Performance Analysis and Feasibility Study of a Parallelized TCP/IP Implementation*

Hyok Kim, Hongki Sung, and Hoonbock Lee
Hallym Information Technology & Electronics Research Center
Hallym University
Chunchon, Kangwon-do, 200-702
{hkim, hksung, hblee}@sun,hallym.ac.kr

Abstract

Due to the popularity of the TCP/IP protocol suite, no other protocols seem to replace its place in the near future. However, emerging bandwidth hungry applications steadily require higher and higher performance of the TCP/IP. Several improvements have been proposed and made. Notably, a parallelized TCP/IP structure has been proposed in [1]. In this paper, we quantify the performance of the parallelized structure by extracting functions from the 4.4BSD-Lite TCP/IP implementation being executed in a Pentium processor. The empirical results shows that simple functional parallelization cannot improve the performance much enough for the future computer communication subsystems so that pipelining technique as well as VLSI implementation of critical functions should be utilized to achieve the high performance TCP/IP.

1 Introduction

The TCP/IP protocol suite is one of the most widely used form of networking between computers. Since its first implementation in the late 1960s, several improvements have proposed and made. Despite many arguments, is is still claimed that it is capable of serving the high performance demands of future networks and no other protocol suite can easily replace the TCP/IP's place mainly due to its popularity [2].

A typical TCP/IP protocol suite stack consists of several protocols in a layered structure. Although several protocol entities exist within a TCP/IP host, only few of them are time critical, meaning their executions are closely related with the data path during a regular data transfer. Such time critical protocols include TCP, UDP, and IP. Other protocol entities have to be processed only when special situations occur. In the quest for increasing the performance of a communication subsystem, parallelized structures of TCP and IP have been analyzed and proposed [1, 3]. However, neither specific implementations nor quantitative performance measurements are shown.

In this paper, we try to quantify the performance of a parallelized TCP/IP structure proposed in Ref. [1], in terms of execution cycles, executed instruction counts, data read counts, and data write counts. Such measures are done in each function of the parallel structure. In doing so, we used an Intel Pentium based machine running the FreeBSD, a UNIX variant. The empirical results show that simple functional parallelization of the existing TCP/IP implementation alone cannot improve the performance much so that we should utilize pipelining techniques as well as VLSI implementations of time consuming functions to achieve high enough performance of the TCP/IP for the future computer communications subsystem.

2 Summary of Parallelized TCP/IP Structure

Receive and send components of TCP/IP protocol processing can be structured in parallel, operating concurrently on different packets. It is shown that the coarse granularity of this division is appropriate for IP, but not for TCP because of the connection oriented nature of the protocol [1]. In this Section, we summarize a parallelized TCP/IP structure proposed in Ref. [1, 3] and identify functions in the parallel structure which are critical to the overall performance of the TCP/IP protocol suite. Such functions are investigated in detail in Section 3 to analyze performance and feasibility of the proposed TCP/IP parallel structure.

2.1 Decomposition and Data Flow

Figure 1 illustrates coarse grain decomposition and data flow within the TCP/IP layers. The TCP layer is subdivided into five major components: TCPDataSend, TCPDataReceive, TCPConnControlSend, TCPConnControlReceive, and TCPConnDynamics. The IP layer is subdivided into two major components. IPSend and IPReceive, with accompanying protocols ICMP and ARP.

*This paper appeared in the *Proc. of Korean Institute of Communication Sciences Conference*, vol. 15, pp. 443–446, Nov. 1996. This research was supported by a grant from the Ministry of Information and Communication of Korean Government.

Figure 1: Data flow within TCP/IP layer

Assuming an already established connection, the regular data path flows through TCPDataReceive, IPSend, and IPReceive components. In the TCPDataReceive, the regular data path is distinguished from the urgent data path. A third type of TCP segments involves the processing of connection management and control functions. Such segments are forwarded to TCPConnControlReceive. Other possible data flows, shown in Fig. 1, represent error cases. If TCP detects an error, the TCPConnControlReceive is invoked. If an error is detected within IPReceive, ICMP is informed and ICMP message might be generated and sent through IPSend. In addition to these flows, the TCPConnDynamics might generate segments in special situations, such as the timeout of a retransmission timer. Finally, the ARP is called by IP to resolve address.

It is easy to see that TCPDataSend, TCPDataReceive, IPSend, and IPReceive components are more time critical than others. Thus, we further decompose such a component into several functions, in the following subsection, to fully extract potential parallelism in segment processing. However, we should note that concurrent processing of other components would also improve the performance, in case of multiple active connections.

2.2 Parallelism within Each Component

Detailed description of each function is this subsection can also be found in Ref. [1]. In this subsection, a name of each function is written in italic for clarity.

2.2.1 TCPSend

Figure 2 depicts the parallel structure of the TCPSend component. TCPSend may receive *multiple* send commands from an application. In this case the corresponding data is queued until the *segment* function decides to form a TCP segment to be sent. Concurrently, parts of the *TCP header* and the *option fields* can be constructed. Completing the segment function, the header fields can be filled. Then, *TCP checksum* can be calculated, the header completed, and the segment forwarded to IPSend component.

Figure 2: Parallel structure of TCPSend

2.2.2 IPSend

Figure 3 shows functions in the IPSend component and their potential parallelism. IPSend receives an IP datagram from TCP. It first performs the *routing* function. Upon completion of the routing function, *address translation* using the ARP protocol, *fragmentation*, and *IP header* processing can be performed in parallel. After construction of a fragment and completion of the header generation functions, the *checksum* can be calculated and the header completed.

Figure 3: Parallel structure of IPSend

2.3 IPReceive

Parallel data flow within the IPReceive component is described in Fig. 4. IPReceive an IP fragment from the lower layers. On arrival of the fragment, all *checks on header* *fields* can be performed in parallel with the IP checksum. For a correct frame, *option* processing function and the *reassembly* function can be performed in parallel. After completion of the reassembly function, IP datagram is passed to TCPReceive.

Figure 4: Parallel structure of IPReceive

2.3.1 TCPReceive

Figure 5 shows parallel data flow within the TCPReceive component. For every TCP segment received the *local connection name* has to be identified. Then, various *header check* functions should be performed in parallel with the *TCP checksum* function. After that, correctly received data segments have to be *ordered* before they can be released to the receive function for delivery to the application.

Figure 5: Parallel structure of TCPReceive

3 Performance and Feasibility Analysis for the Parallelized TCP/IP Structure

In this Section, we discuss what we measured as performance metrics and how we did. Finally we show the empirical results.

3.1 Measurement Tool and Performance Metrics

We made our measurement using event/cycle counters implemented in the Intel Pentium processor. Although such counters are not documented for public users, they have been reported in [4] and an example application can be found in [5].

The Intel Pentium processor has one 64-bit cycle counter and two 64-bit software-configurable event counters. Each event counter can be configured to count one of a number of different hardware events such as data reads, data writes, instructions executed, hardware interrupts, data read cache misses, etc. In addition, the counters can be configured to count events in the kernel mode or in the user mode or in both. However, we counted events only in the kernel mode since the TCP/IP in the FreeBSD is implemented in the kernel. It should be noted that the cycle counter continues to increment when the machine is halted, but no other event counters are incremented.

To compare and contrast performance of functions in the parallelized TCP/IP structure, we measured cycle counts, instruction counts, data read counts, and data write counts. The cycle counts combined with the instruction counts will give us some idea of how complex the function is. The data read/write counts will tell us the memory bandwidth requirement of the corresponding function.

3.2 Empirical Results

In measuring performance of the parallelized structure of TCP/IP proposed in Ref. [1], we used a platform as follows: Our experimental machine was based on the Micronics M54Hi+ motherboard with a 166MHz Intel Pentium process, a 256 K byte cache, and 32M bytes of main memory. Our machine runs the FreeBSD which is a variant of UNIX and its TCP/IP implementation is based on the 4.4BSD-Lite. The Micronics M54Hi+ motherboard has a PCI bus and a ISA bus. The machine is equipped with a PCI Wide-SCSI controller and a 3Com ISA Ethernet controller.

Figure 6 depicts performance of the time critical function in the IPSend component. The graph shows that performance of IPSend is dominated by the sequential execution of IP routing and IP checksum functions. Figure 7 shows performance of IPReceive where the IP checksum function determines performance of IPReceive. Figure 8 and figure 9 depicts the performance of TCPSend and TCPReceive, respectively, where the segment size of 1412 bytes is used.

Figure 6: Performance of IPSend

To estimate speedup due to the parallelization, we also measured the performance of the sequential executions of the IPSend, IPReceive, TCPSend, TCPReceive components. Using the results from the cycle counts, estimated speedups of such components are 1.06, 2.55, 1.14, and 1.02, respectively. Thus, overall speedups of the send and receive parts of the TCP/IP are 1.13 and 1.05, respectively. Thus, according to our performance analysis, we could conclude that simple functional parallelization of the existing TCP/IP code cannot improve performance much enough for the future computer communication subsystems. We should note that this estimated speedup is owing only to the simple functional parallelization, not considering overlapping of functions in a pipelined manner. With the elaborate pipeline design, we could further increase the speedup. In addition, with the aid of VLSI engines of time consuming functions such as checksum calculation, we can enhance the performance of the TCP/IP protocol suite.

4 Discussions and Concluding Remarks

To analyze the performance of the parallelized TCP/IP structure, we focused on the time critical functions in conjuction with the data path during a regular data transfer. Performance is measured in terms of cycle counts, executed instruction counts, data read counts, and data write counts. Since the cycle counter in the Pentium processor

Figure 7: Performance of IPReceive

Figure 8: Performance of TCPSend

Figure 9: Performance of TCPReceive

continues to increment even when the machine is halted, also since it increments when the processor is switched to run other processor and interrupts, it is very hard to measure the precise execution time of the program block being executed. To ensure the measurement as accurate as possible, we repeated the same measure several times to take out such results which are far off the standard deviation. Thus, we can have a rough idea of relative execution time of the program under measurement using the cycle counts in conjunction with the event counts such as instruction counts and data read/write counts.

We also did not consider functions which are not directly related with the regular data transfer such as urgent segment, error messages, etc. Thus, the overall performance which might be drawn from our empirical results should imply nearly peak performance.

To achieve higher throughput of the TCP/IP protocol, the pipelining technique should be used combined with the parallel structure. In doing so, we need to further analyze each function in detail for the determination of the optimum number of pipeline stages and load balance among stages. We may need to subdivide some functions into more sub-functions or group some together for efficient pipeline design. In addition, VLSI implementation of time consuming functions should help in increasing the performance.

References

[1] O. G. Koufopavlou, A. N. Tantawy, and M. Zitterbart, "Analysis of TCP/IP for High Performance Parallel Implementations," *Proc. of 17th IEEE conference on Local Computer Networks*, pp. 576–585, Sep. 1992.

[2] D. D. Clack and V. Jacobson, "An analysis of TCP processing overhead," *IEEE Commun.*, vol. 27, pp. 23–29, June 1989.

[3] O. G. Koufopavlou, A. N. Tantawy, and et.al, "Parallel TCP for High Performance Communication Subsystems," *Proc. of 17th IEEE Global Telecommunications Conference*, pp. 1395–1399, Dec. 1992.

[4] T. Mathisen, "Pentium secrets," *Byte*, pp. 191–192, July 1994.

[5] J. B. Chen and et al, "The Measured Performance of Personal Computer Operating Systems," *ACM Trans. Computer Systems*, Feb. 1996.