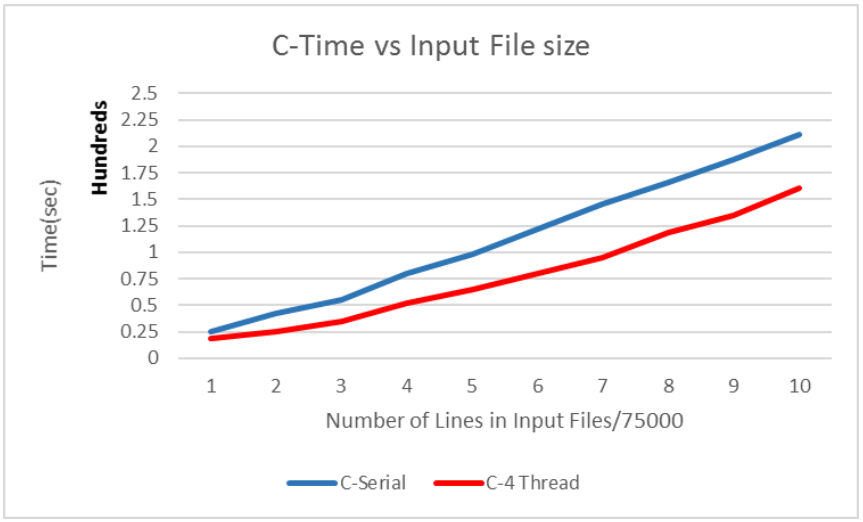

Figure 3. Time vs Input File Size

According to this graph, for every input file size the real time taken for C 4-thread is lesser than the real time taken for C serial version. The difference is higher for large input sizes.

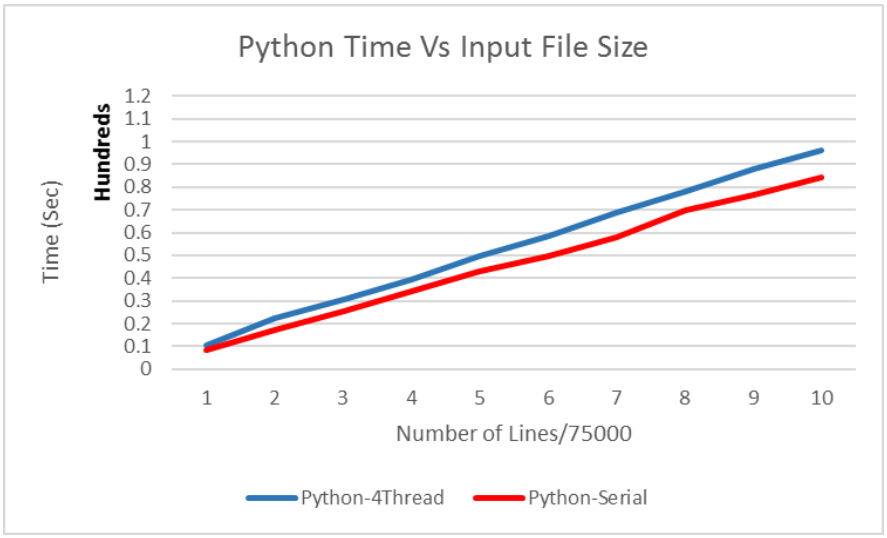

Figure 4. Python Time vs Input Size

According to this graph, for every input file size the real time taken for Python 4-thread is higher than the real time taken for Python serial version. The difference is higher for large input sizes.

4. CONCLUSION

The results show that the time performance of parallel AhoCorasick implemented in Python is lower than its Serial version while the time performance of parallel Aho-Corasick

implemented in C is higher than its serial version. This shows that the effect of GIL of Python is considerate for Aho-Corasick algorithm. Therefore is Python is not suitable for parallelizing Aho-Corasick algorithm as the CPU usage of the algorithm may be considerate compared to its I/O usage.

5. REFERENCES

(@mkennedy), M. (2017). Transcripts Episode #30 Python Community and Python at Dropbox - [Talk Python To Me Podcast]. [Online] Talkpython.fm. Available at: https://talkpython.fm/episodes/transcript/30/python-communityand-python-at-dropbox [Accessed 28 Nov. 2017].

Dabeaz.com. (2017). Cite a Website - Cite This For Me. [online] Available at: http://www.dabeaz.com/python/UnderstandingGIL.pdf [Accessed 28 Nov. 2017].

GIL, W. (2017). Why Was Python Written with the GIL? [Online] Softwareengineering.stackexchange.com. Available at: https://softwareengineering.stackexchange.com/questions/186889/why-waspython-written-with-the-gil [Accessed 28 Nov. 2017].

Google Cloud Platform. (2017). Python on Google App Engine | App Engine Documentation | Google Cloud Platform. [Online] Available at: https://cloud.google.com/appengine/docs/python/ [Accessed 28 Nov. 2017].

Hasib, S., Motwani, M. and Saxena, A. (2013). Importance of Aho-Corasick String Matching Algorithm in Real World Applications. International Journal of Computer Science and Information Technologies, 4(3), pp.467-469.

jesse noller. (2017). Python Threads and the Global Interpreter Lock. [Online] Available at: http://jessenoller.com/blog/2009/02/01/python-threads-and-theglobal-interpreter-lock [Accessed 28 Nov. 2017].

reddit. (2017). Why Was Python Written with the GIL? • R/Python. [Online] Available at: https://www.reddit.com/r/Python/comments/3fqpys/why_was_python_written_with_the_gil/?st=jaivx1b0&sh=603c2c09 [Accessed 28 Nov. 2017].

Sharma, A. (2017). An Approach to Facebook Post Analytics Using Python and Advance Open Source Tools. International Journal of Emerging Research in Management &Technology, 6(6), pp.78-82.

Tiobe.com. (2017). TIOBE Index | TIOBE - The Software Quality Company. [Online] Available at: https://www.tiobe.com/tiobe-index/ [Accessed 28 Nov. 2017].

Toptal Engineering Blog. (2017). Conquer String Search with the Aho-Corasick Algorithm. [Online] Available at:

https://www.toptal.com/algorithms/ahocorasick-algorithm [Accessed 28 Nov. 2017].

Wiki.python.org. (2017). Global Interpreter Lock - Python Wiki. [Online] Available at: https://wiki.python.org/moin/GlobalInterpreterLock [Accessed 28 Nov. 2017].

Yegulalp, S. (2017). Multicore Python: A tough, worthy, and reachable goal. [Online] InfoWorld. Available at: https://www.infoworld.com/article/3079037/open-source-tools/multicorepython-a-tough-worthy-and-reachable-goal.html [Accessed 28 Nov. 2017].

Soni, K., Vyas, R. and Sinhal, A. (2014). Importance of String Matching in Real World Problems. International Journal of Engineering and Computer Science, 3(6), pp.6371 - 63 7 5.

Aho, A. and Corasick, M. (1975). Efficient string matching: an aid to bibliographic search. Communications of the ACM, 18(6), pp.333-340.

Bhukya, R. and Somayajulu, D. (2011). Exact Multiple Pattern Matching Algorithm using DNA Sequence and Pattern Pair. International Journal of Computer Applications, 17(8), pp.32-38.

chana S. Vaidya, P. (2015). Optimization of Parallel Aho-Corasick Multi pattern Matching Algorithm on GPU. International Journal of Innovative Research in Computer and Communication Engineering, 03(06), pp.5191-5200.

C.-H. Lin, S.-Y. Tsai, C.-H. Liu, S.-C. Chang, and J.-M. Shyu, "Accelerating string matching using multi-threaded algorithm on GPU, "in Global Telecommunications Conference (GLOBECOM 2010), 2010IEEE, Dec 2010, pp. 1–5.

Nehemia, R., Lim, C., Galinium, M. and Widianto, A. (2017). Implementation of Multi pattern String Matching Accelerated with GPU for Intrusion Detection System. IOP Conference Series: Materials Science and Engineering, 190, p.012023.