Efficiency Improvement of Data Duplicating Method Based on Pipeline Processing

A. Kongsakul[#1], C. Tantibundhit[#2], and P. Kovintavewat[*3]

[#] Electrical and Computer Engineering Department, Faculty of Engineering, Thammasat University, Thailand
[*] Data Storage Technology Research Unit, Faculty of Science and Technology,
Nakhon Pathom Rajabhat University, Thailand

Email: [1]apichait_chut@hotmail.com, [2]tchartur@engr.tu.ac.th, [3]piya@npru.ac.th

***Abstract*—The data duplicating method is widely used in the application of digital forensics. Practically, it utilizes the Linux operating system and the DD (Disk Dump) program for data duplicating because it can maintain data integrity, and users can be confident that the duplicated data is same as the original data. However, this method is based on single processing, which imposes the limitations in increasing the speed of data processing. This paper proposes a new approach to improve the efficiency of the DD program based on two techniques. The first technique is to use pipeline processing both to read the original data and to write the duplicated data, while the second technique is to separate the two channels connected to the device that stores the original data and the device that stores the duplicated data. Results indicate that combining these two techniques will help the DD program significantly increase the speed of data duplicating.**

I. INTRODUCTION

Data duplicating can be employed with several purposes, for example, i) duplicating data for storing important content for future use when the original data is damaged (sometimes known as "backup data"); ii) duplicating data for installation work in a computer system that utilizes same devices and drivers; and iii) duplicating data for uses in digital forensics.

Digital forensic is the process of acquiring, verifying, analyzing, and storing the evident in digital form that resides in a device, such as a personal computer, a notebook, or a personal digital assistant (PDA). This digital data can be used to identify an offender (or a wrongdoer), or it can be considered as an evident in court. This digital evident is extremely sensitive because it can be damaged or destroyed if precautions are not carefully taken. Unlike a "hard" evident such as fingerprint, the digital evident is easy to be hidden and to be altered. Therefore, the key of digital forensic is to duplicate the original data as completely as possible. Then, only duplicated data will be utilized for further analysis instead of using the original data. In other words, it is required for digital forensics that the evident be maintained in an original form (no modification or alternation) because if we directly do something upon the evident, this evident might be altered. Thus, it is crucial to make a copy of the data that we want to analyze, and this copied data must retain data integrity.

In practice, the Linux operating system and the DD program are generally accepted to be utilized for data copying in digital forensics. This is because this method can retain data integrity and it can guarantee that the duplicated data is same as the original data [1]. This method will duplicate data in a sector-by-sector manner. Also, it can verify data integrity by using a hash function [2] to compute the massage digests of the original data and the duplicated data. If these two massage digests are equal, it implies that the duplicated data is same as the original data [2]. Because the DD program works in a command line mode and duplicates data in a sector-by-sector fashion, it takes a long time to duplicate data.

Additionally, there are many methods for data duplicating, such as using an EDBA algorithm [3], which can duplicate data well but this algorithm can only be used with the file system "Ext 2." Moreover, this EDBA algorithm is still not accepted in digital forensics. Another method to speed up the DD program is to specify the block size of the data that is read from a device called "Source" and that is written into a device called "Target." This is because the block size is set to 512 bytes by default [2], thus taking a long time to read and write data many times for duplicating the whole data. Thus, if we could set the block size corresponding to the buffer in a storage device (e.g., hard disk drives have a buffer of 8 MB), the time for data duplicating would be reduced.

Since the DD program is operated in a single processing mode, it limits data processing speed. We propose a new method to improve the efficiency of the DD program. The proposed method employs two techniques. The first technique is to use pipeline processing to read the original data and to write the duplicated data. This technique will divide the CPU processing into many steps and exploit them in "wait state" [4]; thus improving the processing time. The second technique separates the two channels that are connected to the device that store the original data (referred to as "Source") and the device storing the duplicated data (referred to as "Target"). By combining these two techniques, we can improve the efficiency of the DD program to expedite the processing time of data duplicating.

This paper is organized as follows. After briefly describing the existing method for data duplicating in Section II, Section III explains the details of the proposed method. Simulation results are given in Section IV. Finally, conclusion is provided in Section V.

II. EXISTING METHOD

The existing method uses the Linux operating system and the DD program to perform data duplicating for digital

forensics. However, this method is based on single processing, which imposes the limitations in increasing the speed of data processing.

III. Proposed Method

The proposed method utilizes the two techniques so as to improve the efficiency of the DD program to expedite the processing time of data duplicating. These two techniques can be explained as follows.

A. *First Technique*

In general, CPU operates in a sequential processing manner [5]. For instance, after "Fetch" operation is performed, it sends to "Decode" operation. The total time for processing, T_s, is given by

$$T_s = kn\,, \tag{1}$$

where k denotes the time required to processing in each cycle, and n denotes the number of commands needed to be processed.

In practice, the above-mentioned operation cannot increase the efficiency of process because when subsequently submitting work, CPU will be in an idle state until a new work arrives. In fact, each work can be performed simultaneously.

Pipeline processing [5] is the technique that can execute several overlapped commands at the same time, which in turn improves the CPU efficiency, as shown in Fig. 1. Specifically, the pipeline processing can be divided into 5 steps, namely,

1) "Instruction Fetch" (or a command receiving part) will receive new commands from main memory or from Instruction Cache before sending them to other parts.
2) "Instruction Decode" (or a command translation part) will classify received commands.
3) "Get Operands" (or a data receiving part) will receive and store data used to processing.
4) "Execute" (or a processing part) is the step to processing according to commands and operands obtained from 2) and 3)
5) "Write Result" (or a replied data writing part) is performed when finishes data processing. The result will then be stored in register or in data cache.

By utilizing the pipeline processing technique [5], the total processing time, T_p, can be reduced to

$$T_p = k + (n-1)\,. \tag{2}$$

It is clear from Fig. 1 that when "Get Operands" receives a command, it will subsequently send that command to "Instruction Decode" and wait to receive a new command. Similarly, when "Instruction Decode" receives a command, it will translate and classify that command before sending a result to "Get Operands." This means that this technique can execute many commands at the same time. Nonetheless, some

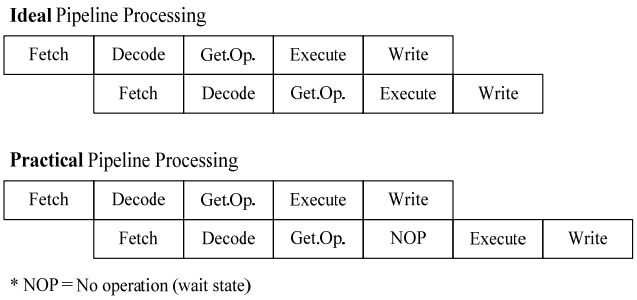

Fig.1. Pipeline processing technique in ideal mode and practical mode.

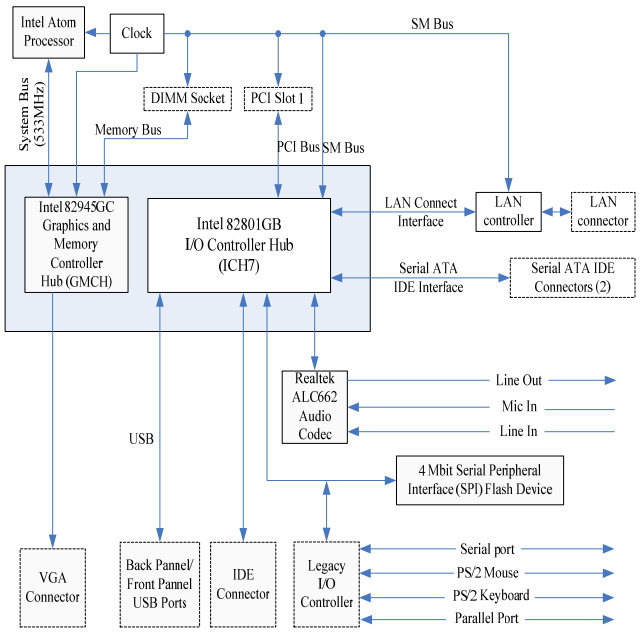

Fig. 2. A structure of Intel D945GCLF

some time slot might take longer than the others, especially during data processing. This results in "Wait State" in CPU, which slows down the processing speed.

It should be noted that the command that can help the pipeline technique perform effectively is a command that works independently from other commands, and it will not wait for a previous command to be the input of an executing command.

B. *Second Technique*

This second technique aims at managing communication channel. Normally, a computer system has many communication channels from I/O to CPU, as illustrated in Fig. 2. In this paper, we focus only on the I/O channel whose controller chip is ICH7. This ICH7 chip will control communication to the I/O devices, which can be divided into 8 channels, namely, USB, IDE, Audio, SATA, LAN, SMBUS, PCIBUS, and Legacy I/O controller [6].

In practice, if only one channel is used to connect "Source" and "Target," for data duplicating, it will cause a bottleneck in the channel. Therefore, if we could separate the channels (one for "Source" and the other for "Target"), this bottleneck problem might be solved.

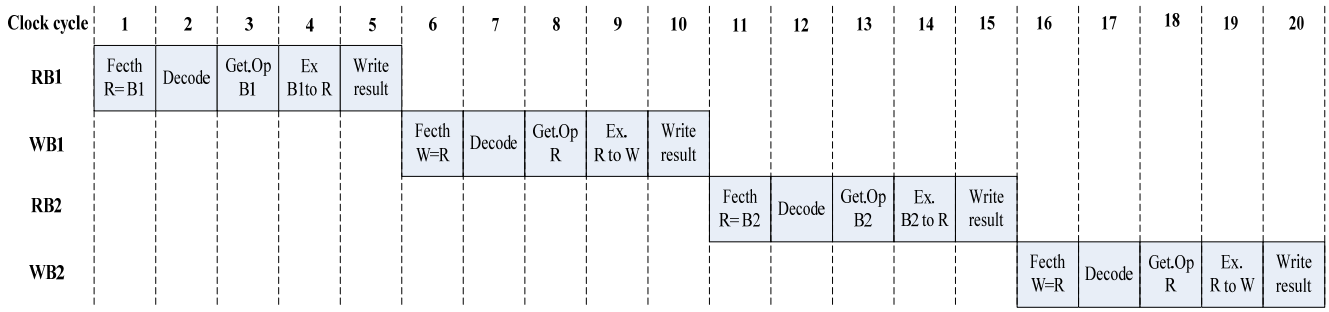

Fig. 4. The operation of the DD program that uses single processing

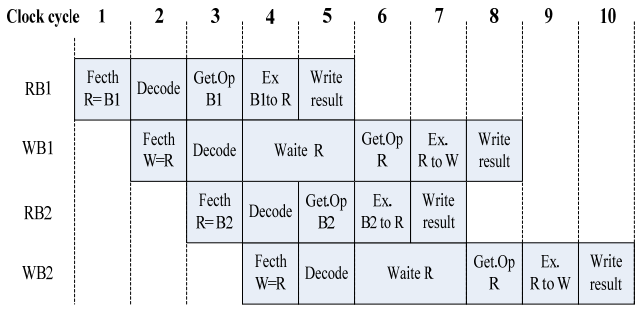

Fig. 3. The operation of the DD program that uses pipeline processing.

IV. Simulation Result

To demonstrate the proposed method, which performs the DD program based on pipeline processing, we first use the DD program to read data from the "Source." Then, the read data is sent to the second DD program via pipe. This second DD program will write the received data into the "Target."

Fig. 3 depicts the operation of the DD program that uses pipeline processing. It is apparent that there are 2 data blocks that are needed to be duplicated. The process involves two steps.

The first step is to read data from the "Source" (RBi) and the second step is to write the read data into the "Target" (WBi). Therefore, the total processing time for pipeline processing to duplicate these two data block is 10 clocks, whereas that for single processing to duplicate these two data block is 20 clocks, as shown in Fig. 4. This means that the proposed method can help reduce the processing time to duplicate data.

Fig. 5 displays the diagram on how the DD program works based on pipeline processing, which will be used to compare the performance of sequential processing and pipeline processing. In general, sequential processing is performed by a command line:

```
#DD if=/dev/source of=/dev/target
```

This means that the read procedure (`if=/dev/source`) and the write procedure (`of=/dev/target`) are in the same process, thus requiring the switching between the read procedure and the write procedure. However, Fig. 5 separates

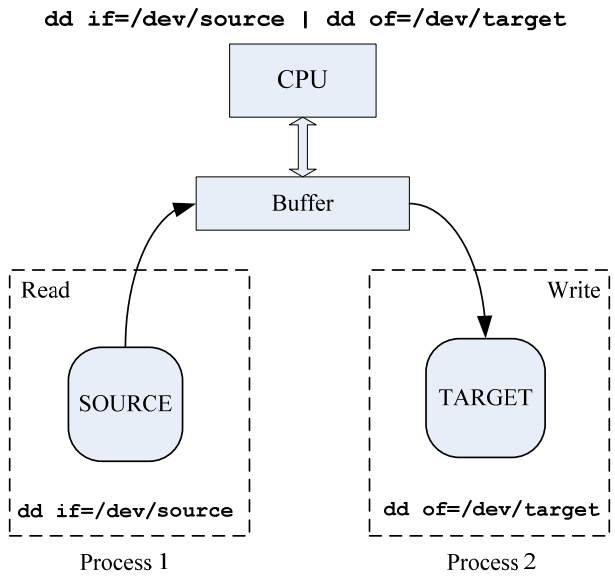

Fig. 5. How the DD program works based on pipeline processing.

these two procedures into 2 processes, and utilizes a pipe command to connect these two processes. Specifically, a pipe command will take the output from the first process to be the input of the second process.

Performance comparison between sequential and pipeline processing is given in Fig. 6. It is obvious that the DD program with pipeline processing has a transfer rate higher than that with sequential processing.

Next, we set up another experiment (using Intel D945GCLF CPU Intel Atom 1.6 GHz with RAM 2GB) to test the transfer rate between USB and USB, which can be divided into 4 cases, i.e.,

1) USB to USB (already come within a board)
2) USB (PCI) to USB (use a card to transform the standard port PCI to USB)
3) USB to USB (PCI) (use a card to transform the standard port PCI to USB)
4) USB (PCI) to USB (PCI) (use a card to transform the standard port PCI to USB)

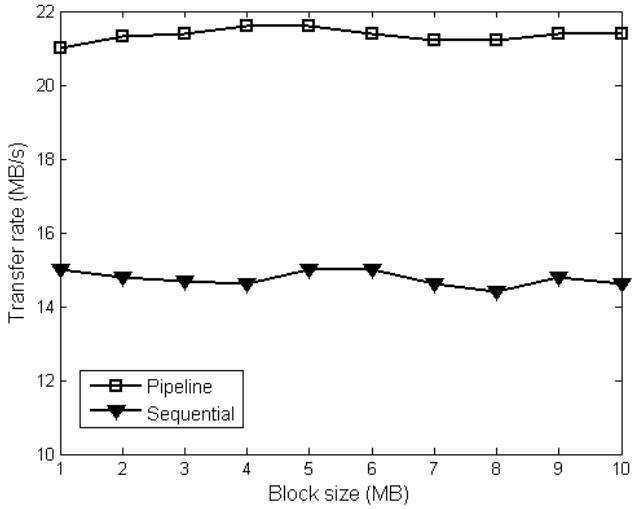

Fig. 6. Performance comparison between sequential and pipeline processing.

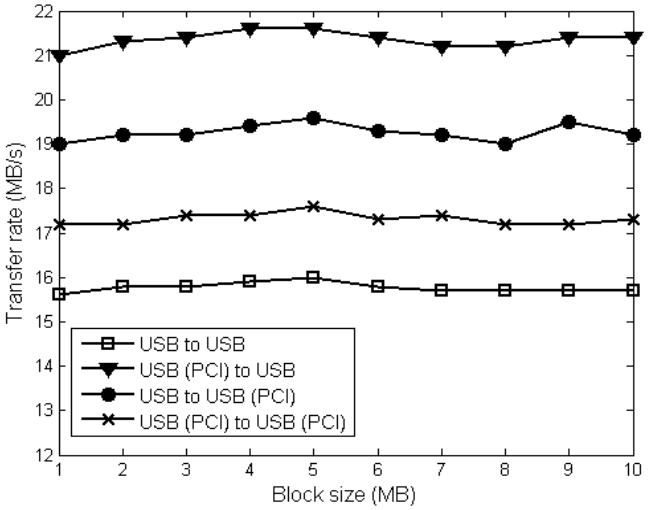

Fig. 7. Performance comparison with different connections.

TABLE I
TRANSFER RATE COMPARISON

Method	**Single** Processing (MB/s)	**Pipeline** Processing (MB/s)
1. USB to USB	10.7	15.7
2. USB (PCI) to USB	14.4	21.2
3. USB to USB (PCI)	13.0	19.0
4. USB (PCI) to USB (PCI)	11.7	17.2

Table 1 shows the transfer rate of different cases. Because case 2 and case 3 employ different communication channels for "Source" and "Target," they have a higher transfer rate than case 1 and case 2, which use the same channel to connect between "Source" and "Target." This result can be confirmed by plotting the transfer rate as a function of data block size, as illustrated in Fig. 7. It is apparent that using different communication channels for "Source" and "Target" can increase the transfer rate.

V. CONCLUSION

From the experiment, we found that of pipeline processing performs better than sequential processing, and can increase efficiency so as to provide good performance.

ACKNOWLEDGMENT

This work was supported by a research grant DSTAR-R&D 02-01-52 from I/UCRC in Data Storage Technology and Application Research Center (D*STAR), King Mongkut's Institute of Technology Ladkrabang, and National Electronics and Computer Technology Center (NECTEC), Thailand.

REFERENCES

[1] National Institute of Justice, "Test Results for Disk Imaging Tools: DD GNU fileutils 4.0.36, Provided with Red Hat Linux 7.1," 2002.

[2] T. Rude, DD and Computer Forensics – Deuce. Retrieved July 14, 2010, from http://www.crazytrain.com/dd2.html.

[3] W. Zhaohui, W. Xingang, and L. Pingping, "Application Research of EDBA Algorithm for Completing Hard Disk Copy," in *Proc. of Control and Decision Conference*, pp. 5834 – 5836, June 2009.

[4] K. Shibata and M. Yokoi, "Optimum Scheduling in Pipeline Processing," in Proc. of *ICCS/ISITA'92*, vol. 3, pp. 1078 – 1083, November 1992.

[5] William Stalling, *Computer Organization* & *Architecture*. 6th-edition, Pearson Education Indochina, 2003.

[6] Intel Cooperation, Intel® Desktop Board D945GCLF Technical Product Specification Manual. Retrieved on July 14 2010, from http://www.intel.com/support/motherboards/desktop/d945gclf/sb/CS-029163.htm