Journal of Software Engineering

ISSN 1819-4311

Journal of Software Engineering 9 (2): 383-391, 2015
ISSN 1819-4311 / DOI: 10.3923/jse.2015.383.391
© 2015 Academic Journals Inc.

Parallel Optimization of String Mode Matching Algorithm Based on Multi-Core Computing

[1,2]Zhanlong Chen, [1]Liang Wu, [1]Jiongyu Ma and [1]Kuo Zheng
[1]School of Information and Technology, China University of Geosciences, Wuhan, 430074, China
[2]State Key Laboratory of Geography Information Engineering, Xi'an, 710054, China

Corresponding Author: Liang Wu, School of Information and Technology, China University of Geosciences, Wuhan, 430074, China

ABSTRACT

String mode matching is a classical computer research question and one of many key technologies in the network security system. With the hardware technology and network technology development in information age. Big data processing and application requirements for new string matching technology presents new challenges. Suffix array is a string matching and efficient data structure. It solves a lot of complex queries in text database application. This study focuses on the system architecture of multi-core computing environment characterized, optimization and improved suffix array algorithm, enhancement algorithm execution performance, improve the efficiency of the parallel algorithm. Finally, compared with the BF algorithm and the serial algorithm of suffix array. Conclusions drawn parallel string pattern matching algorithm to improve application performance optimization and save hardware cost of great significance.

Key words: Multi-core, string matching, suffix array, OpenMP

INTRODUCTION

With the increasing expansion of information resources, as well as network bandwidth, the text needs to deal with the growing scale. The string matching is very important and is one of the commonly used operations of string. Research in this field across from genetics, bioinformatics and computer music, to keyword search from billions of web pages, image processing and pattern recognition; from cyber espionage, virus intrusion detection technology, network distance learning to text compression, data encryption, data mining technology. These heavy computing tasks makes computers and servers spent a considerable amount of time, string matching has become a key factor affecting the performance of applications (Mei *et al*., 2013). However, the pace of development of existing sequential string matching algorithm is far behind the development of computer hardware, the emergence of multi-core technology makes the rapid development of microprocessor parallel processing technology for parallel processing string matching has important reference.

This study describes the present situation of computer development multi-core computing environments and then describes the string matching algorithm used in this study the idea of suffix arrays, followed by studies of the multi-core parallel computing environment optimized for string matching method and finally the experimental data analyzed and summarized (Huang, 2010).

METHODOLOGY

Multicore computing environment: In the early era of single-core CPU, some microprocessors simulate multiple logical CPU cores to achieve multi-threaded computing, parallel computing technology where relevant include: Bit-level parallelism, pipelining and multi-uni. The way these multi-thread processing technology use to make multipal thread running on a single CPU is thread switching. As for the programmer, the advantage of this technique is that the programming language in order allows parallel execution of instructions. However, the degree of functional units' parallelism and pipelining is limited and the overhead of frequent thread switching will cause a serious influence on system performance.

In recent years, with the continuous development of semiconductor process technology and architecture, the advancement of modern semiconductor process technology in which VLSI is the typical one can't meet the need of microprocessor performance development.

Since 2005, the multiple microprocessor cores integrated on a chip called a single-chip multi-processors, each execution core has its own collection as well as the structural resources (Hager and Wellein, 2010). These multicore chips makes memory can be divided into variable size pages, each core has a L1 cache which has a few KB, shown in Fig. 1. Intel's sandy bridge-E integrates eight kernel in it's chip and has 16 threads, 32K L1 cache, 256K L2 cache and 20M shared cache.

Multi-core processors improve the program's computing power and parallel ability, really achieve a multi-threaded running in parallel, give us higher efficiency and powerful parallel processing capability and in the meanwhile also reduce power consumption and the difficulty of heat dissipation. This structure of a processor system has become an inevitable trend of development (Hennessy and Patterson, 2007; Patterson and Hennessy, 2008). To take full advantage of parallel computing resources provided by multi-core architecture, it is necessary to transform the original serial program to multi-core parallel one, there are five existing parallel programming modes (Bai *et al*., 2006): Stealth mode, data parallel mode, messaging patterns and share variable mode. (1) Stealth mode uses compiler and run-time environment to support a program and discover parallel processing of serial programs, (2) A data parallel requires unrelated sets of data, parallelism programs may be implemented by a data partitioning, (3) Messaging patterns represented by MPI, PVM is a variant of the message passing model and (4) The shared address space model: OpenMP (OpenMP, 2011) as a typical one, the main advantage of adding parallelization directives to the serial program, done automatically by the compiler to parallelize.

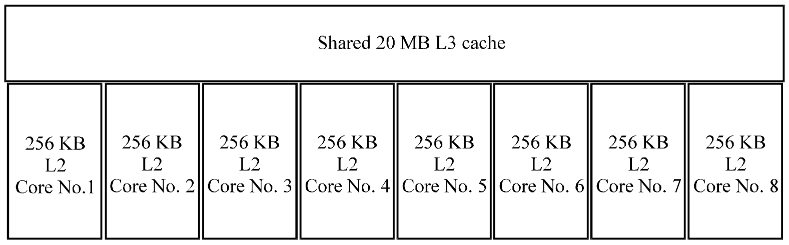


Fig. 1: Shared memory of sandy bridge-E chip

Suffix array: With in-depth study of string matching problem, especially in the fast-growing mass of information retrieval, computational biology and network security field, the problem is now one of the problems in computer science that has been widely studied. String matching problem is a search problem in order to find some symbol sequence (called model) in some of the larger MAmatching symbol sequence (called text) under some certain conditions.

Today, much index structure is designed to optimize the text index (Stoye, 2008). Among them, the suffix array has 20 years of history (Manber and Myers, 1993), has tended to be replaced by compressed suffix array and Burrows-Wheeler (Adjeroh *et al.*, 2008) conversion. However, these new index structures operating efficiency is slower and take more memory space. Suffix arrays are data structure used in a text database to quickly search keywords. In fact, the suffix array is a sequence of all suffixes that have been sorted of a specific word. First, define the text string T [1 ... ..n], length n, the corresponding suffix array SA [1 ... ..n] store text suffix initial position pointers, arrays sorted lexicographically suffix. Figure 2 is an example of the text "Suffixarray $", $ as a suffix, marks the end of an array.

Suffix array stores dictionary sequence suffix information of text T instead of storing text information itself. When we search for a pattern X[1 ... .m], the first to visit is the suffix arrays and

	1	2	3	4	5	6	7	8	9	10	11	12
T:	S	u	f	f	i	x	a	r	r	a	y	$

i	Text suffix	SA[i]
1	$	12
2	array$	7
3	ay$	10
4	ffixarray$	3
5	fixarray$	4
6	ixarray$	5
7	ray$	9
8	rray$	8
9	suffixarray$	1
10	uffixarray$	2
11	xarray$	6
12	y$	11

Fig. 2: Suffix array of string "Suffixarray"

the length of text T, the length of the text T is denoted by |T| = n. Therefore, if we want to find all suffixes text starting with X, an array of sorted dictionaries sequence, we use binary search as our search mode to search suffix arrays: One is the direct pioneer of X and the other is the direct X successor. We obtain the suffix array interval matching pattern in this way. Figure 3 shows how to search.

Figure 4 illustrates the sequence of the NQ search mode. Eventually, in order to avoid interference with the disk access cost, all the patterns of length L is loaded into the main memory. This algorithm uses a continuous array traversal patterns next ( ) function is executed, each of which mode execution SA query substring search algorithm.

Parallel optimization under multi-core computing environments: Multicore technology integrate multiple processor cores into a single chip, without changing the existing structure of the premise, so, computing resources of the entire system obtain a significant increase (Yuan *et al.*, 2011). This makes all the processors are connected to a shared memory unit, the processor when accessing memory use the same memory address space. Since the memory is shared, the data one processor writes in memory can be immediately accessed by other processors, all the processors share memory subsystem and bus architecture. Because of shared memory parallelization, all tasks can be asynchronous read and write shared address space and we can use locks or semaphores to control tasks' access to shared memory (Tinetti and Martin, 2012).

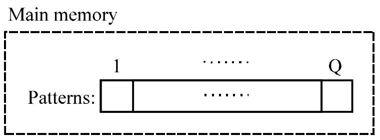
In this process, there is no need to distinguish the identity of the data and no need to care about the communication data between tasks, so, that parallel programming complexity is greatly simplified and reduced the burden on the programmer too.

In this study, we use OpenMP standard, a multi-threaded operating model to achieve a shared memory parallelism within the nodes. OpenMP compiler uses the process to achieve the OpenMP parallel thread (Fig. 5), the program is running according to user-defined maximum number of

```
Search (SA, pattern X, pattern size L)

Left = 1
Right = n
While = (left<right)
    Center = (left<right)/2
    Strncpy (buff, T+SA[center-1],L)
    If (cmp = Strcmp (X, buff) = 0)
            Return (center-1;)
     Else if (cmp<0)
            Right = center-1
            Else
                Left = center-1
```

Fig. 3: Search algorithms on a length l pattern



```
for (int q = 0; q<NQ; q++)
    X = Patterns next ()
    Results = SA. Search (SA, X, L)
```

Fig. 4: NQ search mode

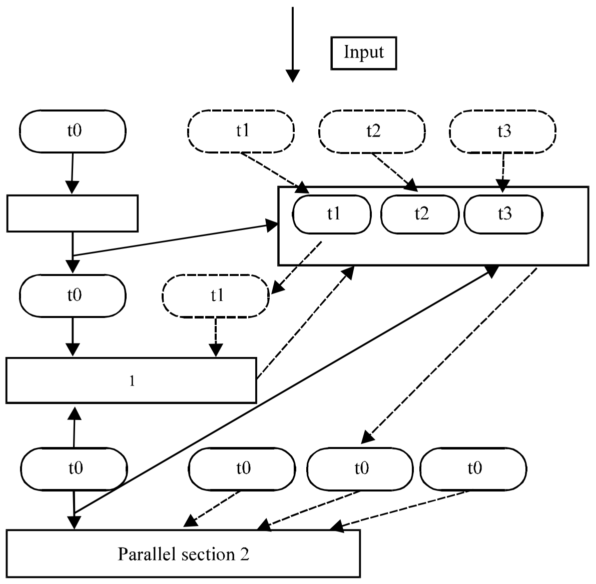


Fig. 5: OpenMP thread management mechanism

threads to avoid the cost which is brought by the frequent great creation and revocation of process. All the threads created when OpenMP job load. After initialization, pick one thread as the main thread and all other threads in the thread pool waiting. When the main thread running to the parallel region, it wake the other threads up by using OpenMP runtime library function comp parallel, after all the other threads exec parallel zone, they return to the thread pool to wait. This method makes multicore, multithreaded processors achieve high execution performance, really reduce system costs and use memory in a much better way. This method makes it easy to implement incremental parallelism. Besides, it have high level of abstraction and portability and is a high-performance and relatively simple parallel programming model on SMP system and in fact it has become the standard for shared memory multiprocessor parallel programming.

We study parallelization of a suffix array in multi-core computing environments, for a P processor cores parallel computer systems. The text T [1...n] is divided into n suffix segments assigned to NT threads, that is the way to construct the suffix array SA, NTi is responsible for matching operation between SA and pattern segment pattern [1...n], in order to avoid read conflicts among each, every thread has a local variable buff to store the length of suffix and use variable left, right and cmp to store the location from where the suffix array SA starts to search. Meanwhile, in order to avoid delay in execution time caused by the critical region, use the result [1...NQ] to store execution results of all threads. In compile guidance statement number pragma omp, “for” command lets a for loop to be executed by multiple threads (Fig. 6).

The features of suffix array search algorithm shows that it more suitable for multi-core processor to improve the efficiency, in particular binary search algorithm we proposed, is more suitable than the other algorithms to access the elements of the suffix array. To further improve efficiency, we have introduced an additional data structure stored suffix array elements which are often accessed showed in Fig. 7.

This new data structure only take very little space, so it can be stored even on the lower level cache storage which can be used to reduce the number of visits of the string, to quickly search for a substring. Figure 8 shows a detailed algorithm: When we start to search a new substring

```
Parallel matching algorithm
Omp set_num_thread (NT)//Number of threads
#pragma omp parallel private (tid, X) shared (L, result)
tid = omp_get_thread_num ()
  #pragma omp for
    for (int q = 0; q<NQ; q++)
          X = patterns [q]
          Results [q] = SA. search (SA, X, L)
```

Fig. 6: Parallel matching algorithm

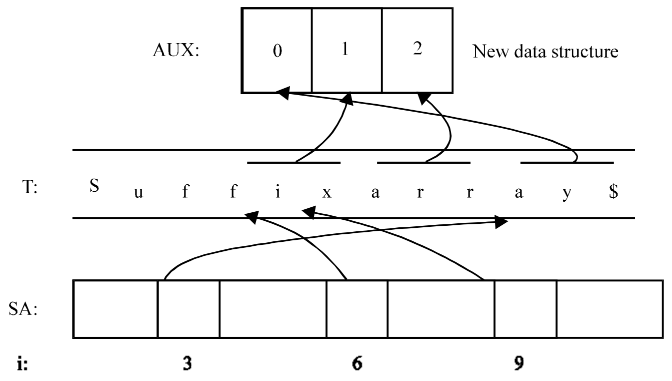


Fig. 7: New data structure

```
Search (SA, pattern X, pattern size L)
left = 1; right = n; step = -1; it = 0;
While (left<right)
    center = (left+right)/2
    if (step = = -1)
        strncpy (buff, T+AUX[1], L)
        else if (step = = 1&& it == 1)
          strncpy (buf, AUX[0], L); // left
            else if (step == 2 && it == 1)
              strncpy (buff, AUX[2[, L);
              else
                strncpy (buff, T+SA[center-1'], L);
    if (cmp = strcmp )X, buff) = = 0)
        return (center-1);
    it++;
    if (cmp<0)
      right = center-1
      step = 1
    else
      left = center+1
      step = 2
```

Fig. 8: Catch optimization

(iter = 1), we first access the second element of AUX and then we access element at the left side of AUX [1] if we need to continue searching. Set iter = 2 and compare the first element of the AUX with substring X, otherwise, compare the third element AUX with substring X. This improvement find the substring we want quickly, enhance the number of cache hits, improve operational efficiency of string matching and achieve cache optimization as well.

RESULTS AND DISCUSSION

In this study, the CPU is the Intel (R) Pentium (R) CPU G860 dual-core processor, clocked 3 GHZ. The operating system is Windows 7. The compiler is VS2005 C++ compiler, in the test, all of the algorithms and testing procedures written in C++ language, We read all the test text into memory at a time and then call the search algorithm to search pattern strings' all appearance in the text, at the beginning and the end of search function we call clock-t to calculate the running time in milliseconds.

When present studies show its search performance, BF algorithms are compared (Huang, 2010). Therefore, this study is no exception. BF algorithm implemented and the suffix array algorithm uses the same experimental environment and compiler to reduce the impact of the environment. We illustrated by experiments the proposed parallel string matching of string pattern algorithm is more efficient than the others and have advantages in terms of computation.

Figure 9 shows in the case of same pattern string tested with different amounts of text data, BF algorithms, suffix arrays serial and parallel algorithms for time-consuming comparison.

As we can see from Fig. 9: In the case with the same pattern string, the execution time of the three algorithms increase with the amount of text data. Among them, the suffix array serial algorithm improves the efficiency of the BF algorithm 3 times. So, suffix array matching algorithm compared with the traditional BF algorithm has greatly improve the performance and efficiency. Then we further investigate much deeper and makes the suffix array algorithm parallelization, experimental results show: The parallel efficiency which is suffix array search algorithm improved is 1.22 times higher than the serial one, performance has exponentially improved.

Experiment studies further on the algorithm's acceleration ratio with different number of threads (accelerated execution time ratio = serial/parallel execution time) (Fig. 10). The amount of

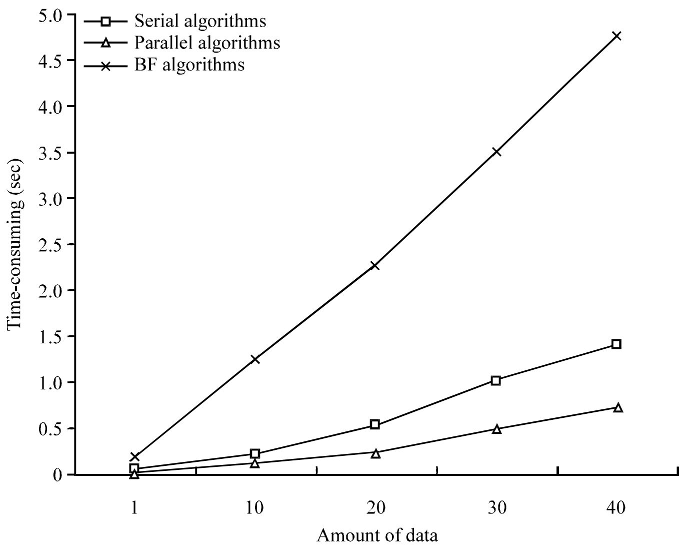


Fig. 9: BF algorithms, suffix arrays serial and parallel algorithms for time-consuming comparison

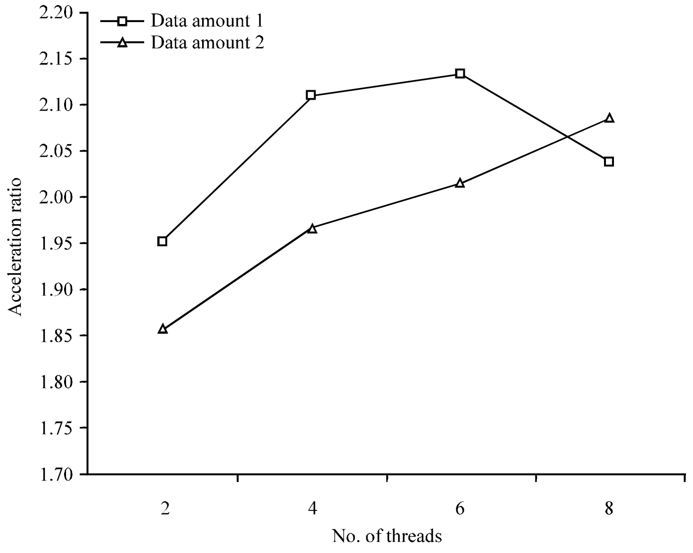


Fig. 10: Acceleration ratio with different number of threads

data to be tested has two different sizes, one is 10 kb, the other is 40 kb. It was found that: With the increase number of threads, accelerate of algorithm than have improved but after reaching a certain number of threads, speedup stabilized, even for the large amount of data, with the increase number of threads, the speedup will decline. This is due to the number of CPU core computer hardware limitations and local conditions, the overall curve is relatively flat, with no significant improvement.

CONCLUSION

At present, the world is faced with the "Information explosion", the advent of the era of big data and the increasingly high performance requirements of matching string. On the other hand, the rapid development of computer hardware provides a powerful shared memory and parallel resources provide a powerful support for string matching study. In the study, we analyze the suffix array string matching algorithms and the parallelization method in multicore computing environment and proposed optimization program to increase the number of cache hits. Experiment use a sample to analyze and compare with traditional BF algorithm, the experimental results also proved that the suffix array of parallel matching algorithms in terms of performance and efficiency has greatly improved. However, due to hardware limitations the laboratory conditions, the existing parallelization is not maximize the reflect. In future study, we will choose a more excellent experimental hardwares, propose a more optimal solution and use parallel technology in multi-core environment to other string matching algorithms to make full use of it's power.

ACKNOWLEDGMENTS

We acknowledge "Five-second" National Science and Technology Support Program for "The research of network service platform for geographic spatial information tool set" (2011BAH06B04) and Fundamental Research Funds for the Central Universities (CUGL090251) and Geographic Information Engineering and National Key Laboratory Fund (No. SKLGIE2013-Z-4-1) and

InformationEngineering in Surveying, State Key Laboratory funded projects (No. 13I02) and Central Universities Fundamental Research special projects (CUGL130260), Research Programs national natural science foundation of China (41401443).

REFERENCES

Adjeroh, D., T. Bell and A. Mukherjee, 2008. The Burrows-Wheeler Transform: Data Compression, Suffix Arrays and Pattern Matching. Springer Science and Business Media, New York, USA., ISBN-13: 9780387789095, Pages: 364.

Bai, Z.Y., X. Yang and J. Kuang, 2006. Parallel Computer Architecture. Science Press, Beijing, China, pp: 19-20.

Hager, G. and G. Wellein, 2010. Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Boca Raton, ISBN-13: 9781439811931, Pages: 356.

Hennessy, J.L. and D.A. Patterson, 2007. Computer Architecture: A Quantitative Approach. 4th Edn., Morgan Kaufmann, San Francisco, CA., USA., ISBN-13: 9780080475028, pp: 198-214.

Huang, H., 2010. Studies on the general parallel method to improve the performance of string matching algorithm. Master's Thesis, Xi'an University of Architecture and Technology.

Manber, U. and G. Myers, 1993. Suffix arrays: A new method for on-line string searches. SIAM J. Comput., 22: 935-948.

Mei, H., X.Y. Wang and L. Zhang, 2013. Progress in research on string analysis. J. Software, 1: 37-49.

OpenMP, 2011. OpenMP application program interface. Version 3.1, July 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf

Patterson, D.A. and J.L. Hennessy, 2008. Computer Organization and Design: The Hardware/Software Interface. 4th Edn., Morgan Kaufmann Publishers Inc., San Francisco, CA., USA., ISBN-13: 978-0123744937, Pages: 912.

Stoye, J., 2008. Su-x Tree Construction in Ram. In: Encyclopedia of Algorithms, Kao, M.Y. (Ed.). Springer Science and Business Media, New York, USA.

Tinetti, F.G. and S.M. Martin, 2012. Sequential optimization and shared and distributed memory parallelization in clusters: N-body/particle simulation. Proceedings of the 24th IASTED International Conference on Parallel and Distributed Computing and Systems, November 12-14, 2012, Las Vegas, USA.

Yuan, Q., J. Zhao, M. Chen and N. Sun, 2011. Performance bottleneck analysis and solution of shared memory operating system on a multi-core platform. J. Comput. Res. Dev., 12: 2268-2276.