2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th Intl. Conference on Data Science and Systems

A GPU-Based Bit-Parallel Multiple Pattern Matching Algorithm

Che-Lun Hung
Dept. Computer Science and Communication Engineering
Providence University
Taichung, Taiwan
clhung@pu.edu.tw

Hsiao-Hsi Wang
Dept. Computer Science and Information Management
Providence University
Taichung, Taiwan
hhwang@pu.edu.tw

Tzu-Hung Hsu
Dept. Computer Science and Information Management
Providence University
Taichung, Taiwan
g1030394@pu.edu.tw

Chun-Yuan Lin
Dept. Computer Science and Information Engineering
Chang Gung University
Taoyuan, Taiwan
cyulin@mail.cgu.edu.tw

***Abstract*—String matching algorithms have played critical role in many applications, such as DNA sequence comparison, network intrusion detection systems, and so forth. In this paper, we present a parallel multiple pattern matching method based on general purpose graphic processing units to realize fast string searching. In proposed method, we adopt a bit-parallel pattern comparison concept to accelerates string search to achieve efficient parallel search of multiple patterns of different lengths. In addition, we use CUDA framework to enhance the performance of searching string by leveraging GPU computing power. From the experimental results, the proposed method can achieve higher search throughput than other string matching methods. The proposed method is useful for genome sequence comparison and packet payload filtering.**

Keywords—Multiple Pattern; String Match; Bit-Parallel; Parallel Computing; GPU

I. INTRODUCTION

The string matching has played a very improtant role in many domains, such as network, bioinformatics, and so forth. The network intrusion detection systems usually use the pattern matching approaches to compare the packets with patterns to protect computer systems from malicious access [1, 2]. Pattern matching is used to map short "reads" produced by Next Generation Sequencing (NGS) technologies on a reference genome [3]. It is also used in metagenomics analyses to identify at which species belongs each read by searching biologoic databased.

Usually, the string matching algorithms can be classified into to two groups, single pattern matching and multiple pattern matching. The Knuth-Morris-Pratt and the Boyer-Moore algorithmes are the exact string matching algorithms for the comparison of single pattern [4, 5]. Aho-Corasick proposed a multiple pattern matching method by using finite state machine and regular expression [6]. To enhance the performance of searching pattern in the text Baeza-Yates and Gonnet [7] a exact pattern matching algorithm by using bit operations, bit shift and AND, when the the length of the pattern is less than the length of a typical integer word type. Since the bit operation can improve the performance, some methods [8, 9, 10] adopted the concept of bit-parallelelism to take advantage of the parallelism of bit operations. Wu and Manber [8, 9] developed the exact and approximate multiple pattern matching algorithms that the patterns are representation of regular expressions. Kulekci [10] developed a exact multi-pattern matching algorithm for fixed length patterns. In additon, many approximate string matching algorithms [11, 12, 13] for multiple patterns have been proposed to perform genomic sequence alignment including insertion and deletion.

However, such string matching applications are computation-consuming problem. Many parallel processing methods for pattern matching have been proposed [14, 15, 16] to efficiently parallelize string search. In the past few decades, Message Passing Interface (MPI) [17], has played as a very important role for parallel processing applications in many domains. Qu et al. [18] proposed a parallel string matching algorithm based on Aho-Corasick algorithm for large dictionary string matching on a multicore architecture. Chen et al. [19] implemented suffix array on multiple core environment by using OpenMP. Recently, general graphic processing units (GPGPUs) has been used to develop new string matching algorithms. In our previous works [20, 21, 22], we developed the Aho-Corasick and RFC algorithms on GPGPU by using CUDA. Since the properties of memory of GPU we proposed a bloom filter string matching algorithm based on GPU for

978-1-5386-6614-2/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2018.00205

network intrusion detection systems (NIDS) to filter the malicious packets.

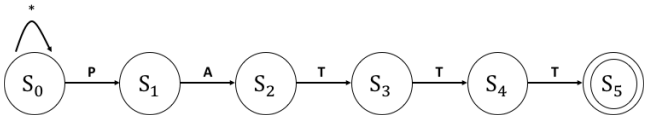

Fig. 1. Automaton for pattern "PATTT".

In this paper, we present a novel bit-parallelism string matching algorithm for multiple pattern based on GPGPU by using CUDA. In the proposed method, we adopted Kusudo's approach [23] as string matching engine. The text and patterns are stored in GPU's memory as bit data. Every thread launched by kernel preforms the string matching engine. The experimental result shows that the proposed method can achieve better performance than other string matching methods.

II. METHOD

A. Bit Parallel Pattern Matching Algorithm

The basic idea of the bit-parallel algorithm is to compare multiple and contiguous charecters of the text with a signle pattern. The First bit parallel algorithm, named Shift OR algorithm, was introduced in 1992 [7]. It was an approximate multiple pattern matching algorithm, and the pattern length is equal or less than word length. The Shift OR algorithm includes two phases: pre-processing and searching phase.

In pre-processing phase, the bit vector $\boldsymbol{B}$ of every character of alphabet is produced. let n and m be the length of a string $\boldsymbol{T}$ and pattern $\boldsymbol{P}$, respectively. let $\boldsymbol{T}_i$ and $\boldsymbol{P}_i$ be the ith character of $\boldsymbol{T}$ and $\boldsymbol{P}$, respectively, where $i \geqslant 1$. A nondeterministic finite automaton is used to indicate a paptern. Fig. 1 shows a automaton from the given pattern. The automation has m states corresponding to the length of $\boldsymbol{P}$. The state s_i is corresponding to $\boldsymbol{P}_i$ of the pattern, where $1 \leqslant i \leqslant m$. In the searching phase, s_i is the state of the automaton to indicate that the pattern $\boldsymbol{P}_{[0 \to i]}$ match the string $\boldsymbol{T}_{[k-l]}$, where $|k - l| = i$. The bitwise operation for state transition can be given by

$$s_i = (s_i{-}1 \ll 1) \mid \boldsymbol{B}[\boldsymbol{T}_i] \qquad (1)$$

where symbols $\ll$ and | represent logical left shift and logical disjunction, respectively. The state s_i transits to s_{i+1} on character $\boldsymbol{T}_k$ if and only if ith bit in $\boldsymbol{B}[\boldsymbol{T}_k]$ is 0. Let $\boldsymbol{S}$ be the state vector where ith bit is 0 to represent state s_i. The pattern is found in the string $\boldsymbol{T}$ where 0 occurs at the most significant bit of $\boldsymbol{S}$ The algorithm is shown as Fig 2.

B. CUDA-Based Bit Parallel Algorithm for Multiple Pattern

The proposed algorithm extends the bit parallel algorithms [23] to simultaneously search for patterns of differing lengths based on GPGPU architecture. The procedure of the proposed algorithm is shown in Fig 3. The proposed algorithm includes

The Shaft OR Algorithm

```
j ← 0 // Initial state of automation for a pattern
All bit in the vector for Text B[T] ←1
for i ← 0 to m: //m is the length of pattern
      B[P_i] ← B[P_i] & ~ (s_j << 1)
end for
k ← 0
i ← 0
while k < n: //n is the length of text
     s_i ← (s_i-1 ≪ 1) | B[T_k]
     if s_i < 1
        output the position k - m + 1
       // the position of text matching the pattern
        k ← k - m + 1
        i ← 0
     end if
end while
```

Fig. 2. The Shaft OR algorithm.

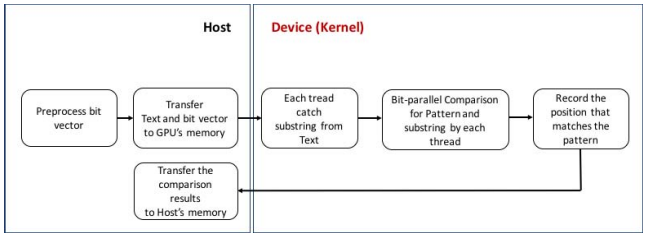

Fig. 3. The Shaft OR algorithm.

two parts, host and kernel parts. The code of the host is responsible for preprocessing input data and transfering data to GPU's global memory. The code of the kernel is responsible for perform search process, and every thread launched by GPU cores executes kernel in parallel. The proposed algorithm is implemented by CUDA [24].

Fig 4. shows the psudo code of GPU kernel function of the proposed algorithm. Each thread compares the pattern and a part of text.

GPU-Based Bit-Parallel Algorithm

```
/*
tid is thread ID launched by kernel
m is length of pattern
n is length of text
p(1) is (m-1)^2
p(4) is m-1
*/
while |(s_tid)| < m:
      s_tid ← ((s_tid >> 1)|p(1))
      s_tid ← s_tid & B[T[tid +|s_tid| × n] % 32] + |B|)
      if(s_tid & (1<<(p(4)- |s_tid|))) != (1<<(p(4)-
|s_tid|)):
         break; //not match
      else
          s_tid ← s_tid >> 1
end while
if 1 == (s_tid & 1):   // match success
   output tid to GPU's global memory
end if
```

Fig. 4. The GPU-based bit-parallel algorithm on CUDA.

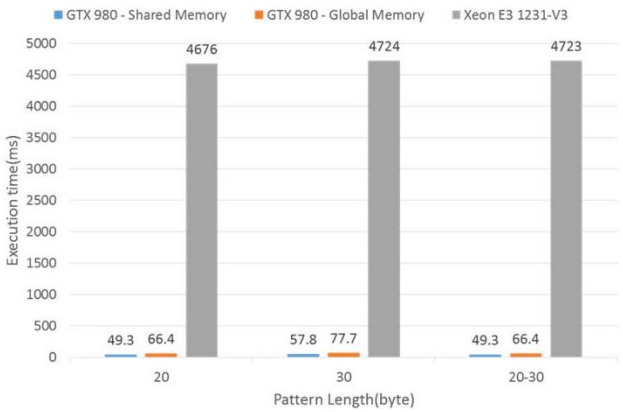

Fig. 5. The comparison of performance among the GPU-based bit-parallel algorithm by accessing two different memories and CPU-based bit-parallel algorithm.

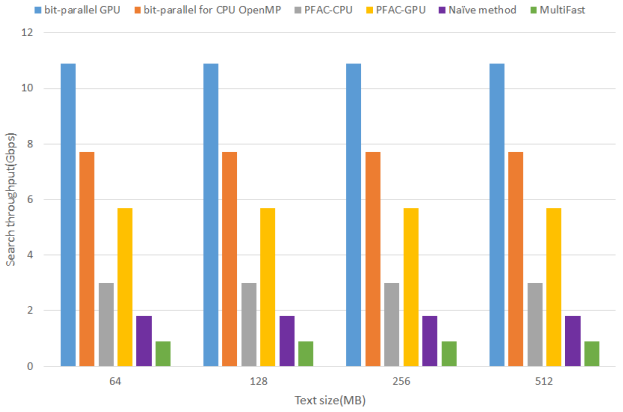

Fig. 6. The comparison of performance among GPU-based bit-parallel algorithm on CUDA and other string matching algorithms.

III. EXPERIMENT

The proposed algorithm is implemented on a server that is equipped with an Intel Xeon E3-1231, a 32 Gbyte RAM, and an NVIDIA GeForceGTX 980 graphics card. The CUDA version is 7.5. The text and patterns are produced randomly. At least, more one pattern can be found in the text. The number of patterns is 10 and the lengths of patterns are 20 and 30. The sizes of text are 64, 128, 256, and 512 MB.

A. Comparison of search throuthput

To evaluate our method, which is implemented by using CUDA and shared memory on GPU, with other well-known algorithms, including Kusudo's approach [23], PFAC based on CPU [25], PFAC based on GPU [25], Naive [23], and MultiFast [26], we measured search throughput while varying lengths of pattern and text, respectively. In this experiment, the size of text is 64MB and the number of patterns is 10 with 20 bytes. The text and patterns are stored in GPU's shared memory. Each thread moves the corresponding part of text and patterns to GPU's shared memory. Fig. 5 presents search throughput measured for our proposed method and other methods. The proposed method can achieve 10.9 Gbps throughput over than other methods.

B. Performance of a varity of matches

In this experiment, we measure the performance of the proposed algorithm while a variety of ratios of pattern matches. The ratio is listed as below,

$$R = m/(|T|/l)$$

where r, m, |T| and l are representation of the ratio, number of the pattern matches, length of text, and length of pattern, respectively. The range of ratios are from 0% to 100%. Fig. x shows the performance comparison. Although, the threads, which cannot find any patterns, are blocked without perform the completed comparison. These threads still need to wait the rest of the threads to finish the completed comparison. Therefore, the performance between different ratios are similar.

C. Performance betwee global memory and shared memory

To optimize the performance of the proposed algorithm using CUDA, we implement the proposed algorithm with two different memories, global and shared memories. The access latency of GPU's shared memory is lower than that of GPU's global memory. Therefore, the comparison of performance among the proposed algorithm with shared memory, the proposed algorithm with global memory, and the Shift OR algorithm on CPU [23] is shown in Fig. 5. The lengths of patterns are 20 and 30. Also the arbitrary length of patterns between 20 and 30. The number of patterns for each length is 10. Obviously, the proposed algorithm with shared memory can achieve better performance than that with global memory.

D. Performance on a varity of block sizes

Since GPU execute the instruction warp by warp. Each black can launch xx warp. Therefore, the number of executing blocks is a factor which affect the performance. For example, launching 512 threads with 8 blocks, it has two execution steps. First 8 warps (256 threads) work, and then other 8 warps work. When using 16 blocks there is only one step, all the threads complete its own job.

In this experiment, we evaluate the performance among different number of blacks. Fig. x presents that the proposed algorithm with more blocks can achieve better performance than that with less blocks.

From the above experiments, it is obvious that the proposed algorithm can achieve better algorithm than other algorithms by leveraging GPU computing power. To optimize the use of GPU, we implement the proposed algorithm with the use of GPU's different memories. In CUDA architecture, the access latency of shared memory is faster than that of global memory. Therefore, our method benefits from use of shared memory.

IV. CONCLUSION

In this paper, we propose an efficient bit-parallel multiple pattern matching algorithm on GPU. We implement the pattern matching engine by CUDA to leveraging GPU computing power. We also implement the proposed algorithm with two different GPU's memory types, global and shared memories. From the experimental results, the proposed algorithm can significantly achieve the better performance than other string matching algorithms. In addition, the performance of the proposed algorithm with GPU shared memory is better than that with GPU global memory. However, the state shaft is the bottleneck of the performance. In the future, we will integrate hash table approach and state automaton to solve this problem.

ACKNOWLEDGMENT

This research was partially supported by the Ministry of Science and Technology under Grants MOST 105-2221-E-126 -003, MOST 106-2218-E-126 -001, and MOST 106-2221-E-126 -001 -MY2.

REFERENCES

[1] A. Tumeo, O. Villa, D. Sciuto, "Efficient pattern matching on GPUs for intrusion detection systems", Proc. 7th Int'l Conf. Computing Frontiers, pp. 87–88, 2010.

[2] G. Vasiliadis, S. Antonatos, M. Polychronakis, E.P. Markatos, S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors", in: Proc. 11th Int'l Symp. Recent Advances in Intrusion Detection, RAID'08, pp. 116–134, 2008.

[3] N. B. Nsira, T. Lecroq, M. Elloumi, "A fast Boyer-Moore type pattern matching algorithm for highly similar sequences", Int J Data Min Bioinform, 13(3), pp. 266–88, 2015

[4] D. E. Knuth, J. Morris, V. Pratt, "Fast pattern matching in strings", SIAM J. on Computing, vol. 6, pp. 127–146, 1977.

[5] R. S. Boyer, J. S. Moore, "A fast string searching algorithm", Communications of the Association for Computing Machinery, vol. 20, pp. 762–772, 1977.

[6] A. V. Aho, M. J. Corasick, "Efficient string matching: an aid to bibliographic search", Communications of the ACM, vol. 18, pp. 333–340, 1975.

[7] B. Y. Ricardo, G. Gonnet, "A new approach to text searching", In Proc. 12th Ann. Int. ACM Conf. on Research and Development in Information Retrieval (SIGIR'89), pp. 168–175, 1989.

[8] S. Wu, U. Manber, "Fast text searching allowing errors", Commun. ACM, 35(10), pp. 83–91, 1992.

[9] S. Wu, U. Manber, "A fast algorithm for multi-pattern searching", Report TR-94-17, Department of Computer Science, University of Arizona, 1994.

[10] M. O. Külekci, "Filter based fast matching of long patterns by using SIMD instructions", in: Proc. Prague Stringology Conf., PSC'09, pp. 118–128, 2009.

[11] S. Altschul, W. Miller, W. Gish, E. W. Myers, D. Lipman. "Basic local alignment search tool" J. Mol. Biol., 215, pp. 403–410, 1990.

[12] D. Lipman, W. R. Pearson, "Rapid and sensitive protein similarity searches" Science, 227, pp. 1435–1441, 1985.

[13] W. R. Pearson, D. J. Lipman, "Improved tools fr biological sequence comparison" , In Proc, Natl. Academy Science, vol. 85, pp. 2444–2448, 1988.

[14] A. Tumeo, O. Villa, D. G. Chavarría-Miranda, "Aho–Corasick string matching on shared and distributed-memory parallel architectures", IEEE Trans. Parallel Distrib. Syst., 23(3), pp. 436–443, 2012.

[15] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors", in: Proc, 11th Int'l Symp. Recent Advances in Intrusion Detection, RAID'08, pp. 116–134, 2008.

[16] X. Zha, S. Sahni, "GPU-to-GPU and host-to-host multipattern string matching on a GPU", IEEE Trans. Comput, 62(6), pp. 1156–1169, 2013.

[17] "Message Passing Interface Forum", *http://www.mpi-forum.org/*.

[18] J. Qu, G. Zhang, Z. Fang, J. Liu, "A Parallel Algorithm of String Matching Based on Message Passing Interface for Multicore Processors", International Journal of Hybrid Information Technology, vol. 9, No.3, pp. 31-38, 2016.

[19] Z. Chen, L. Wu, J. Ma, K. Zheng, "Parallel Optimization of String Mode Matching Algorithm Based on Multi-Core Computing" Journal of Software Engineering, 9, pp. 383-391, 2015.

[20] C. L. Hung, C. Y. Lin, P. C. Wu, "An Efficient GPU-Based Multiple Pattern Matching Algorithm for Packet Filtering", Journal of Signal Processing Systems, pp. 1-12, 2016.

[21] C. L. Hung, C. Y. Lin, H. H. Wang, "An efficient parallel-network packet pattern-matching approach using GPUs", Journal of Systems Architecture, 60(5), pp. 431-439, 2014.

[22] C. L. Hung, S. W. Guo, "Fast Parallel Network Packet Filter System based on CUDA", International Journal of Networked and Distributed Computing, 2(4), pp. 198-210, 2014.

[23] K. Kusudo, F. Ino, K. Hagihara, "A bit-parallel algorithm for searching multiple patterns with various lengths", J. Parallel Distrib. Comput., 76, pp. 49–57, 2015.

[24] "CUDA 7.0", *https://developer.nvidia.com/cuda-toolkit*

[25] C. H. Lin, C. H. Liu, L. S. Chien, S. C. Chang, "Accelerating pattern matching using a novel parallel algorithm on GPUs", IEEE Trans. Comput, 62(10), pp. 1906–1916, 2013.

[26] K. Kanani, "MultiFast: Multiple string search via Aho–Corasick C library", *http://sourceforge.net/projects/multifast/*., 2013.