International Journal of Security and Its Applications
Vol.9, No.12 (2015), pp.29-34
http://dx.doi.org/10.14257/ijsia.2015.9.12.04

A Parallel Method of Deep Packet Inspection based on Message-Passing Interface

Jia-xing Qu[1*], Guo-yin Zhang[1], Xi-zhong Wang[2], Jia-hui Liu[3] and Da-hua Song[4]

[1]*Harbin Engineering University, Harbin 150001, China*
[2]*Heilongjiang Province Electronic Information Products Supervision Inspection Institute, Harbin 150090, China*
[3]*Harbin University of Science and Technology, College of Computer Science and Technology, Harbin 150080, China*
4Mudanjiang Medical University, Center of Educational Technology and Information, Mudanjiang 157011, China
bsuljh@163.com

Abstract

With the increasing number of cores in multicore processors, it is challenging task how to take advantage of powerful parallel computing for the deep packet inspection. This paper introduces the deep packet inspection with a parallel method which exploits the message-passing interface (MPI). The parallel procedure includes the master thread and the slave thread. The master assigns the data packet to the slave. The slave executes the string matching with rules for inspecting. Both the master and the slave communicate by using MPI functions. The experimental results show that the parallel method is suitable for the trend of the increasing number of cores in multicore processors. Moreover, when the number of threads is equal to the number of cores in multicore processors, the performance arrives at the maximum throughput.

Keywords: *Parallel computing, Deep packet inspection, MPI, String matching*

1. Introduction

Deep packet inspection has been widely used for viruses, spam e-mail filtering, network intrusion detection and prevention systems, and so on. When a computer network packet passes an inspection point, defined criteria are to decide whether the packet may pass or not.

Multicore processors have been mainstreamed. The cores of processors are visibly increasing. The performance of computers' data processing is largely enhancing. Parallel computing has been extensively used to improve existing and traditional algorithms in order to take advantage of multicore processors.

In recent years, many parallel methods are used to improve the algorithms in order to get the enhanced performance [1-3]. Some algorithms based on the software and the hardware are realized for the deep packet inspection.

Typical realized hardware of the deep packet inspection includes FPGA, GPU, and so on. For instance, Baker and Prasanna implemented the efficient pattern matching on FPGA [4]. Peng et al. introduced the string matching system with advanced Aho-Corasick algorithm based on the GPU [5].

Moreover, Fan et al. reported an efficient parallel string matching algorithm based on DFA of the Aho-Corasick algorithm [6]. [1]

*Corresponding author

ISSN: 1738-9976 IJSIA

In addition, Yang et al. pointed out that the deep packet inspection with string matching based on the SSE instruction in CPU, and exploited block predetermination by using automaton optimization [7].

The multicore processors have been widely used to the deep packet inspection with string matching. For example, Scarpazza et al. reported a high-speed string matching based on the CeII/B.E. processor [8].

Villa et al. reported an accelerating real-time string matching realized in multicore processors [9]. Ko et al. reported a parallel method of Aho-Corasick algorithm for the multicore SIMD and the cloud computing environments [10].

As mentioned above, the parallel method of string matching for the deep packet inspection could largely enhance the performance.

Message-passing interface is one of the most popular communication protocols for parallel programming environments [11]. MPICH2 is an implementation of the message-passing interface. The goals of MPICH2 are to provide an MPI implementation for important platforms, including clusters, SMPs, and massively parallel processors.

In this paper, we present a parallel method of the deep packet inspection with string matching by using MPICH2 in multicore processors. The major contributions of this work are as follows.

1. We present a parallel method of the deep packet inspection. In comparison with the existing and traditional systems of the deep packet inspection, we exploit message-passing interface for data parallelism, which can take advantage of multicore processors for enhancing performance.
2. The parallel procedure includes the master thread and the slave thread. The design will largely satisfy with the trend of the increasing number of cores in multicore processors.
3. We demonstrate that the maximum throughput can be arrived when the number of threads is equal to the number of cores in multicore processors.

The rest of this paper is organized as follows. Section II introduces the parallel method of the deep packet inspection. Besides, the master thread and the slave thread are described. Section III gives the experimental environments and analyzes the performance of the proposed algorithm. The last section concludes the paper.

2. Parallel Method

In this section, we present the design and the implement of the parallel method.

2.1. Design of Parallel Algorithm

A data packet or IP packet is transmitted from the source to the destination through network routers. The system of deep packet inspection will inspect the packet and decide whether it may pass or not. Note that when the data packet is encrypted, it needs to be decrypted. We ignore the encoded situation.

Figure 1, shows the system of the packet inspection. The packet inspection system is designed and realized in multicore processors which can quickly process data.

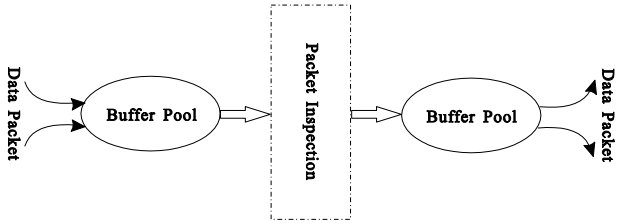

Figure 1. The System of Packet Inspection is Plotted

In this paper, we use the data parallelism for the acceleration of the data packet inspection.

Figure 2, shows the multicore processor with four cores which communicate by using shared memory. Each core can execute a thread or more. Cores can simultaneously realize parallel computing.

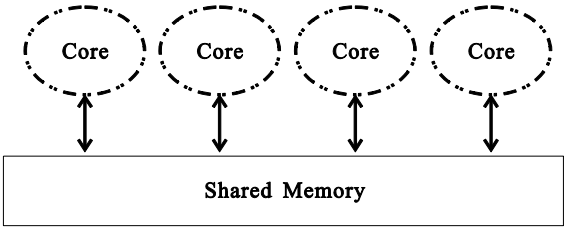

Figure 2. The Architecture of a Multicore Processor with Four Cores is Plotted

Figure 3 shows the parallel packet inspection. The master is a thread in the core.

It distributes the data packet to slave threads for inspecting process. The data packets are located in the buffer pool. The master uses the function "MPI_Send" to send the data.

The slave thread will execute packet inspection. The function "MPI_Receive" is used to receive the data by the slave thread. A thread is executed in a core of multicore processors.

A rule is defined as a string pattern. The packet inspection system uses the algorithm of string matching to inspect the packet.

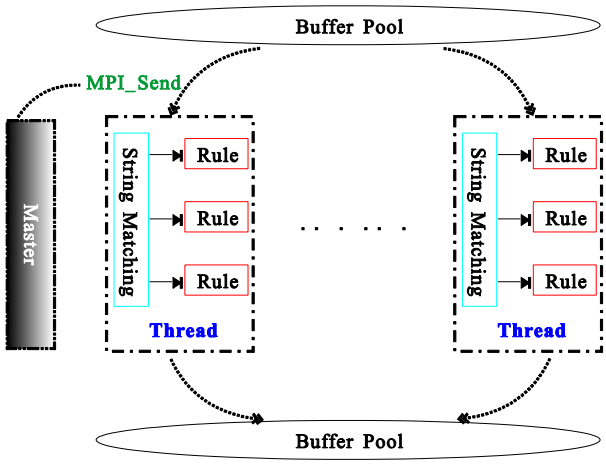

Figure 3. The Packet Inspection System in Data Parallelism is Plotted

2.2. Implement of Parallel Algorithm

The implement of the parallel algorithm includes two parts which are the master thread and the slave thread. Both them can be executed in cores in multicore processors.

The master thread is described as follows.

```
Function_Master(){
  MPI_Initialize();
  while (Not Empty(buffer_pool) = = True) {
    GetPacketFromBuffer();
    GetIdleThread(ThreadID);
    MPI_Send(ThreadID, DataPacket, PacketID);
    MPI_Receive (Flag, ThreadID, PacketID);
    if (Flag = = True){
```

```
        Accept(PacketID);
       } // if
    } // while
    MPI_Finalize();
}// master function;
The slave thread is described as follows.
Function_Slave(){
    if(Myid = = ThreadID){
        InformMaster(Busy, Myid);
        MPI_Receive(DataPacket, ThreadID);
        Flag = StringMatching(Rules);
        MPI_Send(Flag, Myid, PacketID);
        InformMaster(Idle, Myid);
    } // if
} // slave function;
```

The function "MPI_Initialize" is to initialize MPI executing environment. For example, it will specify the total number of threads which will be executed in cores of multicore processors. Besides, each thread will get "Myid".

The function "MPI_Finalize" is to finalize MPI program.

When the buffer pool is not empty, the master procedure gets a data packet from the buffer, and specifies "PacketID". When a thread is idle, the master will assign a data packet to the slave thread. After the thread receives the data packet, the thread will inform the master that it is busy now. The thread does not change the status until it finishes the string matching in rules.

The slave thread will send the result of string matching for the master. Besides, it notices the master that the status of the slave thread will be changed. The master receives the information from slave threads.

3. Performance Analysis

We use the set of rules which includes 447 string patterns.

The experiments are tested on the software environment of Microsoft Windows XP Professional, Visual Studio 6.0 and MPICH2. Besides, the hardware environments are listed as following.

- Condition 1: Intel Celeron G1820 with 2 cores, 2Gbytes memory, and CPU 2.7 GHz;
- Condition 2: Intel Core i5 750S with 4 cores, 4Gbytes memory, and CPU 2.4GHz.

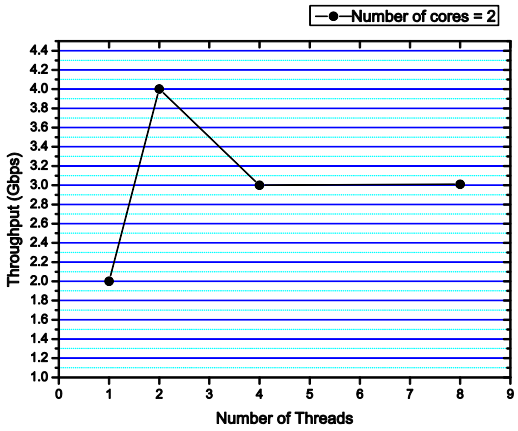

Figure 4. The Performance of the Parallel Method is shown when the Number of Cores is Equal to Two

Figure 4 shows that the experiment is tested in Condition 1. From Figure 4 we can see that the throughput arrives at the maximum about 4.0 Gbps with two threads, which are equal to the number of cores in multicore processors.

As the number of threads is set to four, the performance of the system will decrease. On the one hand, threads are executed in cores of multicore processors with unbalance. It seems that the one of cores is busy processing data while the other is idle. On the other hand, as the number of threads is specified to eight, the performance of the system does not improve.

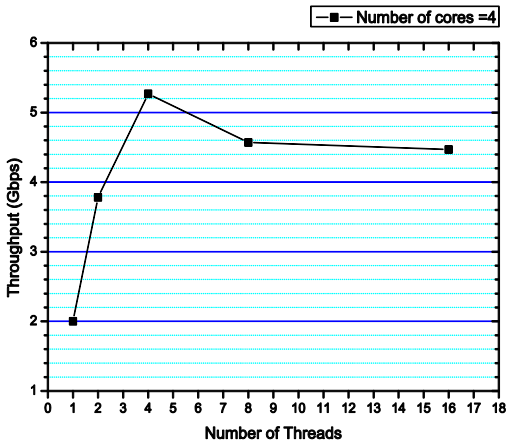

Figure 5. The Performance of the Parallel Method is shown when the Number of Cores is Equal to Four

Figure 5, shows that the experiment is realized in Condition 2. We use Condition 2 in order to further observe the unbalance between the cores of multicore processors and the specified threads. From Figure 5, we can see that the throughput arrives at the maximum about 5.3 Gbps with four threads, which are equal to the number of cores in multicore processors.

When the number of threads is less than the number of cores in multicore processors, the performance of the system will enhance with the increasing number of threads. For example, the throughput with one thread arrives at about 2.0 Gbps. In addition, the throughput with two threads arrives at about 3.8 Gbps. The performance reaches a peak in four threads.

When the number of threads is larger than the number of cores of multicore processors, it is similar that the performance of the system could decrease.

As the number of threads is set to eight, the performance of the system will reduce. When the number of threads is specified to sixteen, the performance of the system does not become better.

As mentioned above, when the number of threads is equal to the number of cores in multicore processors, the performance of the system could arrive at the maximum throughput. Besides, with the increasing number of cores in multicore processors the parallel computing will become better.

4. Conclusions

In summary, we introduce the parallel method of the deep packet inspection which exploits the message-passing interface. The procedure includes the master and the slave threads. The master assigns a data packet for the idle slave thread. Moreover, the slave thread executes the string matching for the packet inspection.

The proposed method is suitable for the trend of the increasing cores in multicore processors. It can take advantage of multicore processors and enhance the performance of the parallel system.

References

[1] W. Lin and B. Liu, "Pipelined parallel AC-based approach for multi-string matching. In: 14th International Conference on Parallel and Distributed Systems", **(2008)**, pp. 665-672.
[2] J. Liu, D. Song . and Y. Xu, " A parallel encryption algorithm for dual-core processor based on chaotic map. In: 4th International Conference on Machine Vision-Computer Vision and Image Analysis Pattern Recognition and Basic Technologies, **(2012)**, pp. 83500B
[3] J. Liu , H. Zhang , D. Song, G. Sun , B"i W. and M.K Buza, "A parallel encryption algorithm of the logistic map for multicore with OpenMP", In: 8th International Forum on Strategic Technology, **(2013),** pp. 47-50
[4] Z.K Baker and V.K Prasanna, "Time and area efficient pattern matching on FPGAs. In: ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays", **(2004)**, pp. 223-232
[5] J. Peng, H. Chen, and S. Shi, "The GPU-based string matching system in advanced AC algorithm. In: 10th International Conference on Computer and Information Technology", **(2010)**, pp. 1158-1163
[6] Y. Fan , H. Zhang , J. Liu, and D. Xu "An efficient parallel string matching algorithm based on DFA" In: International Conference on Trustworthy Computing and Services Communications in Computer and Information Science, **(2013)**, pp. 349-356
[7] T. Yang , H. Zhang , X. Cao , Z. Tian and M.T Qassrawi, "SSE instruction and block predetermination based automaton optimization", In: International Conference on Electrical and Electronics Engineering, **(2012)**, pp. 2229-2242
[8] D.P Scarpazza, O. Villa and F. Petrini , "High-speed string searching against large dictionaries on the CeII/B.E. processor". In: 22nd International Symposium on Parallel and Distributed Processing, **(2008)**, pp. 1-12
[9] O. Villa, D.P Scarpazza. And F. Petrini, "Accelerating real-time string searching with multicore processors" IEEE Computer, vol. 41, no. 4, **(2008)**, pp. 42-50
[10] Y. Ko , M. Jung , Y.S Han. And B Burgstaller, " A speculative parallel DFA membership test for multicore SIMD and cloud computing environments", International Journal of Parallel Programming vol. 42, no. 3, **(2014)**, pp. 456-489
[11] Message Passing Interface, http://www.mpi-forum.org/

