Fully Parallelized Lossless LZW decompression for CUDA-enabled GPUs

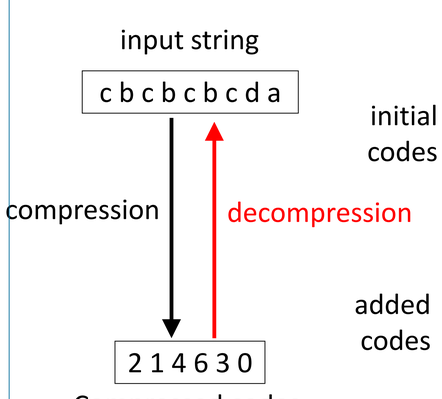
Koji Nakano, Shunji Funasaka, Yasuaki Ito
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527, JAPAN
HIROSHIMA UNIVERSITY

Abstract

We propose a parallel algorithm for LZW decompression and implement it in GeForce GTX 980. It achieves up to 40 times speedup over the sequential implementation using Intel Corei7-4790 (3.66GHz) processor.

LZW data compression

- Developed by Lempel, Ziv, and Welch
- Lossless : original data can be perfectly recovered
- Dictionary-base: dictionary is created sequentially
- Widely used: used in Unix "compress" command, and GIF/TIFF image formats.
- Original input: a string of 8-bit integers such as pixel values/characters
- Compressed output: a sequence of 9 to 12-bit codes



dictionary

	code	string
initial codes	0	a
	1	b
	2	c
	3	d
added codes	4	cb
	5	bc
	6	cbc
	7	cbcd
	8	da

LZW compression algorithm

Initialization: Create dictionary with initial codes

Repeat the following steps until all input are read

1. Find the longest substring W from the beginning of the input that is in the dictionary
2. Output the code corresponding to W and remove W from the input.
3. Read the next character x and add substring Wx to the dictionary

LZW decompression algorithm

Similar to LZW compression

The dictionary is created from compressed codes

Our parallel LZW decompression

Pointer-character table is created with each thread being assigned to each code, that is, fully parallelized

code	0	1	2	3	4	5	6	7	8
pointer					2	1	4	6	3
character	a	b	c	d	b	c	c	d	a

The character of each code in the pointer-character table is the last character of the string in the dictionary

Step 1: Create pointer-character table from the compressed codes

Compressed codes

2 1 4 6 3 0

code	0	1	2	3	4	5	6	7	8
pointer					2	1	4	6	3
character	a	b	c	d					

Step 2: Compute the character of each code by traversing pointer

code	0	1	2	3	4	5	6	7	8
pointer					2	1	4	6	3
character	a	b	c	d	c				

Step 3: Compute the string of each code by traversing pointer

code	0	1	2	3	4	5	6	7	8
pointer					2	1	4	6	3
character	a	b	c	d	b	c	c	d	a

code	2	1	4	6	3	0
string	c	b	c b	c b c	d	a

Step 4: Concatenate strings of all codes

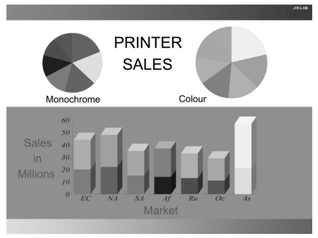
c b c b c b c d a

Experimental Results

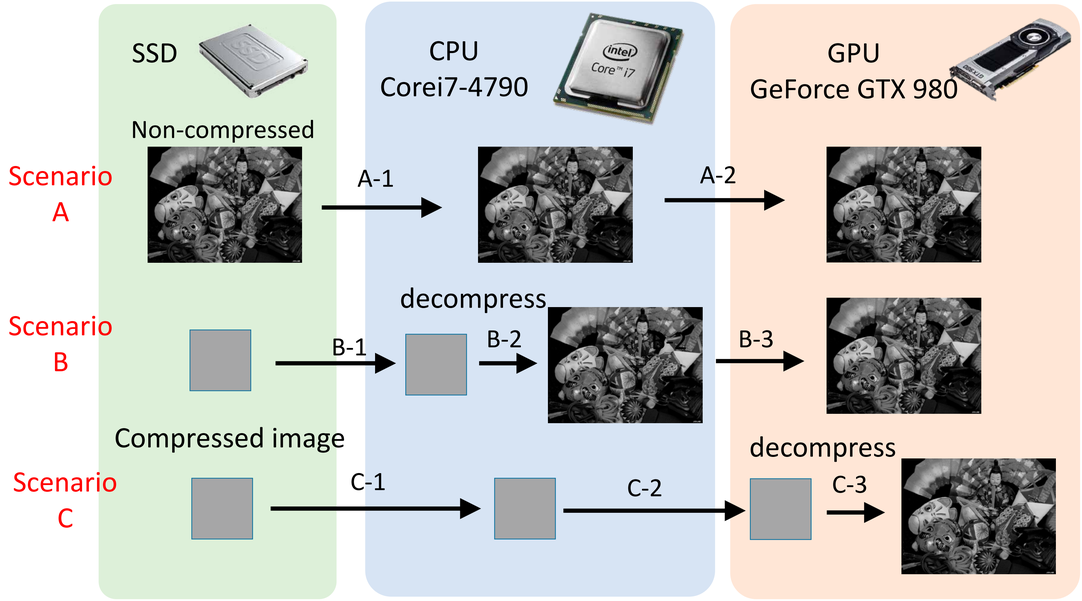
Evaluated the time for LZW-compressed TIFF images of 4096x3072 pixels (%:compression ratio)

Crafts: 59.1%

Flowers: 42.7%



Graph: 2.6%



Time in milliseconds

		SSD-CPU (A-1)	CPU-GPU(A-2)	Total
Scenario A	Crafts	6.38	3.53	9.91
	Flowers	6.43	3.52	9.95
	Graph	6.55	3.50	10.05

		SSD-CPU(B-1)	CPU Decompress(B-2)	CPU-GPU(B-3)	Total
Scenario B	Crafts	3.90	62.04	3.56	69.50
	Flowers	2.57	50.00	3.55	56.12
	Graph	0.18	25.00	3.59	28.77

		SSD-CPU(C-1)	CPU-GPU(C-2)	GPU Decompress(C-3)	Total
Scenario C	Crafts	3.89	2.43	2.12	8.44
	Flowers	2.58	1.73	1.25	5.56
	Graph	0.16	0.15	1.77	2.08

Conclusion

Our GPU implementation for the LZW decompression is up to 50.00/1.25=40 times faster than the CPU implementation. Also Scenario C is up to 10.05/2.08=4.8 times faster than Scenario A. These scenarios can be applied to big data applications such as training phase of deep learning using the GPU, because the GPU stores the original uncompressed image at the end of scenarios.