Multithread Content Based File Chunking System in CPU-GPGPU Heterogeneous Architecture

Zhi Tang
Department of Electronics and Computer Engineering
Hanyang University
Seoul Korea
tangzhi@hanyang.ac.kr

Youjip Won
Department of Electronics and Computer Engineering
Hanyang University
Seoul, Korea
yjwon@hanyang.ac.kr

***Abstract*—the fast development of Graphics Processing Unit (GPU) leads to the popularity of General-purpose usage of GPU (GPGPU). So far, most modern computers are CPU-GPGPU heterogeneous architecture and CPU is used as host processor. In this work, we promote a multithread file chunking prototype system, which is able to exploit the hardware organization of the CPU-GPGPU heterogeneous computer and determine which device should be used to chunk the file to accelerate the content based file chunking operation of deduplication. We built rules for the system to choose which device should be used to chunk file and also found the optimal choice of other related parameters of both CPU and GPGPU subsystem like segment size and block dimension. This prototype was implemented and tested. The result of using GTX460(336 cores) and Intel i5(four cores) shows that this system can increase the chunking speed 63% compared to using GPGPU alone and 80% compared to using CPU alone.**

Keywords-CPU-GPGPU; Content Based File Chunking; Deduplication; Incremental Modulo-K

I. INTRODUCTION

Accompany with the fast development of computing ability, memory space and I/O bandwidth of massive multi-core Graphics Processing Unit (GPU), the general-purpose usage of GPU (GPGPU) has become popular [1]. The huge number of cores in GPGPU (latest GTX590 contains 1024 cores) makes it efficient in processing large amount of data and providing high parallelism. However, some specific properties of GPGPU like memory hierarchy make it impossible to substitute CPU which leads to the conclusion that the CPU-GPGPU heterogeneous architecture will last for a long time. In a CPU-GPGPU heterogeneous architecture computer, CPU and GPGPU are integrated and CPU is used as the host processor.

In this work, we use the high computational ability of CPU-GPGPU architecture to accelerate the file chunking phase of deduplication. Deduplication eliminates file redundancy in block grained. Generally, it contains two phase: file chunking phase and redundancy detection phase. In file chunking phase, files are split into chunks and the way of splitting mainly includes fixed size chunking and content based chunking. Fixed size chunking is simple, fast but only suitable for files that barely updated. Content Based Chunking (CBC) is robust in detecting redundancy but it is computationally expensive [2] since judging the chunk boundary of a file takes a lot of computation. The computational complexity of CBC operation is one of the key issues that affect the performance of deduplication system.

In this work, we designed a Content Based File Chunking prototype system which includes a CPU chunking subsystem and GPGPU subsystem. This heterogeneous system can exploit the hardware organization of CPU-GPGPU architecture and decide which subsystem will be used to chunk a given file. Also, according to the information of the input file and hardware organization of the computer, the system is also able to set proper parameters to reflect the capabilities of both CPU and GPGPU device in the computer.

II. RELATED WORKS

Deduplication can be used in many types of file systems such as distributed and shared file system[3][4], backup[5], peer-to-peer file system[6], web-proxy server[7]. Won et al. found that chunking is one of the major overheads for deduplication process[2][8]. Meister et al. analyzed the deduplication efficiency under various chunking scheme.

The chunking speed of fixed size chunking is very fast but this method performs poor on finding redundancy of shifted data stream. CBC overcomes this shortage [9]. CBC is used in many application domains such as backups, file system and data transfers [4][5][12][13][14][15].

Rabin fingerprint algorithm [12] is widely used [4][5] to determine chunk boundary in content based chunking, because of its algorithmic simplicity. Rabin fingerprint is usually combined with Sliding Window Algorithm[4] to do content based chunking. To prevent the appearance of too small and too large chunks, minimum and maximum bound on chunk size are enforced [4]. Tang et al. introduced a chunk size control algorithm: Two Thresholds, Two divisors (TTTD), which sets a minimum and maximum boundary on chunk size and uses two divisors. TTTD was proven to have only 62.5% overhead of using Rabin fingerprint algorithm alone [10]. Youjip Won et al. developed the incremental Modulo-K algorithm in the PRUN system to simplify the computations needed for generating one signature which was proven to be 40% faster than the Rabin fingerprint algorithm while doing file chunking [2].

This work is sponsored by KOSEF through National Research Lab at Hanyang University (R0A-2007-000-20114-0).

978-0-7695-4528-8/11 $26.00 © 2011 IEEE
DOI 10.1109/CCP.2011.20

Although the PRUN system is proven to be efficient in chunking, it can not fully utilize the computing resources thoroughly as the hardware organization is changing all the time. The advancement of massive multicore processors makes multithread a way to improve the performance of content based chunking. In this paper, not only the multicore CPU, but also GPGPU will be brought into accelerating content based chunking.

We built a heterogeneous system which implements content based chunking by using either CPU or GPGPU in a CPU-GPGPU integrated computer. This system is mainly motivated by PRUN system, however, it can exploit the hardware organization of computer and coordinate CPU and GPGPU to do content based multithread file chunking. This system is also implemented and tested in this work.

The organization of this paper is as following, Section Ⅲ is about the system design and some related issues of this prototype system, Section Ⅳ is about the determination of which device would be used to chunk input files. Section Ⅴ shows the performance of this system, Section Ⅵ is the conclusion.

III. SYSTEM DESIGN

A. System Architecture

As mentioned above, this prototype system was designed to fully utilize computing resource of CPU-GPGPU heterogeneous system. Therefore, the first step of this system is to get the hardware information of the computer and then based on this, determining whether CPU or GPGPU will be used to chunk file. The way of how to decide whether CPU or GPU would be used will be discussed later.

The architecture of this system will be shown in Fig. 1. As mentioned in Section Ⅰ, this system includes both CPU subsystem and GPGPU subsystem. After either CPU subsystem or GPGPU subsystem has been chosen, file chunking would start. The input file will be separated into several segments and assigned to several threads to be chunked in parallel in both CPU and GPU.

As shown in Fig. 1, the main difference between these two subsystems is: in CPU subsystem, every thread will be in charge of chunking several segments; and in GPGPU subsystem, one thread will be responsible of chunking only one segment. This difference is caused by the different threads switching schemes of CPU and GPGPU.

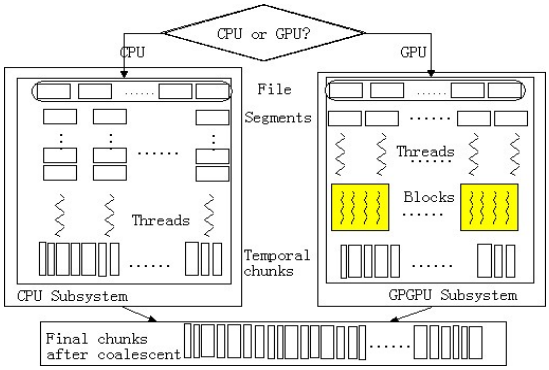


Fig. 1 System architecture

Thread switching in CPU includes the suspending of the current thread, saving its state (e.g., registers), and then restoring the state of the thread switched to which cause a lot of switching overhead in system. However, threads in GPGPU are lightweight and almost have no switching overhead. Threads in GPGPU will be grouped into warps to schedule. Every warp contains 32 threads. Since the memory instructions in GPGPU, especially global memory instructions, have big overhead, the instant switching of warps can be used to hide memory instruction latency. Once the GPGPU meets memory instruction, it switches to other warps to execute their arithmetical instructions. Fig. 2 shows the GPGPU warp scheduler.

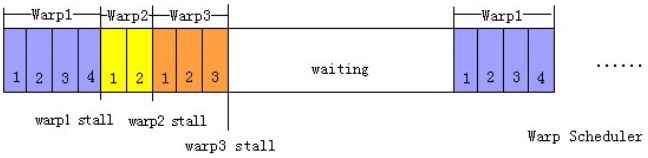


Fig. 2 GPGPU warp scheduler

Because of these two different thread switching schemes, number of threads in CPU subsystem should be equal to the number of CPU cores and number of threads in GPGPU subsystem should be as much as possible. Meanwhile, the biggest number of threads in GPGPU subsystem is the number of segments of the input file. Therefore, in GPGPU subsystem, the number of threads should be equal to number of segments and every thread chunks one segment.

B. Chunking Algorithm

The chunking algorithm used in this system is the same as the PRUN system. Generally, The chunking algorithm of both CPU and GPGPU subsystem are the same as PRUNE, the incremental Modulo-K algorithm combining with the BSW and with minimum and maximum bound on chunk size.

However, the key issue of a multithread chunking system is making sure no matter what segment size and degree of multithread will be used, the output of chunking a same file must be the same. This is to make sure that if different client machines are backing up a same file, the server will store same chunks for this file.

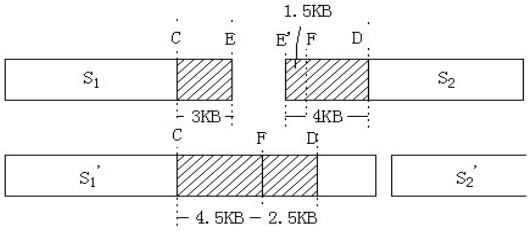


Fig. 3 Chunking variability

As shown in Fig.3, S_1 and S_2 are two neighbored segments with size 50Kbyte and S_1' and S_2' are the same piece of file but separated into two 60Kbyte segments. Suppose this piece of file has three chunk points C, F and D, as shown in Fig. 3. If using two threads to chunk segments S1 and S2, there will be two chunks CE and E'D with size 3Kbyte and 4Kbyte since chunk point F will be ignored because the first 2Kbyte of S_2 was jumped by using minimum bound on chunk size. However, if using two threads to chunk 60Kbyte segments S_1' and S_2', the same piece of file will have two chunks CF and FD. This is one possible chunking variability problem.

To prevent the appearance of chunking variability, a dual mode chunking scheme was used in this work. Dual mode chunking scheme contains two chunking mode, bare mode which chunks file without minimum and maximum boundaries and accelerate mode which chunks file with minimum and maximum boundaries. Besides the first segment of a file, threads start chunking segments by using bare mode, then after the first appearance of a chunk whose size is in [minimum, maximum–minimum] and not the first chunk of this segment, the accelerate mode will be switched.

C. Coalescent

The dual mode scheme successfully prevents the happening of the situation in which chunk points within the minimum bound of the beginning of segment will be ignored. However, it might produce chunks whose size is bigger than the maximum bound or smaller than the minimum bound. What's more, it still can not prevent the chunking variability problem. As shown in Fig. 3, if using dual mode scheme, S_1 and S_2 would have chunks CE, E'F and FD while S_1' and S_2' would still have chunks CF and FD. This is caused by the different ending points if using different segment sizes.

To make sure chunk variability will not happen, the temporal output of all threads need to be coalesced. The coalescing of chunks has 2 steps as shown in Fig. 4. First, merge a chunk whose size is less than the maximum bound with the next chunk to check whether the size of the new chunk is bigger than maximum bound. If not, the new chunk will be put into the final chunk list. Else, split it into two parts, one with size of maximum size, put into the final chunk list and the other one will be put into the temporal list again. Secondly, for chunks whose sizes are bigger than the maximum bound, split them into two chunks immediately.

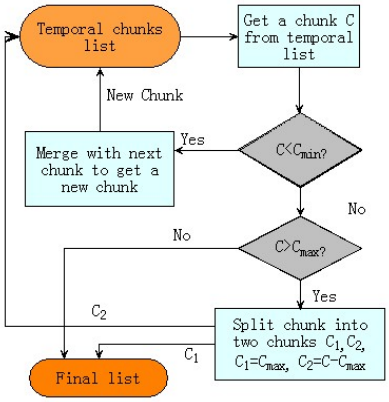


Fig. 4 Coalescent

IV. CHOOSING DEVICE

The main difficulty in this heterogeneous system is to find the rules of choosing a device between the CPU and GPGPU to do the chunking related computations. To find the rules of determining which device would be used, we need to find the elements that affect the chunking performance in this system. We consider the elements that affect the chunking performance of CPU and GPGPU subsystem separately.

A. CPU chunking subsystem

Firstly, we consider the CPU chunking subsystem. We implemented the CPU chunking subsystem and tested it to find the elements that affect the performance of CPU chunking subsystem.

A computer integrates both multicore CPU and GPU was used here. The hardware specification of this computer A is shown in Table 1 and Table 2 contains the data sets that will be used to test this system.

Table 1 Hardware specification of Computer A

CPU	i5 760(four cores)	2.66GHZ
GPGPU	GTX460(336 cores)	1350 MHz 1GB Global memory
Storage	3TB(RAID 0)	380MB/Sec
RAM	DDR3	2GB

Table 2 Data set

Name	Type	Description
A	rmvb	Single file, size range from 10Kbyte to 512MByte
B	iso, rar avi, zip jpg, exe cab, pdf other	Real data set from the backup server of lab, contains 487 files in total, the total size is 2.929GByte, average size is 6.015MByte

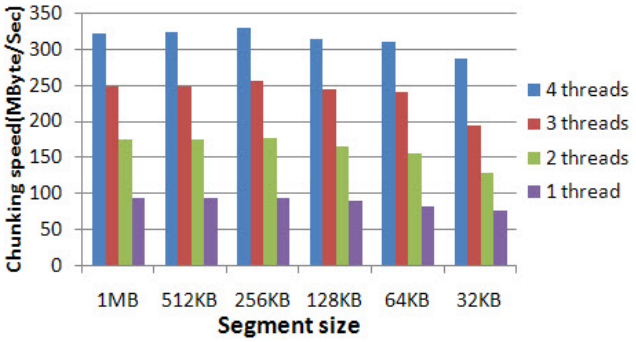
It's clear that number of CPU cores is an element that affects the chunking speed of CPU subsystem since it directly affects the thread degree. In order to test the effects of thread degree, the number of threads will be set to be 1, 2, 3, 4 separately to chunk a 256MByte file in data set A of Table 2. And the segment size ranges from 32Kbyte to

1Mbyte to show the effect of segment size. The result was shown in Fig.5 (a).

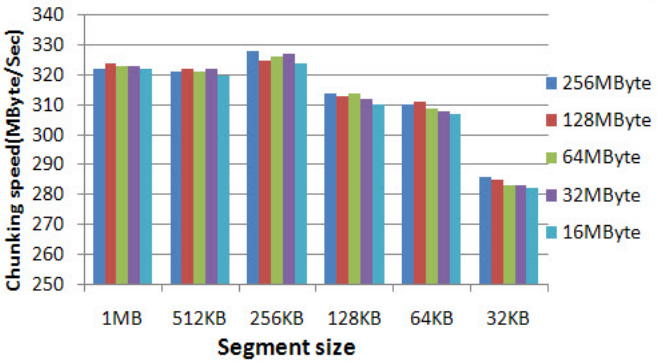
Meanwhile, since size of real files ranges from several bytes to thousands of Mega bytes, we need to test the effect of file size too. As shown in Fig.5 (b), 16Mbyte, 32Mbyte, 64Mbyte, 128Mbyte and 256Mbyte files would be chunked under different segment sizes by four threads.

Chunking speed of using CPU subsystem alone shows that file size doesn't have big influence on the chunking speed. For different file sizes, the chunking speed is almost stable with same segment size and threads number. However, number of threads does affect the chunking speed greatly. Also, segment size influences the chunking speed a lot, if the segment size is 32Kbyte, its chunking speed of using 4 threads is only 280Mbyte/Sec while 256Kbyte segment size has nearly 330Mbyte/Sec chunking speed.

To get the highest chunking speed, obviously, the degree of threads should be equal to the number of CPU cores. It will not have thread switching latency and utilize the computing capability of CPU thoroughly. Therefore, segment size and number of CPU cores are elements that affect the chunking speed of CPU chunking subsystem.



(a) Chunking 256MByte file

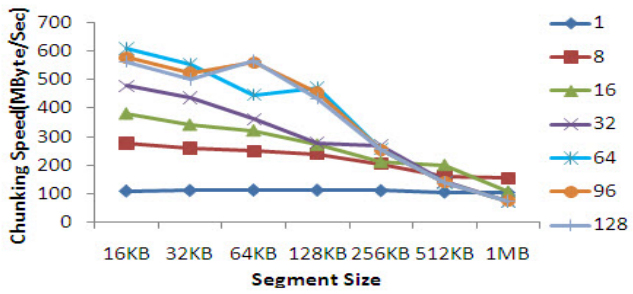


(b) Chunking different files with 4 threads

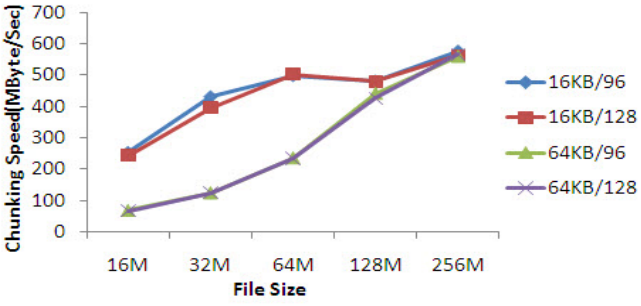
Fig. 5 Chunking speed of CPU subsystem

B. *GPGPU chunking subsystem*

Then the elements that affect the chunking speed of GPGPU chunking subsystem would be considered. As we used Nvidia graphics card and CUDA platform, number of GPGPU cores, the block dimension which is number of threads and segment size might be the elements affect the performance of GPGPU chunking subsystem. We also used the computer A in Table 1 to test the effect of these three elements. Files used here are the same as used in section Ⅳ-A to test CPU subsystem.



(a) file size = 256MByte



(b) Comparison

Fig. 6 Chunking speed of GPGPU subsystem

Firstly, we used different segment size and block dimension which means number of threads per block to chunk a 256MByte individual file. The chunking speed result was shown in Fig. 6a. Besides block dimension is one, the chunking speed of GPGPU subsystem varies greatly as different segment size and block dimension are used.

The block dimension 64 and segment size 16Kbyte produce highest chunking speed of chunking this 256Mbyte file. However, the chunking speed of block dimension 64 vibrates greatly as the segment size varies. Meanwhile, we found that block dimension 96, 128 threads/block produce comparable stable chunking speed as the segment size ranges from 16Kbyte to 64Kbyte, and also the chunking speed of segment size 16Kbyte and 64Kbyte are the segment size that have highest chunking speed.

Therefore, we use four combinations of 96, 128 block dimension and segment size 16Kbyte, 64Kbyte which are 16Kbyte/96, 16Kbyte/128, 64Kbyte/96, 64Kbyte/128 respectively to chunk files with different sizes which ranges from 16Mbyte to 256Mbyte. The result was shown in Fig. 6b. The result shows that 16Kbyte performs better than 64Kbyte segment size. What's more, Fig. 6b shows that the chunking speed of GPGPU varies greatly as the size of files vary.

We can get that file size, block dimension, segment size directly affect the chunking performance of GPGPU subsystem. Block dimension is dependent on file size, segment size and GPGPU cores, therefore, number of GPGPU cores, file size and segment size are elements that affect the performance of GPGPU subsystem.

C. *Determing rules*

As shown in sections Ⅳ-A and Ⅳ-B, elements involved in determining the performance of two subsystems including, number of CPU cores, number of GPGPU cores, segment size and file size. Among these four facts, segment size is flexible and decided by the user, also, the optimal value of segment size in CPU and GPGPU subsystem is different. Therefore, if we use a function f to describe the rules that which device should be chosen to chunk files, it contains number of CPU cores, number of GPGPU cores and file size as parameters. After whether CPU or GPGPU will be used is determined, the value of segment size can be determined by the subsystem itself. f can be written as in (1).

$$f(C,G,F) \tag{1}$$

C,G,F in (1) stand for number of CPU cores, number of GPGPU cores and file size separately. To study the influence of these 3 parameters more accurately, some more experiments need to be carried out. Also a new computer B with different devices integrated as shown in Table 3 will be introduced in this section to do experiments.

Table 3 Hardware specification of Computer B

CPU	Intel Core E7500 (2 cores)	2.93GHZ
GPGPU	9600GT(64 cores)	1500 MHz 512MB Global memory
Storage	320GB	136MB/Sec
RAM	DDR2	2GB

To show the influence of file size clearly, use Intel i5, Intel E7500, 9600GT and GTX460 to chunk files with different size and then write down the highest chunking speed of all devices. The result was shown in Fig. 7.

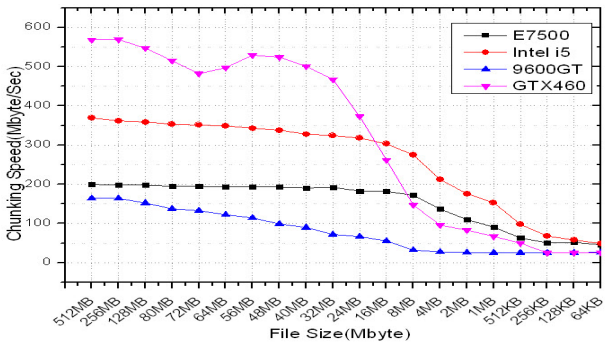


Fig. 7 Chunking speed of different devices

Fig.7 shows clearly that no matter what CPU and GPGPU are installed in one computer, files smaller than 8MByte should be chunked by CPU since even 336 cores GPGPU performs poorer than a two cores CPU. For files bigger than 8MByte, both the number of cores of CPU and GPGPU need to be considered. From Fig. 5, a simple conclusion that one thread of CPU subsystem can get nearly 90MByte/Sec chunking speed can be achieved. Also from Fig. 6, for the optimal choice of segment size and block dimension, one GPGPU core can generally have a highest chunking speed of 2Mbyte/Sec. Therefore, Equation 2 can be derived. Also results in Fig. 7 can help proving Equation 2. While doing content based chunking, the two cores CPU is more efficient than 64 cores 9600GT while 336 cores GTX460 is more efficient than 4 cores CPU.

$$(C*90) >= (G*2) = \begin{cases} \text{true} & \text{use CPU} \\ \text{false} & \text{use GPGPU} \end{cases} \tag{2}$$

If CPU was used, the only thing left is to determine the number of threads and segment size. And the number of threads of course should be the number of CPU cores. From Fig. 5, it's easy to see that when file size is equal to or bigger than 16Mbyte, the size of segment should be between 128Kbyte and 256Kbyte which produces the highest chunking speed for all choices of threads number. For files smaller than 16MByte, the CPU subsystem will be tested again (Intel i5 used) to get the optimal segment size. The result was shown in Fig. 8. Files between 16Mbyte and 8Mbyte should still have 256Kbyte as segment size while files smaller than 8Mbyte should use 64Kbyte as segment size.

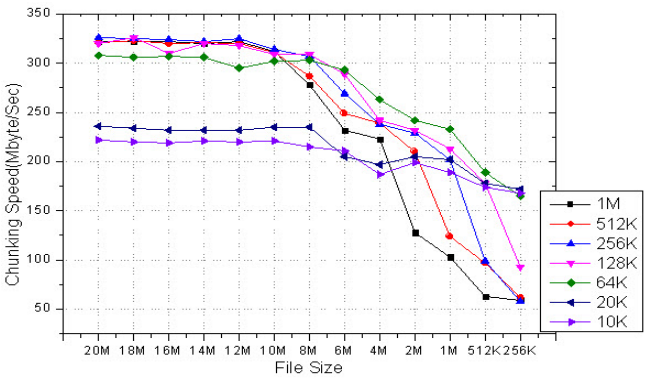
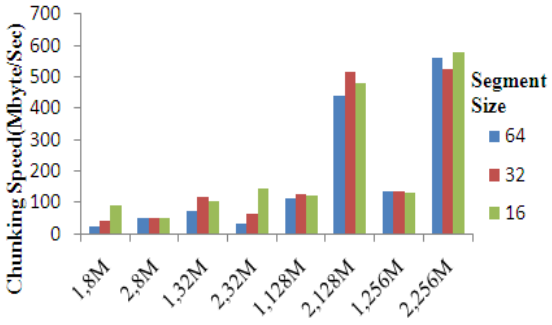


Fig. 8 Segment test for CPU subsystem

If GPGPU subsystem was used, besides segment size, block dimension also need to be determined by using file size and number of GPGPU cores. From Fig. 6, segment size 16Kbyte has best performance which means the GTX460 device can use 16Kbyte as the segment size while chunking files. However, the segment size that might produce the highest chunking speed might differ from the GPGPU cores one GPGPU device can have. Therefore, we test the 9600GT card and then compare to the GTX460 card.

We test the chunking speed of using GPGPU subsystem to chunk individual files in Data set A under segment size 16Kbyte, 32Kbyte and 64Kbyte and block dimension 96 and 128. The result was shown in Fig. 9. As shown in Fig. 9, the x axis stands for the device type and size of file it chunked. 1 stands for the 9600GT card while 2 stands for GTX460, i.e, 1,8M means using 9600GT card to chunk a 8Mbyte file.

Although the chunking speed varies according to the block dimension and segment size, generally, for both cards, segment size of 64Kbyte and the block dimension 128 threads/block have the best chunking speed for almost all situations. Therefore, these two values would be used in the GPGPU subsystem. The detail of determining the best segment size and block dimension is out of scope of this paper which needs to consider the programming model, hardware organization of the GPGPU device.



(b) Block dimension=128Kbyte

Fig. 9 Chunking speed of two different GPGPU devices

V. EXPERIMENT

So far, the rules of determining whether CPU or GPGPU subsystem should be used are determined. Also the values of best segment size of using both CPU and GPGPU subsystem and the optimal block dimension of GPGPU system have been found. Then the performance of the overall system should be tested. Computer A in Table 1 was used to test the performance of the overall system.

Firstly, files in data set A of Table 2 are used to test the system. Besides testing the overall system, CPU and GPGPU subsystem will be tested as comparison. Fig. 10 shows the result of chunking single file in data set A. The result shows that the overall system can reflect the best performance of both CPU and GPGPU. Then data set B in Table 2 is used to test the system again to find out how this heterogeneous system works on real world.

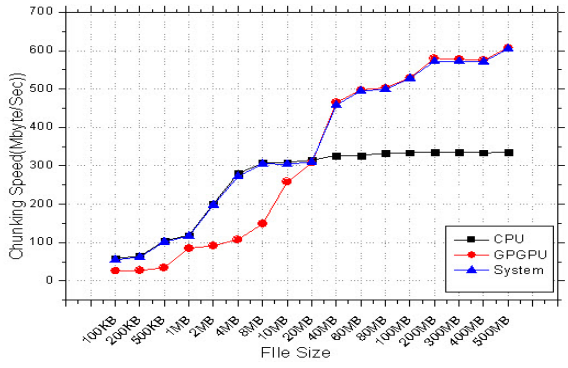
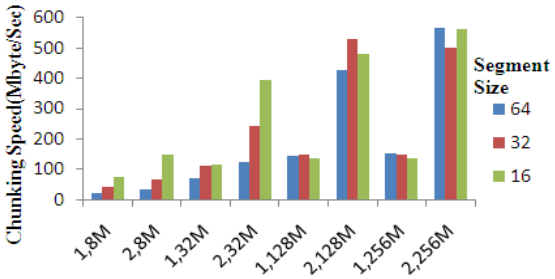


Fig. 10 Performance of chunking single file



(a) Block dimension = 96

The segment size is set to be 256Kbyte for CPU and 64Kbyte segment size, 128threads/block is set as block dimension for GPGPU. The chunking speed of using CPU only is 218Mbyte/Sec, using GPGPU only is 242Mbyte/Sec and the heterogeneous system reaches 393MByte/Sec. This system can increase the chunking speed 63% compared to use GPGPU alone and 80% compared to use CPU alone.

VI. CONCLUSION

In this work, we built a multithread file chunking system in CPU-GPGPU heterogeneous architecture system to coordinate CPU and GPGPU devices to accelerate the content based file chunking operation of deduplication system. This system can exploit the hardware information of computer and combining it with the choosing rule to choose the device that has faster chunking speed. At last, a real data set achieved from the lab server was used to test how this system works with real data. Using the devices GTX460(336 cores) and Intel i5(four cores), the system successfully shows that it can increase the chunking speed 63% compared to use GPGPU alone and 80% compared to use CPU alone.

REFERENCES

[1] Owens, J. D., D. Luebke, et al. (2007). "A Survey of General-Purpose Computation on Graphics Hafrdware. " Computer Graphics Forum 26(1): 80-113.

[2] Youjip Won, Rakie Kim, Jongmyeong Ban et al., "PRUN: Eliminating Information Redundancy for Large Scale Data Backup System," iccsa, pp.139-144, 2008.

[3] B. Hong, D. Plantenberg, D. Long, and M. Sivan-Zimet, "Duplicate data elimination in a SAN file system," in Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004). Citeseer, 2004, pp. 301–314.

[4] A. Muthitacharoen, B. Chen, and D. Mazi`eres, "A low-bandwidth network file system," SIGOPS Oper. Syst. Rev., vol. 35, no. 5, pp. 174–187, 2001.

[5] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.

[6] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," LECTURE NOTES IN COMPUTER SCIENCE, pp. 21–40, 2003.

[7] M. Mitzenmacher, "Compressed bloom filters," IEEE/ACM Trans. on Networking, vol. 10, no. 5, pp. 604–612, October 2002.

[8] Youjip, W., B. Jongmyeong, et al. (2008). "Efficient index lookup for De-duplication backup system. " Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008.

[9] Dirk, M., Andr, et al. (2009). "Multi-level comparison of data deduplication in a backup scenario. " Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference. Haifa, Israel, ACM.

[10] Kave Eshghi, H. K. T. (September 22, 2005). "A Framework for Analyzing and Improving Content-Based Chunking Algorithms."

[11] M. O. Rabin, “Fingerprinting by random polynomials,” 1981.

[12] Neil, T. S. and W. David (2000). "A protocol-independent technique for eliminating redundant network traffic." SIGCOMM Comput. Commun. Rev. 30(4): 87-95.

[13] Jeffery, C. M., C. Yee Man, et al. (2004). "Design, implementation, and evaluation of duplicate transfer detection in HTTP. " Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1. San Francisco, California, USENIX Association.

[14] Himabindu Pucha, D. G. A., Michael Kaminsky (April 2007). Exploiting Similarity for Multi-Source Downloads Using File Handprints. 4th Symposium on Networked System Design and Implementation (NSDI '07), Cambridge, MA.

[15] Landon, P. C., D. M. Christopher, et al. (2002). "Pastiche: making backup cheap and easy." SIGOPS Oper. Syst. Rev. 36(SI): 285-298