A CPU-GPGPU Based Multithread File Chunking System

Zhi Tang* Youjip Won
Department of Electronics and Computer Engineering, Hanyang University

Abstract
The popularity of general purpose GPU (GPGPU) makes the CPU-GPGPU heterogeneous architecture normal. Therefore, tradeoff the usage of CPU and GPGPU becomes a way to improve performance of programs. In this work, we exploit the properties of the CPU-GPGPU heterogeneous architecture and use them to accelerate the content based chunking operation of deduplication. We built a prototype system which is able to coordinate CPU and GPGPU to chunk file and has been proven to have a better performance compared to using either CPU or GPGPU alone.

1. Introduction
Accompany with the fast development GPU, the research of using GPU to do general purpose usage becomes popular [1]. However, properties of GPGPU restrict its substitution of CPU which leads to the CPU-GPGPU heterogeneous architecture.

In this work we studied how to utilize the CPU-GPGPU heterogeneous architecture to accelerate the file chunking phase of deduplication. Won et al. found that file chunking is one of the major overheads for deduplication process [2][3]. The way of doing file chunking method usually includes content based chunking and fixed size chunking. Meister et al. analyzed these two chunking scheme. The fixed size chunking is fast but performs poor on finding redundancy of shifted data stream; however, content based chunking is able to overcome this shortage [4].

Rabin fingerprint algorithm[5] and the Basic Sliding Window Algorithm(BSW)[6] are widely used[6][7] to determine chunk boundary in content based chunking. Also, to prevent too small and too large chunks, minimum and maximum bound on chunk size are enforced[5]. Youjip Won et al. developed a PRUN system which uses the incremental Modulo-K(INC-K) algorithm to find chunk boundary and was proven to be 40% faster than the Rabin fingerprint algorithm[1].

Motivated by PRUN, this work built a multithread file chunking system which implements content based chunking by either CPU or GPGPU in CPU-GPGPU heterogeneous architecture. This paper is organized as following: system design in section 2, experimental results in section 3 and section 4 is the conclusion.

2. System Design
2.1 System Architecture
As shown in Fig.1, this system contains 4 modules. The hardware detection and segment allocation module gets information of computer and input file separately, and then use them to decide whether CPU or GPGPU will be used in chunking module. But how to decide whether using CPU or GPGPU will not be discussed here for space limitation.

Fig. 1 System Architecture  Fig. 2 Chunking Module
The output of chunking module is the sequential combination of all threads’ output. Coalescent module gathers and coalesces them. Chunks whose size is less than the minimum bound will be merged into next chunk and chunks whose size is bigger than the maximum bound will be split into two chunks, one with size of maximum bound and the left part is the other.

2.2 Chunking Module
As shown in Fig. 2, chunking module contains two separate subsystems. In CPU subsystem, every thread needs to chunk several segments while in the GPGPU subsystem, every thread chunks one segment. This difference comes from the different thread switch schemes of CPU and GPGPU.
The thread switching overhead in CPU is much heavier compares to thread switching in GPGPU. And thread switching is usually used to hide the overhead of memory operation in GPGPU. So the number of threads in CPU subsystem should equal to the number of CPU cores and number of threads in GPGPU subsystem should be as much as possible which equals to the number of segments of the input file.

Generally, segment size 200Kbyte is best for CPU. 64Kbyte segment size and 128 threads per block is best for GPGPU.

2.3 Chunking Algorithm

The chunking algorithm is shown in Fig. 3. Window size is 48byte and minimum and maximum bound are 2Kbyte and 64Kbyte separately as same as PRUN.

To prevent the chunking variability problem, a dual mode chunking scheme will be introduced which contains two chunking mode. Bare mode which chunks file without minimum and maximum bound and accelerate mode which chunks file with minimum and maximum boundaries.

Besides the first segment of a file, all other segments are chunked as shown in Fig. 3. At first, the bare mode will be used which slides the window (48byte) from the beginning of segment, generates signature of the window region and then compares the signature with a predefined signature. If equal, the end position of the window is a chunk boundary. Then, after the appearance of a chunk whose size is in [minimum, maximum–minimum] and not the first chunk, slides the window from the minimum bound.

![Fig. 3 Chunking algorithm](image)

3. Experiment

A computer integrates an Intel i5 CPU(4 cores) and Nvidia GTX460 GPGPU(336 cores) was used to test this system. Firstly, chunk single rmmv files with different sizes using the system, CPU only, GPGPU only separately. The segment size and number of threads per block of using CPU or GPGPU alone are set as same as the heterogeneous system. Fig. 4 shows that this system can successfully reflect the best performance of both CPU and GPGPU.

Then chunk real data set to test how the system works in real world. Data set is achieved from the backup server of lab, it contains 487 files and the average file size is 6.015MByte. File type includes iso, rar, avi, zip, jpg, exe, cab, pdf and others. The segment size and number of threads per block are set to be the value mentioned in section 2.2. The chunking speed of using CPU only is 223MByte/Sec, using GPGPU only is 242MByte/Sec and the heteroge–neous system reaches 398MByte/Sec. Our system shows 64% increase comparing to use GPGPU only and 78% increase comparing to use CPU only.

4. Conclusion

In this work, we built a multithread file chunking system which exploits the information of CPU–GPGPU heterogeneous architecture and coordinates them to chunk file. Through using Intel i5 CPU and GTX460 GPGPU to implement this system, it shows 64% increase compared to use GPGPU alone and 78% increase compared to use CPU alone.

Reference