International Journal of Multimedia and Ubiquitous Engineering
Vol.11, No.12 (2016), pp.161-170
http://dx.doi.org/10.14257/ijmue.2016.11.12.15

# Performance Comparison of Asynchronous Transfer Configurations for UHD Game Image Compression with GPGPU

Youngsik Kim

*Department of Game and Multimedia Engineering, Korea Polytechnic University*
*E-mail: kys@kpu.ac.kr*

***Abstract***

*Ultra high definition (UHD) game scenes have caused the memory bandwidth problem. The lossless DPCM-GR based compression algorithm [12] using NVIDIA CUDA(Compute Unified Device Architecture) like general purpose GPU (GPGPU) computing relieves the bandwidth problem without sacrificing image quality, which supports bit parallel pipelining. This paper increases the memory bandwidth efficiency using the shared memory of CUDA based on the compression algorithm [12]. Also, various asynchronous transfer configurations which can overlap the kernel execution and data transfer between the Host and the CUDA device are implemented with the page-locked host memory. Experimental results show that GPGPU CUDA computing obtains the maximum 87.5 and 30.6 times speedups for GTX650Ti and GT330, respectively, comparing to Host CPU. Also, the maximum reductions of the compression time for GTX650Ti and GT330 are 54.1% and 30.3%, respectively, among various concurrency transfer configurations.*

***Keywords****: GPGPU (General Purpose GPU), NVIDIA CUDA (Compute Unified Device Architecture), Lossless Compression, Asynchronous Transfer, DPCM-GR (Differential Pulse Code Modulation – Golomb Rice), concurrency transfer configurations*

## 1. Introduction

In the latest computer games, the display resolution has been rapidly increasing from HD(1920x1080) to UHD(4096x2160, 7680x4320) for various digital devices such as large screen monitors, digital TVs, and smartphones. The huge memory bandwidth has been needed to meet the UHD resolution of the game scenes on those platforms. The game image compression has been studied to reduce the memory bandwidth for UHD display. In order to prevent deterioration of image quality, in general, the lossless compression is suitable for the compression of UHD game image.

Instead of the image quality degradation, the compression ratio of the lossless compression algorithm is not fixed because of the variable-length compressed data. Since the lossy compression algorithm has the fixed compression ratio, the memory access pattern is simple and the bit parallel processing is easy. To apply lossy compressions, however, there are constrains that must be visually lossless. Both the memory interface and pit parallelism should be addressed in order to use the variable-length data by applying lossless compressions. A lot of hardware architectures for the lossless compression are studied in [1-6].

Many studies using NVIDIA CUDA [7] like GPGPU computing have been conducted to make the large-scale processors inside in GPU performed in parallel [8-11]. In [8], to compute the summed area table, the given memory bandwidth is efficiently used by dividing the input data into a sub-image of a square where the parameters are spread and by reducing the amount of global memory accesses by almost half. The optimization technique was proposed to accelerate the color format conversion of the non-memory-

ISSN: 1975-0080 IJMUE

aligned webcam video using CUDA in [9]. The memory for accelerating the generation of the maximum-minimum octree using CUDA was proposed in [10]. Real-time DXT(DirectX Texture) compression techniques was proposed by utilizing the CUDA programming in [11]. However, since the DXT compression algorithm is lossy, the image degradation cannot be prevented. In the previous work [12], the author proposed the lossless compression of DPCM-GR (Differential Pulse Code Modulation – Golomb Rice) by modifying the DDPCM-GR compression algorithm [4] which can support bit-parallel processing, in order to resolve the memory bandwidth issue for the UHD (4096x2160) game image.

In this paper, the CUDA-based lossless image compression technique for the UHD game is proposed based on the DPCM-GR algorithm [12] which can prevent the image degradation. For this, the memory efficiency is considered through the use of shared memory of CUDA. For the performance optimization, also, various asynchronous transfer configurations which can overlap the kernel execution and data transfer between Host and CUDA are implemented with the page-locked host memory. Experimental results evaluate the asynchronous transfer configurations for the UHD game image compression.

## 2. Related Works for Lossless Image Compression

HMD-ExpG (Hierarchical Minimum and Difference + Exponential Golomb) algorithm [1] computes both the minimum value and its difference values with other pixels for 2x2, 4x4, 8x8 hierarchical blocks, respectively, for the prediction. Then the exponential Golomb coding is performed for entropy coding. It has a 64-bit quantized table address to enhance the compression ratio. The advantage is that it considers the random access and the compression ratio of 2.1:1 is relatively high. But the shortcoming is that it has the table overhead (6.1KB/frame) and the hardware latency for exponential Golomb coding, 12 cycles, is high. Furthermore, the method to obtain the minimum value for hierarchically calculating the difference is difficult to operate at a high speed bit parallel pipelines and the hardware implementation is difficult to support the burst access.

To reduce the decoding latency of HMD-ExpG algorithm [1], HMD-NBS (HMD + Nonzero Bit Selection) algorithm [2] uses the same prediction method of HMD-ExpG However, it uses the different NBS method for entropy coding. It divides the block having the difference value into five groups and obtains the minimum number of bits to represent the difference between the minimum value per each group. Then the starting position of each difference value group can be seen in advance. The advantage of this approach is to reduce the decoding cycles 3 cycles. Still, it is difficult to operate at a high speed pipeline and having a bit parallel because of the table overhead in HMD-ExpG [1].

HACP-SBT (Hierarchical Average Copy Prediction + Significant Bit Truncation) algorithm [3] uses five ways like horizontal average prediction, vertical average prediction, horizontal direct prediction, vertical direct prediction, and four-pixel average prediction for the prediction. And for the entropy coding, it uses SBT scheme similar to the HMD-NBS algorithm [2] which includes the minimum number of bits for each group. Since it has the high compression ratio of 2.56:1, the memory bandwidth reduction can be 61%. However, it cannot support the random access. Also, it is difficult to implement the pipelined bit-parallel hardware because of the encoding cycle overhead and the complex prediction.

DDPCM-GR algorithm (Differential Differential Pulse Code Modulation + Golomb-Rice) [4] uses DDPCM [5-6] devised in ATI for the prediction and Golomb-Rice for entropy coding. The advantage of this compression algorithm can support the fully pipelined bit-parallel hardware structure using a simple prediction method. It provides the two-cycle encoding/decoding in pipelined fashion. The compression ratio is 1.64:1 for the 8x8 block.

 

The DDPCM [5-6] devised in ATI is for the Z (depth) data compression in 3D graphics. It is assumed that the Z depth value is linearly interpolated in DDPCM. The depth value is distinguished as -1, 0, 1, and an escape value after DDPCM. Then it performs entropy coding by mapping 0, 1, -1, and an escape value into 00, 01, 10, and 11, respectively. But, since DDPCM is originally devised to compress the Z data, a simpler DPCM (differential pulse code modulation) can be applied without the degradation of the compression ratio for the Game scene color pixels. In the previous work [12], the author proposed the lossless compression of DPCM-GR by modifying the DDPCM-GR compression algorithm [4] to support the bit-parallel processing.

## 3. DPCM-GR Lossless Image Compression

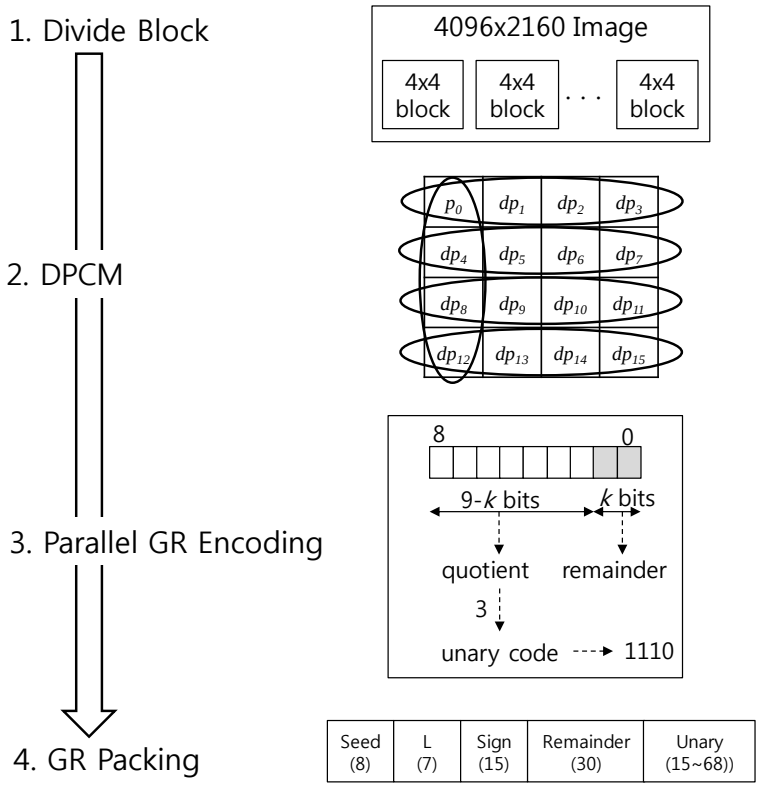


**Figure 1. DPCM-GR Lossless Compression Algorithm [12]**

This section describes the DPCM-GR lossless compression algorithm [12] by modifying the DDPCM-GR algorithm [4] to support the bit-parallel processing. As in Figure 1, the full lossless compression algorithms for the UHD game image consists of major four steps like (1) dividing the original image data into 4 x 4 blocks, (2) DPCM calculated for the data block, (3) Golomb-Rice parallel encoding, and (4) GR Packing. To calculate DPCM, one seed data and fifteen difference data are derived for each data block. By applying the GR bit parallel algorithm to the fifteen difference data, overall, it derives a lossless data compression.

The seed data is the uncompressed original data. The DPCM difference data is divided by $2^k$ for GR encoding. In this paper, the GR parameter, k, is assumed to be 2. Then, the remainder portion is a 15 x 2 = 30 bits as a fixed bit number. And fifteen quotients are represented in unary code. For example, the quotient value of 3 can be coded as 1110 (0 is a separate bit). The separation bit positions ($t_i$) can be computed at the same time in bit-parallel in [Eq.1].

$$t_i = \sum_{j=1}^{i} q_j + i - 1, (1 \le i \le 15)$$

[Eq. 1]

For example, when $q_1$=1, $q_2$=3, and $q_3$=5, the unary code is 101110111110 and separation bits are $t_1=q_1$=1, $t_2=q_1+q_2+1$=5, and $t_3=q_1+q_2+q_3+2$=11. If the length of compressed bits is longer than that of the original bits, GR packing chooses the original one. Otherwise, in Figure 1, each bit fields for the seed data, the length, the remainder, and the unary code are 8 bit, 7 bit, 30 bit, and 15~68 bit wide, respectively.

## 4. Asynchronous Compression Using GPGPU CUDA

This section optimally implements the lossy DPCM-GR compression algorithm [12] using GPGPU CUDA computing. First, the original game image of UHD(4096x2160) should be transferred to the CUDA device memory from the Host memory.

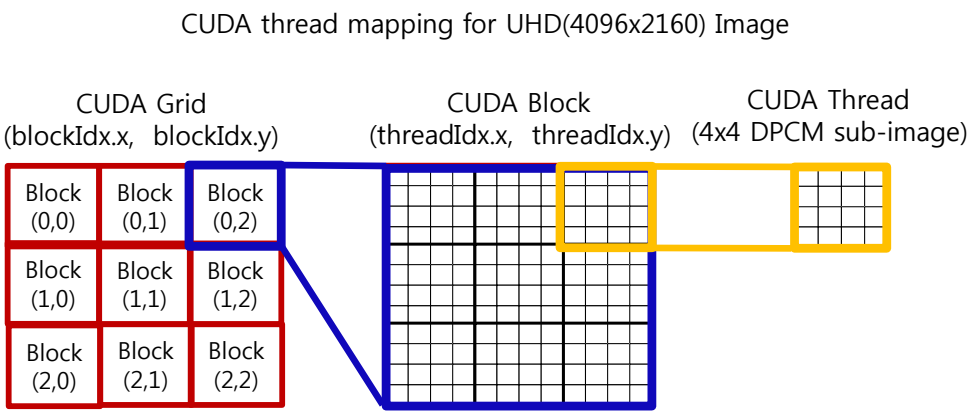


**Figure 2. CUDA Grid, Block, and Thread for DPCM-GR Compression [12]**

In Figure 2, a single UHD game image is divided into 4x4 sub-images which are handled by each CUDA thread for DPCM computation. The grid consists of CUDA blocks which have their own block identifier (blockIdx.x, blockIdx.y). Each block also consists of CUDA threads which have their own thread identifier (threadIdx.x, threadIdx.y). The number of available CUDA threads varies as the CUDA environment provided by the graphics hardware. After all CUDA threads simultaneously compute DPCM for 4x4 sub-images, they perform GR encoding, as shown in (2) DPCM and (3) parallel GR encoding of Figure 1. Finally, each CUDA thread performs (4) GR packing of Figure 1. All GR packed data of its 4x4 sub-image for whole UHD image is going to be transferred to Host system.

There are various types of memory inside CUDA system. The largest but slow memory is the Global device memory. The data can be transmitted and received between Host and CUDA through it. The Local memory is directly handled by GPU, which is a small portion allocated in the Global memory. There are also areas for constants and texture data in the Global memory. The fastest memories are the shared memory and register. The access speed of the shared memory is about 100 times faster than that of the global memory which is made of DRAM.

In this paper, in order to utilize the fast access speed of the shared memory, once each CUDA thread loads 4x4 sub-image on the shared memory, and then it performs its DPCM computation and GR encoding. If CUDA Compute Capability compiles the DPCM-GR kernel of this paper with its compile option of "-ptxas-options-v", we can obtain the shared memory and register usage in Table 1. It is reasonable that the number of register per thread is 14 and the usage of shared memory per block is 124bytes.

**Table 1. Shared Memory and Register Usage**

| Memory Type | Usage |
|---|---|
| Shared Memory per Block | 124 bytes |
| Register per Thread | 14 registers |

In order to optimize the performance, the CUDA compression kernel can be overlapped by the data transmission between Host and CUDA using asynchronous function processing. The page-locked (pinned) hosts memory should be used for the asynchronous data transfer. The page-locked memory is faster than the general Host memory. Also the CUDA stream can make both the kernel processing and the asynchronous transfer overlapped in pipelined fashion.

In this paper, the asynchronous function overlaps the CUDA DPCM-GR kernel execution with the transfers of both the original image from Host to CUDA and the compressed data from CUDA to Host. In Figure 3, four CUDA configurations for the asynchronous transfer are shown. The configuration A is for the serial which serially performs all data transfer and kernel executions. The configuration B (two-way concurrency 1) is that the original image data transfer from Host to CUDA is serial and the kernel execution and the compressed data transfer from CUDA to Host is overlapped. In the configuration C (two-way concurrency 2), that the original image data transfer from Host to CUDA and the kernel execution is overlapped and the compressed data transfer from CUDA to Host is serial. Finally in the configuration D (three-way concurrency), the original image data transfer from Host to CUDA, the compression kernel execution, and the compressed data transfer from CUDA to Host are asynchronously overlapped in pipelined fashion.

**This paper evaluates the performance of four concurrency configurations through experiments.**

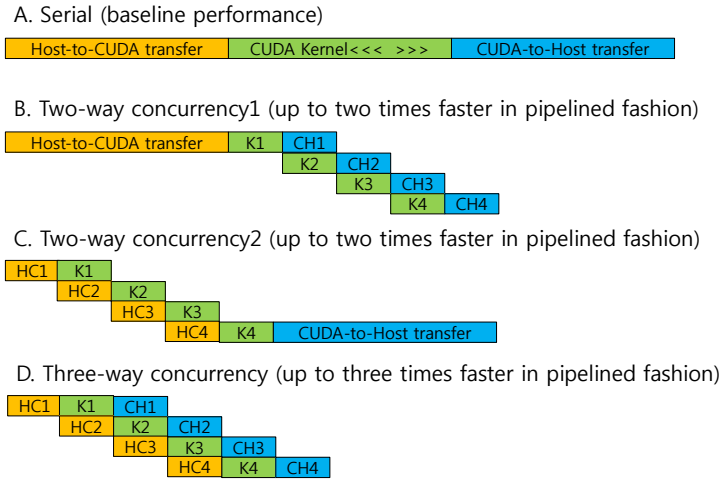


**Figure 3. Four Concurrency Configurations According to Asynchronous Transfers [12]**

## 5. Performance Evaluation

In this Section, to evaluate the performance of four concurrency configurations about the CUDA kernel execution and the asynchronous data transfer, two systems including

GPGPU CUDA systems such as NIVIA GeForce GTX G650Ti and GT330M are prepared as shown in Table 2.

For the experiments, the system I of Table 2 is about Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz 3.40GHz with 8GB RAM and NVIDIA GeForce GTX650Ti (Compute Capability 3.0, 768 Cores, 1.03GHz, 2048MB Global Memory). Also, the system II of Table 2 is about Intel(R) Core(TM) i5 CPU M520 @ 2.40GHz 2.40GHz with 8GB RAM and NVIDIA GeForce GT330M (Compute Capability 1.2, 48 Cores, 1.1GHz, 256MB Global Memory).

Under the asynchronous transfer environment using the page-locked Host memory, the memory bandwidths of the Host-to-CUDA transfer and the CUDA-to-Host transfer are good up to 11847.8MB/s and 12019.2MB/s, respectively.

This paper uses six benchmark UHD (4096x2160) game images for experiments in Figure 4. To compute the computation time reduction, the Host CPU execution times for DPCM-GR compression for six benchmark images of Figure 4 are measured according to System I and System II as in Table 3. The average execution CPU times of System I and System II are 1019.45ms and 1686.76ms, respectively. The average compression ratio is 0.68 for six benchmark images.

**Table 2. GPGPU CUDA Computing Environment**

| | System I | System II |
|---|---|---|
| Host CPU | Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz 3.40GHz | Intel(R) Core(TM) i5 CPU M520 @ 2.40GHz 2.40GHz |
| CPU RAM | 8GB RAM | 8GM |
| GPGPU CUDA System | NVIDIA GeForce GTX 650 Ti | NVIDIA GeForce GT330M |
| Driver version | 7.5 | 6.5 |
| CUDA Capability | 3.0 | 1.2 |
| Total Global Memory | 2048MB | 256MB |
| Cores | 768 Cores =<br>4 Multiprocessors x<br>192 CUDA cores | 48 Cores =<br>6 Multiprocessors x<br>8 CUDA cores |
| GPU Clock rate | 1.03 GHz | 1.1 GHz |
| Shared Memory/block | 48KB | 16KB |
| Warp size | 32 | 32 |
| Max threads per block | 512 | 512 |
| H2C Pinned memory bandwidth | 11847.8 MB/s | 12170.3 MB/s |
| C2H Pinned memory bandwidth | 12019.2 MB/s | 12228.3 MB/s |

 

Image 1 [13] Image 2 [14] Image 3 [15]

Image 4 [16] Image 5 [17] Image 6 [18]

**Figure 4. Benchmark UHD (4096x2160 Resolution) Game Images**

**Table 3. Host CPU Times and Compression Ratio for Benchmark Images**

| | Host CPU Time (ms) | | Compression Ratio |
|---|---|---|---|
| | System I | System II | |
| Image 1 | 1031.56 | 1699.39 | 0.621 |
| Image 2 | 1039.35 | 1702.93 | 0.743 |
| Image 3 | 1023.01 | 1704.84 | 0.652 |
| Image 4 | 1038.21 | 1697.37 | 0.847 |
| Image 5 | 1002.79 | 1666.89 | 0.615 |
| Image 6 | 981.77 | 1649.12 | 0.601 |
| average | 1019.45 | 1686.76 | 0.680 |

Figure 5 presents the compression times according to four concurrency configurations (A, B, C, and D) of Figure 3 for two Systems I and II in Table 2 under the GPGPU CUDA computing environment. In addition, four types of the original image slices like 2, 4, 8, and 16 are applied for the asynchronous pipelined processing. The performance differences among the execution times for six benchmark images are negligible. It is also same to those of four image slices. The reason is that the preparation overhead time between the kernel execution and the data transfer does much more important role.

However, Figure 5 shows the clear performance differences among four concurrency configurations in two Systems I and II. The configuration D (three-way concurrency) has the best performance which reduce the compression time up to 54.1% and 30.3%

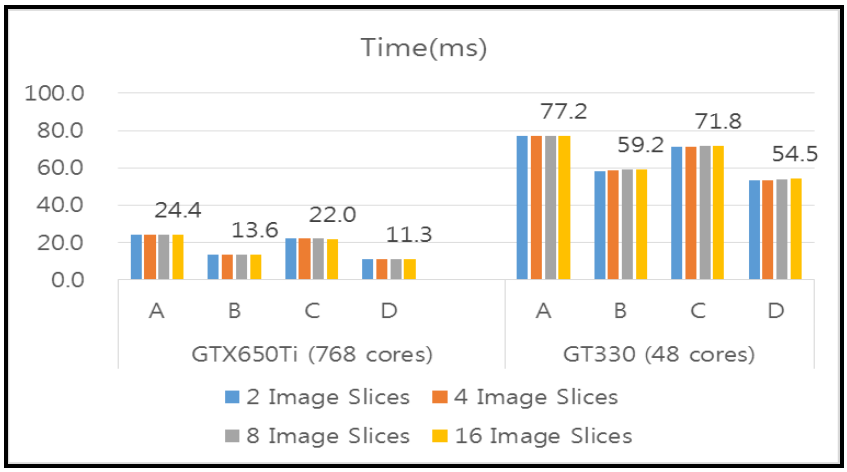


**Figure 5. GPGPU CUDA Time According to Graphics System, the Number of Image Slices, and Concurrency Configurations**

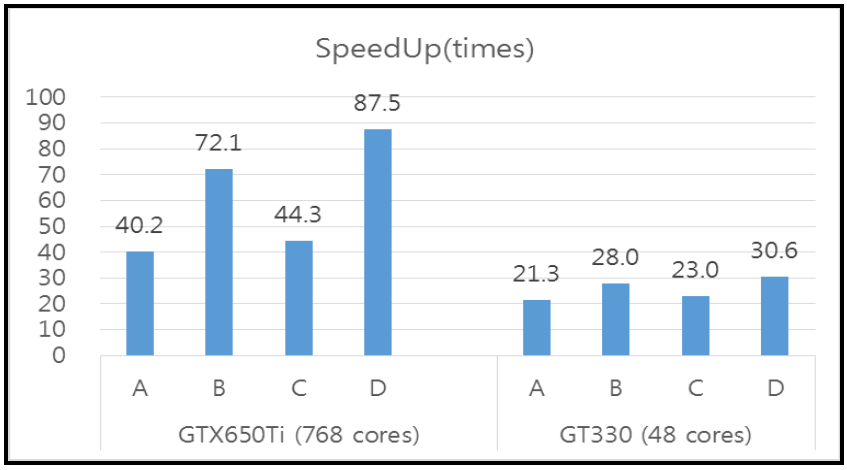


**Figure 6. SpeedUp (CPU Time / GPGPU CUDA Time) According to Graphics System, the Number of Image Slices, and Concurrency Configurations Holliday**

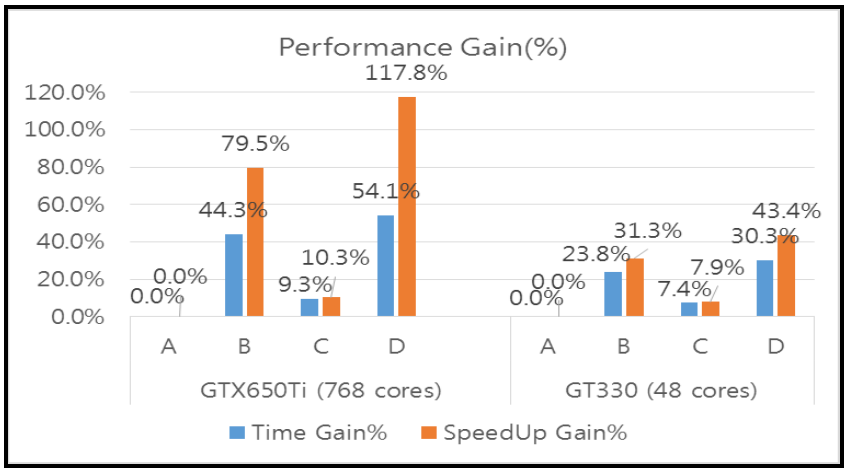


**Figure 7. Performance Gain (Time Gain% and SpeedUp Gain%) According to Graphics System and Concurrency Configurations**

Than the concurrency configuration A (Serial) for System I and II with GTX650Ti and GT330, respectively. The configuration B (two-way concurrency 1) has the performance gain up to 44.3% and 9.3% about the compression time than the concurrency configuration A (Serial) for System I and II with GTX650Ti and GT330, respectively. Also, the configuration C (two-way concurrency 2) reduce the compression time up to 23.8% and 7.4% than the concurrency configuration A (Serial) for System I and II with GTX650Ti and GT330, respectively. The reason where the performance of the configuration C is not better than that of the configuration B is that the compressed data is smaller and more irregular than the original image.

Figure 6 shows the speed ups of the GPCPU CUDA computation time than the Host computation time. For two Systems I and II, four concurrency configurations like A, B, C, and D have the speed ups of 40.2, 72.1, 44.3, and 87.5 times in the GTX 650 Ti than the Host computation time. For GT330M, four concurrency configurations like A, B, C, and D have the speed ups of 21.3, 28.0, 23.0, and 30.6 times than the Host computation time. Overall the configuration D is best.

Figure 7 explains the time gain and the speed up gain for four concurrency configurations according to two Systems I and II. The speed ups of A than B, C, and D in GTX 650Ti are 79.5%, 10.3%, and 117.8%, respectively. Also, the speed ups of A than others in GT330M are 31.3%, 7.9%, and 43.4%, respectively. Overall the performance of the concurrency confirmation D is the best.

Through Figure 5, 6 and 7, the computation time of the configuration D in GTX 650Ti has better performance of 79.2% than that of the GT330M.

## 5. Conclusion

To reduce the memory bandwidth for the UHD game display, the study of the frame image compression is required. The lossy compression algorithm is suitable in order to prevent the degradation of the image quality.

In this paper, in order to prevent the degradation of the image quality, this paper increases the memory bandwidth efficiency using the shared memory of CUDA. Also, various asynchronous transfer configurations which can overlap the kernel execution and data transfer between host and CUDA are implemented with the page-locked host memory based on the compression algorithm [12].

Experimental results show that the configuration D obtains the maximum 87.5 and 30.6 speedups for GTX650Ti and GT330, respectively, comparing to Host CPU. Also, the maximum reductions of the compression time for GTX650Ti and GT330 are 54.1% and 30.3%, respectively, among various configurations.

## References

[1] S. Lee, M. Chung, S. Park, and C. Kyung, “Lossless frame memory recompression for video codec preserving random accessibility of coding unit”, IEEE Trans. Consumer Electro., vol. 55, no. 4, **(2009)**, pp. 2105-2113.
[2] S. Lee, N. Eum, M. Chung and C. Kyung, “Low Latency Variable Length Coding Scheme For Frame Memory Recompression”, 2010 IEEE International Conference on Multimedia and Expo (ICME), IEEE, **(2010)**, pp.232-237.
[3] J. Kim and C. M. Kyung, “A lossless embedded compression using significant bit truncation for HD video coding”, IEEE Transaction on Circuit and System for Video Technology, vol. 20, no. 7, **(2010)**, pp. 848-860.
[4] H. S. Kim, J. Lee, H. Kim, S. Kang and W. C. Park, “A Lossless Color Image Compression Architecture Using a Parallel Golomb-Rice Hardware Codec”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 21, no. 11, **(2011)**, pp. 1581-1587.
[5] S. Morein, “ATI Radeon HyperZ Technology”, ACM/Eurographics Symposium on Graphics Hardware, **(2000)**.
[6] J. Deroo, S. Morein, B. Favera and M. Wright, “Method and Apparatus for Compressing Parameter Values for Pixels in a Display Frame”, In US Patent 6,476,811, **(2002)**.
[7] “NVIDIA CUDA”, http://developer.nvidia.com/object/cuda.html.

 

[8] S. W. Ha, M. H. Choi, T. J. Jun, J. W. Kim, H. R. Byun and T. D. Han, "Bandwidth Efficient Summed Area Table Generation for CUDA", Journal of Korea Game Society, vol. 12, no. 5, **(2012)**, pp. 67-78.
[9] J. W. Kim, Y. Jung, J. Park, Y. J. Park and T. D. Han, "Optimization of Color Format Conversion of WebCam Images Using the CUDA", Journal of Korea Game Society, vol. 11, no. 1, **(2011)**, pp. 147-157.
[10] J. H. Lim and B. S. Shin, "Min-Max Octree Generation Using CUDA", Journal of Korea Game Society, vol. 9, no. 6, **(2009)**, pp. 191-196.
[11] N. L. Kim and J. W. Kim, "GPU-based Low-latency DXT Compression and Transport for 4K Ultra-high-definition Media Sharing", KIISE Transactions on Computing Practices, vol. 18, no. 8, **(2012)**, pp. 573-581.
[12] Y. Kim, "CUDA based Lossless Asynchronous Compression of Ultra High Definition Game Scenes using DPCM-GR", Journal of Korea Game Society, vol. 14, no. 6, **(2014)**, pp. 59-68.
[13] Image 1, Asura's Wrath from http://www.giantbomb.com/
[14] Image 2, Tom Clancy's the Division from http://www.dlh.net/
[15] Image 3, Tom Clancy's the Division from http://www.dlh.net/
[16] Image 4, Devil May Cry 5 from http://-www.gamehdwall.com/
[17] Image 5, ForgeMaster from http://www.over3000.net/
[18] Image 6, China Lake Grenade Launcher Pop Gun from http://www.polycount.com/

## Authors

**Youngsik Kim**, received the B.S., M.S., and Ph.D degree in Dept. Computer Science from the Yonsei University, Korea, in 1993, 1995, and 1999 respectively. He had worked for System LSI, Samsung Electronics Co. Ltd from Aug. 1999 to Feb. 2005 as a senior engineer. Since March 2005 he has been working for Dept. of Game & Multimedia Engineering in Korea Polytechnic University. His research interests are in 3D Graphics and Multimedia Architectures, Game Programming, and SOC designs.

 