Parallelization of KMP String Matching Algorithm

Ubaid S. Alzoabi[a], Naser M. Alosaimi[a], Abdullah S. Bedaiwi[a], Abdullatif M. Alabdullatif[a]

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh, KSA

***Abstract*—Knuth Morris-Pratt (KMP) algorithm is one of the most popular strings searching algorithm. The parallelism of KMP is one of research topic over last two years. In our paper we simulated two current parallel algorithms with our new proposed parallel algorithm. Our simulation result presents improvement of our proposed parallel algorithm.**

Keywords-component; KMP algorithm; String matching; parallelism; parallel algorithm.

I. INTROCUCTION

String matching is one of the most challenging, sensitive and time-consuming task since it is involved in most of current applications such as text editing, graphics, literature retrieval, biology and others. The basic idea of string matching algorithm is to search for occurrence of a pattern of size m in a text of size n. Some of well know string matching algorithms are Bitap algorithm, Boyer–Moore and KMP algorithms. KMP is founded in 1977 by Donald Knuth, Vaughan Pratt and James H.Morris and considered as first linear time string-matching algorithm with a serial cost of O(n + m).

KMP was an improvement of native algorithm by avoiding extra comparisons of characters in text with pattern. Many researches have been tried to improve the matching efficiency of serial version but recently some researchers proposed more efficient way to improve the performance of KMP algorithm using the concept of parallel processing.

Parallel processing improves the performance by doing a set of tasks simultaneously assigned to different processors. Applying parallel processing in KMP will make the pattern matching faster and then improve the overall speed and performance of the algorithm.

In Section 2 we will discuss the related work, in section 3 the proposed parallel algorithm will be presented, in section 4 we will presents our simulation results and finally we will conclude our study.

II. RELATED WORK

In study [1], the researcher proposed a new parallelized KMP algorithm based on SIMD parallel architecture in which the text is divided into parts and assigned to different k processors. KMP is executed in each part undependably in parallel. Finally, the KMP is applied in the strings at division points between these parts. If the size of the pattern is m, these strings will be the last m-1 characters in the first part and the first m-1 characters in second part. These strings also can be processed in parallel. Although, this algorithm achieves good speedup, it introduces unnecessary overheads since these strings at the division points will be processed only after completing the processing of the texts' parts. The time complexity of the algorithm of text of size n and using k processor is O($\frac{n}{k}$ + 2n-2 +c). In our algorithm we can achieve better parallel cost by reducing the division points cost 2n-2 to (m-1) and overhead c to be almost 0.

Another studies [2-3], proposed parallel algorithms based on MPI programming model to get higher efficiency. The algorithm proposed another way in dividing the text parts. Each text part also contains the first m-1 characters from the next part. By this way the overhead of searching at the division points as in [1] was omitted. The algorithm in [2-3] divides the texts into $\frac{n}{k}$ where the former $\frac{n}{k}$ - 1 parts are equal having length $\frac{n}{k}$ + m-1 whereas the last part's length is $\frac{n}{k}$. However, if n is not multiple of k, the reminder R of the division of $\frac{n}{k}$ will be distributed equally starting from the former parts. This may lead to that some part has length greater than others. For example if n=11, k=3, and m-1 = 2 the algorithm divides the text into 3 parts where the length of the first and the second part is 6 and the last part is 3. In our algorithm we improve the division by balancing the load between the tasks. For example the pervious division can be improved so all the parts have length 5.

III. PROPOSAED PARALLEL ALGORITHM

The text with length n is divided to many parts depend on the number of available processors k. Every part is assigned to a task which is associated with one processor. Each of these tasks does the KMP algorithm in its part in parallel with each other's. We will not explain the concept of serial KMP algorithm since our focus on how to do it in parallel.

Suppose we have four processors in our system. So we divide the text into four parts and store the pattern on the shared memory. Sine we are applying SIMD (Single Instruction Multiple Data) architecture, four processors process the pattern on the four different parts using KMP algorithm in parallel. There is a problem that the pattern may come on the data division between the adjacent parts so will not be detected. To solve this, each part also contains the first m-1 characters from the next part. So the former $\frac{n}{k}$ -1 parts are equal having length $\frac{n}{k}$ + m-1 whereas the last part's length is $\frac{n}{k}$. Another problem, if n is not multiple of k. Let R the reminder of the division $\frac{n}{k}$. In order to provide better load balance between the processors, if R > m-1, R-m+1 will be distributed equally stating from the former parts and m-1 will be added to last part length. If R <= m-1, R will be added to the last part. Suppose that n=15, m=3 and k=4.

Text:

c	a	a	c	a	b	a	a	d	c	a	a	c	a	b

Pattern

A	**c**	**a**

The text will be divided int 4 parts.Each part intiailly will have $\frac{n}{k}$ + overlap m-1 → 3+2=5 and the last part is 3. But, since R=3, the last part length is increase by m-1 to becomes 5 and the reaming 1 from R-m+1 will be added to the first part , so it becomes 6. The divisions shown below.

Text after division:

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

The algorithm divides the text as follows:

- The first part <123456> has length 6→ 3+additional from <R-m+1> + overlap 2 from the next part.
- The second part <56789> has length 5→3 + overlap 2 from the next part.
- The third part <89ABC> has length 5→3 + overlap 2 from the next part.
- The forth part <BCDEF> has length 5→3+ 2 from <m-1> .

Algorithm utilizes the available processors by make the number of string divisions equal to the number of processors k and approximately with same number of characters. Time complexity of our parallel algorithm is O($\frac{n}{k}$ + m-1) .The algorithm process all part in parallel in one stage without dependency between them.

IV. SIMULATION RESULTS

We simulated the algorithms in [1], [2-3] with our algorithm and sequential algorithm. The simulation environment is:

- Processor: i7.
- RAM: 6 GB.
- OS: windows 7.
- Language: java.

Figure 1 shows the simulation result. The yellow line is sequential result, the green line is [1] algorithm result and blue line is [2-3] algorithm result and the red line is our purposed algorithm result. We set k=4, 5 and 6 so the number of parallel tasks is k. The pattern length m is 4. The simulation is done on 31 texts sample with different length n from 33,604 to 1,041,724 characters and each next text is grater by 33,604 characters. The simulation result is the time taken to complete each search in the text in nanosecond. The simulation shows that our algorithm and algorithm in [2-3] gave better performance than the algorithm in [1] and of course better than the sequential algorithm. Although, our algorithm and algorithm in [2-3] gave almost the same performance, but figure 1 and the simulations result in figure 2 and 3 shows that our algorithm slightly more efficient as the text size increases.

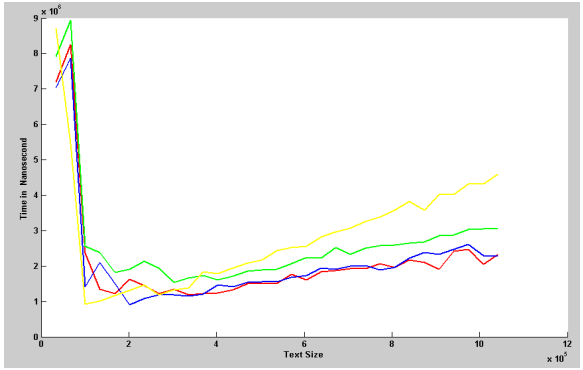

Figure 1. Simulation Result of 4 Processors

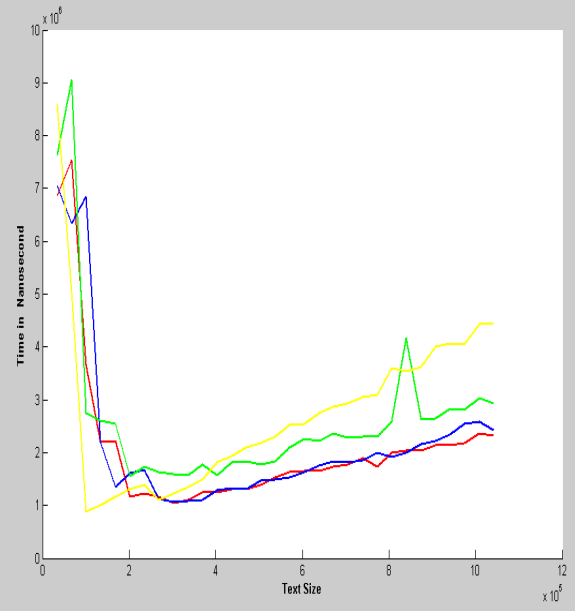

Figure 2. Simulation Result of 5 Processors

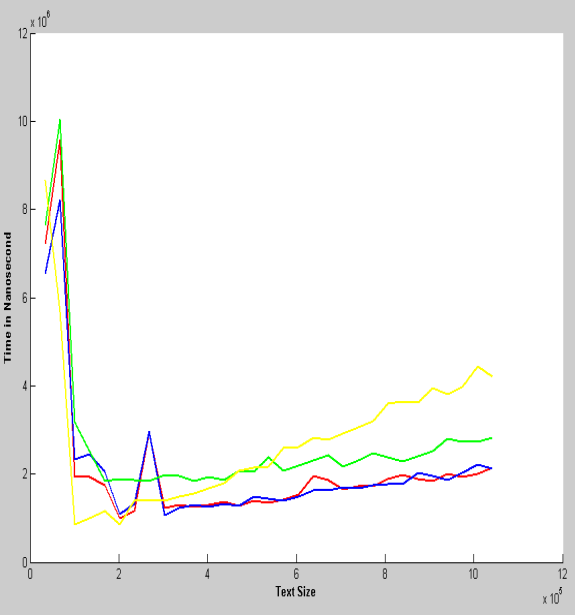

Figure 3. Simulation Result of 6 Processors

V. CONCULSION

The new proposed algorithm reduce the overheads in searching at the divisions points between the parts and also balances the texts divisions between the processor. The simulations show slight improvement over algorithm in [2-3] and great improvement over algorithm in [1].

REFERENCES

[1] Rasool Akhtar, "Parallelization of KMP String Matching Algorithm on Different SIMD architecture: Multi-Core and GPGPU's," Internation Journal of Computer Applications, vol. 49, pp. 0975–8887, July 2012.

[2] Panwei Cao, "Parallel Research on KMP Algorithm," National Natural Sceience Foundation of China, vol. 11, pp. 978–1-61284-459-6, 2011.

[3] G. Duan, Sh. Weichang, C. Jiao and Q. Lin, "The implementation of KMP algorithm based on MPI + OpenMP," 9th International Conference on Fuzzy Systems and Knowledge Discovery, vol. 10, pp. 978-1-4673-0024-7, 2012.