Analysis of Image Compression Algorithm：GUETZLI

Lingyi Li
August 18, 2017

Abstract

How to balance picture size and quality is the core of image compression. This paper evaluates Google's jpeg image compression algorithm Guetzli by comparing with the traditional encoder Libjpeg-turbo in terms of compression rate, compression time, memory usage and other aspects of the VTune optimization software developed by Intel. Tests show that Guetzli can compress the jpeg image by 20% to 30% on the basis of the existing compression algorithm, and there is no change in the quality of the picture. But at the same time, the time to squeeze the picture greatly increased. If the compression time is shortened, Guetzli will provide new possibilities for image compression.

Key Word：Guetzli；Libjpeg-turbo；Image compression；VTune；

1. Guetzli Introduction

Guetzli is a Google JPEG encoder released in 2017, designed to achieve high visual quality in the excellent compression density. Guetzli produces images that are typically 20-30% smaller than the equivalent quality images generated by other compression algorithms. The current calculation of Guetzli is very slow.

1.1 Guetzli Installation

Guetzli is open source. Google published all code on GitHub.

https://github.com/google/guetzli
See Appendix A.

1.2 Guetzli Features

Guetzli uses an iterative optimization process. In order to make the problem simpler, the optimizer is not guided by the file size. Instead, it is driven only by perceived quality. The aim is to create a JPEG encoding with a perceived distance that is below and as close as possible to a given threshold. Each iteration produces a candidate output JPEG, and finally selects the best one.

Guetzli uses the closed-loop optimizer to adjust the image in two ways: optimizing the JPEG global quantization table and the DCT coefficients in each JPEG block. Specific optimization process see below:

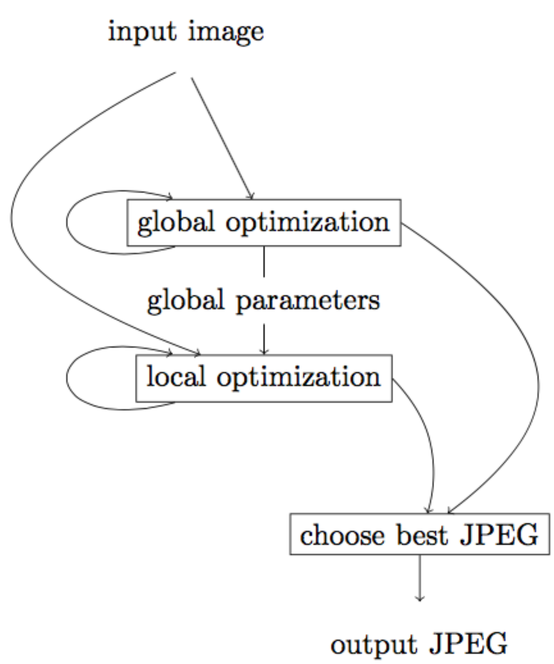


Figure 1 Guetzli optimization process (https://arxiv.org/pdf/1703.04421.pdf)

1.3 Butteraugli Metrics

Guetzli uses Google' s perceived distance metric Butteraugli as a source of feedback in its optimization process. Butteraugli is a model that "evaluates color perception and visual masking more thoroughly and in more detail than other encoders." The goal is to find the smallest JPEG that the human eye cannot distinguish from the original image.

Butteraugli takes into account three visual features that most JPEG encoders do not use. First, gamma correction should not be applied to each RGB channel, respectively, due to the overlap of the sensitivity spectra of the cone. For example, the amount of yellow light seen by the human eye is related to the sensitivity of the blue light, so the blue change near the yellow can be less accurate. The YUV color space is defined as a linear transformation of the gamma-compressed RGB, and is therefore not sufficient to model this phenomenon. Second, the resolution of the human eye in the blue is lower than that of the red and green, and there is almost no blue receptor in the high-resolution region of the retina, so that the high frequency variation of the blue can be less accurately encoded. Finally, the visibility of the fine structure in the image depends on the amount of visual activity nearby, that is, we can less precisely encode areas with large amounts of visual noise. The above considerations make Guetzli to ensure uniform loss of image.

2.Libjpeg-turbo Introduction

To test Guetzli performance, this article compares Guetzli with another commonly used jpeg encoder, Libjpeg-turbo.

2.1 Libjpeg-turbo Installation

https://github.com/libjpeg-turbo/libjpeg-turbo
See Appendix B.

2.2 Libjpeg-turbo Features

Libjpeg-turbo is a branch of libjpeg that uses the SIMD instruction to speed up baseline JPEG encoding and decoding, compressing bmp or ppm images into jpg format.

3. Performance Testing

3.1 Testing Purposes

Compare Guetzli compressed images and Libjpeg-turbo compressed images with the compression time and compression rate under different CPU and different quality parameters.

3.2 Test Environment

System hardware environment

Platform	**Broadwell**
Processor	E5-2699 v4
Frequency	2.20 GHz
Max Turbo Frequency	3.50 GHz
Memory	8 * 32GB 2133 MHz
FSB/QPI Frequency	9.6 GT/s
Thread(s) per Core	2
Sockets	2
Number of Core per	22
L1d Cache	32KB
L1i Cache	32KB
L2 Cache	256KB
L3 Cache (Total)	56320KB
SMT/MUNA/TURBO	ON

Table 1 CPU1 information

Platform	**Skylake**
Processor	8180
Frequency	2.5 GHz
Max Turbo Frequency	3.5 GHz
Memory	12 * 16GB 2666 MHz
FSB/QPI Frequency	10.4 GT/s
Thread(s) per Core	2
Sockets	2
Number of Core per	28
L1d Cache	32KB
L1i Cache	32KB
L2 Cache	1024KB
L3 Cache (Total)	39424KB
SMT/MUNA/TURBO	ON

Table 2 CPU2 information

System Software Environment

OS	CentOS 7.3.1611
kernel	3.10.0
Compiler	gcc:4.8.5 20150623

Table 3 System software environment

3.3 Test Implementation

Image	Pixel	Size (byte)
nightshot_iso_100.bmp	192*144	82998
head.bmp	444*600	799254
lagochungara.bmp	871*573	1499022
ahom3.bmp	1024*768	2359350
earth.bmp	2048*1024	6291510

Table 4 Image information

Test five different sized bmp photos in order. In same CPU, first use Libjpeg-turbo to compress bmp image into jpg format, then use Guetzli to compress the second time under different quality coefficients. Change CPU for repeated operation.

CPU selected are Intel E5-2688 V4 and Skylake 8180. Skylake is a higher performance processor.

Due to the minimum of Guetzli quality parameters is 84, select 84,90,95 three parameters for comparison.

Command line：

```
time -p ./cjpeg -outfile test.jpg image.bmp
time -p ./bin/Release/guetzli --quality 84 test.jpg output.jpg
```

Finally, change single process to multi-process testing.

Multi-process code：

```
#include <stdlib.h>
#include <omp.h>
int main()
{
    #pragma omp parallel for
    for(int i=0; i<=1000; i++)
    {
        ./bin/Release/guetzli —quality 84 test.jpg output.jpg;
    }
}
```

Command line：

```
g++ omp.cc -fopenup
./a.out
```

3.4 Test Result

Single process

CPU	Image size(byte)	L memory consumption(byte)	L comp time(s)	L comp size(byte)	L comp rate	G quality	G memory consumption(byte)	G comp time(s	G comp size(byte)	G comp rate
E5-2699 V4	82998	25368	0.03	4357	94.75%	84	19196	0.89	3366	22.75%
						90	19196	0.83	3624	16.82%
						95	19196	0.83	3879	10.97%
	799254	25314	0.04	39063	95.11%	84	36800	12.85	33795	13.49%
						90	34848	11.28	35832	8.27%
						95	36800	9.04	36900	5.54%
	1499022	25330	0.04	116070	92.26%	84	56040	26.12	95496	17.73%
						90	55868	25.06	102766	11.46%
						95	55868	21.28	109555	5.61%
	2359350	25316	0.04	40584	98.28%	84	78748	37.26	31022	23.56%
						90	77084	27.26	32562	19.77%
						95	74344	23.87	33908	16.45%
	6291510	25314	0.05	237097	96.23%	84	182472	104.36	190895	19.49%
						90	182472	93.76	206747	12.80%
						95	174792	93.27	220427	7.03%

Table 5 single-process E5-2699 V4 test results

CPU	Image size(byte)	L comp time(s)	G comp time(s)
SKL8180	82998	0.01	0.74
			0.66
			0.66
	799254	0.01	10.43
			8.89
			7.13
	1499022	0.01	21.73
			20.19
			18.57
	2359350	0.02	30.68
			22.30
			19.63
	6291510	0.02	91.95
			81.50
			81.46

Table 6 single-process Skylake 8180 test results

Multi-process

CPU	Image size(byte)	G quality	G comp time(s)	Throughput(/s)
E5-2699 V4	82998	84	22.69	44.07
		90	20.55	48.66
		95	20.20	49.50
	799254	84	326.55	3.06
		90	280.34	3.57
		95	220.03	4.54
	1499022	84	658.49	1.52
		90	618.13	1.62
		95	561.95	1.78
	2359350	84	892.16	1.12
		90	653.18	1.53
		95	575.17	1.74
	6291510	84	2468.88	0.41
		90	2192.57	0.46
		95	2180.96	0.46

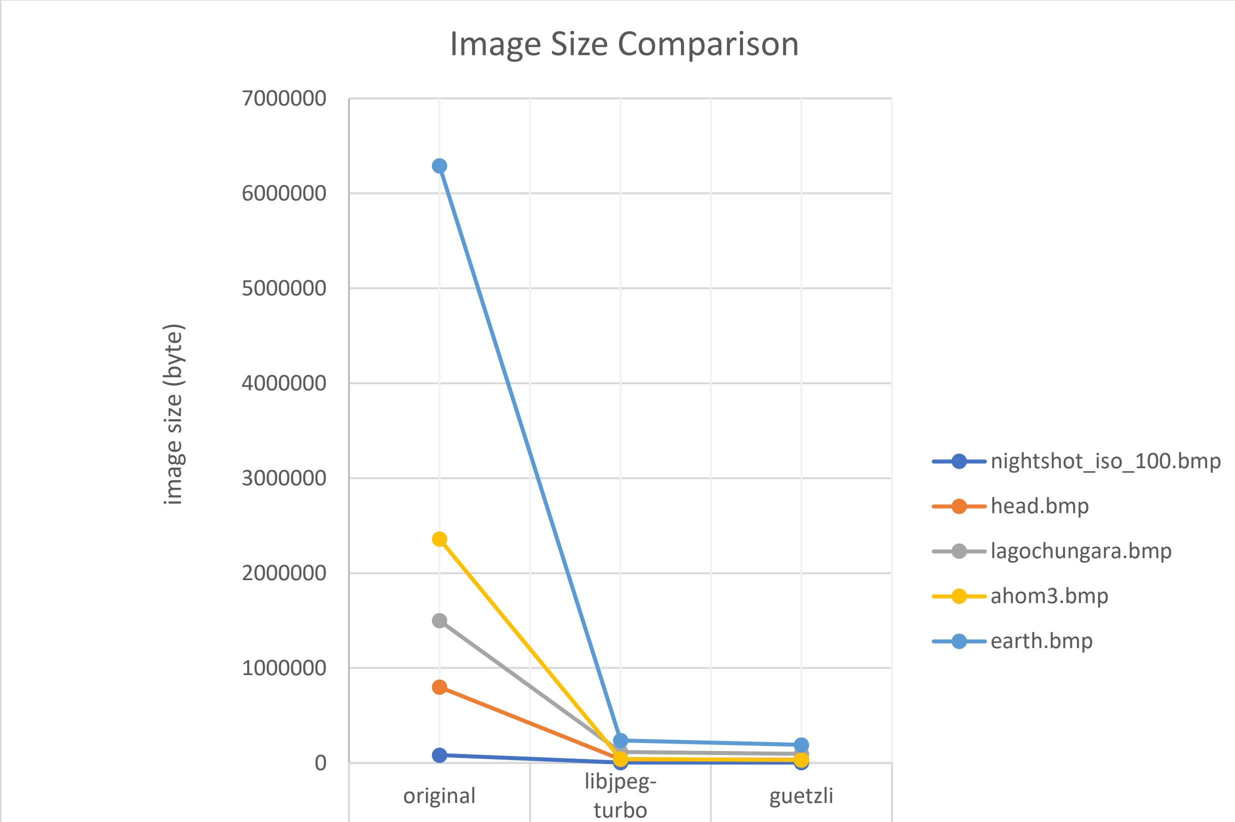
CPU	Image size(byte)	G quality	G comp time(s)	Throughput(/s)
SKL8180	82998	84	20.19	49.53
		90	17.96	55.68
		95	15.76	63.45
	799254	84	274.21	3.65
		90	222.96	4.49
		95	188.18	5.31
	1499022	84	539.56	1.85
		90	510.93	1.96
		95	468.20	2.14
	2359350	84	781.23	1.28
		90	569.02	1.76
		95	429.91	2.33
	6291510	84	2272.84	0.44
		90	1989.99	0.50
		95	2002.44	0.50

Table 7 multi-process test results

3.5 Result Analysis

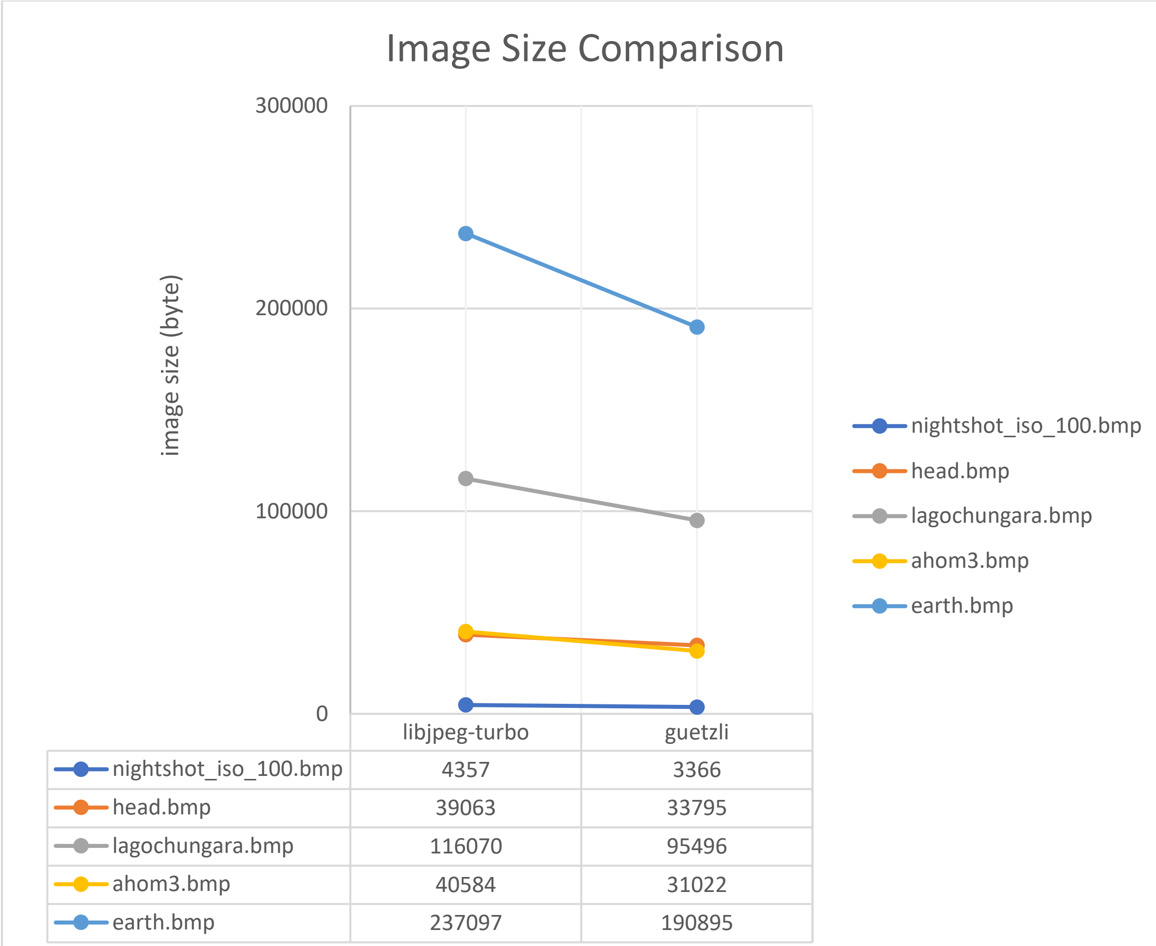
Analyzing data in section 3.4, the following graphs can be drawn:

- Image size comparison



	original	libjpeg-turbo	guetzli
nightshot_iso_100.bmp	82998	4357	3366
head.bmp	799254	39063	33795
lagochungara.bmp	1499022	116070	95496
ahom3.bmp	2359350	40584	31022
earth.bmp	6291510	237097	190895

Figure 2 Image size comparison



	libjpeg-turbo	guetzli
nightshot_iso_100.bmp	4357	3366
head.bmp	39063	33795
lagochungara.bmp	116070	95496
ahom3.bmp	40584	31022
earth.bmp	237097	190895

Figure 3 Image Size Comparison

According to Figure 2 and 3, Guetzli can compress additional 15% on the basis of Libjpeg-turbo.

- Single-Process Compression Time Comparison (Different CPU)

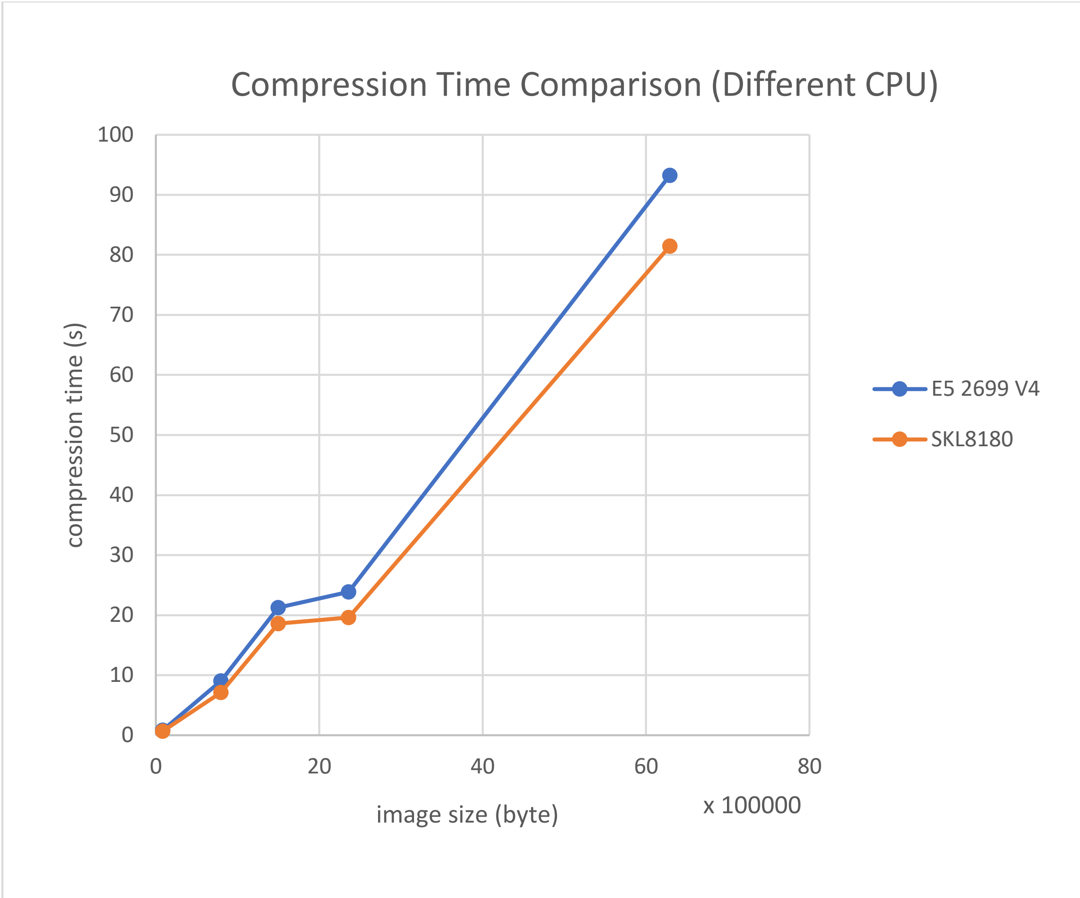


Figure 4 Single-Process Compression Time Comparison (Different CPU)

According to Figure 4, the larger the image size, the longer the Guetzli compression time. Skylake 8180 can reduce the compression time by about 20%.

- Compression Time Comparison (Different quality)

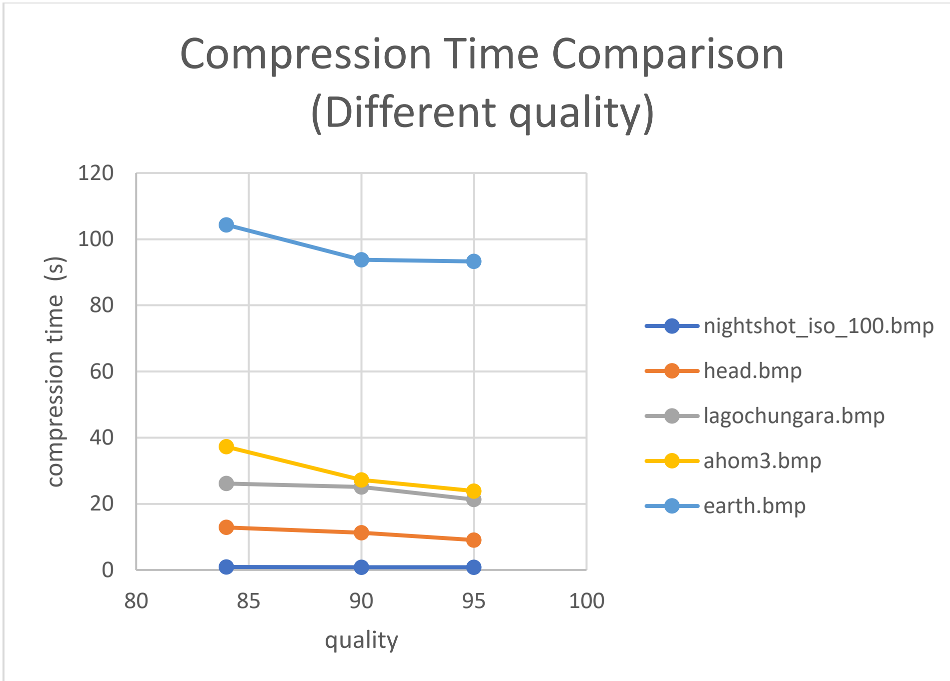


Figure 5 Compression Time Comparison (Different quality)

According to Figure 5, the greater the quality factor, the shorter the compression time of Guetzli.

- Compression Rate Comparison

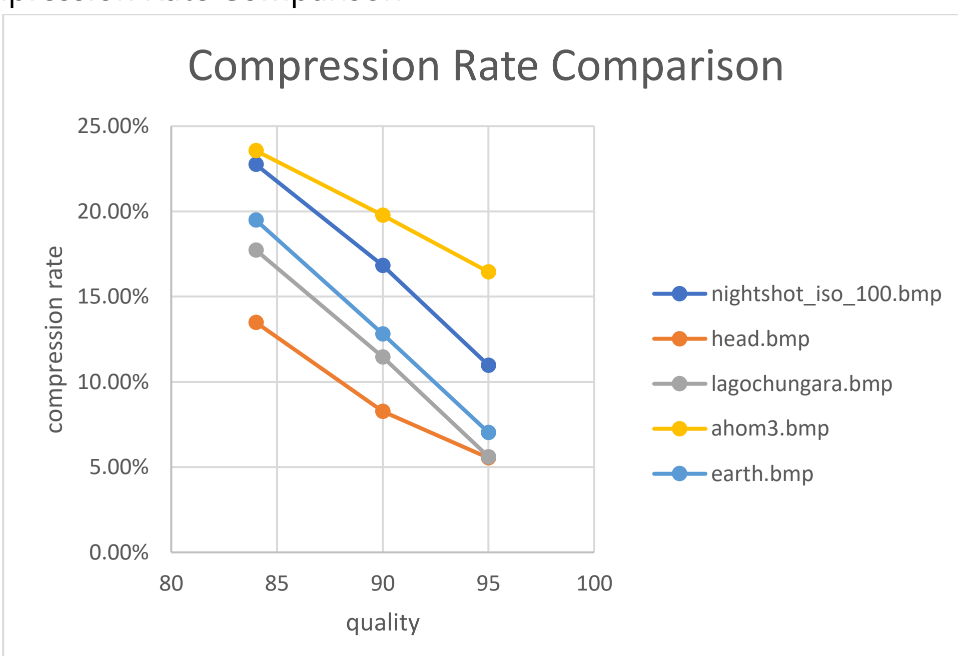


Figure 6 Compression Rate Comparison

According to Figure 6, the higher the mass coefficient, the lower the compression ratio.

- Memory Consumption Comparison

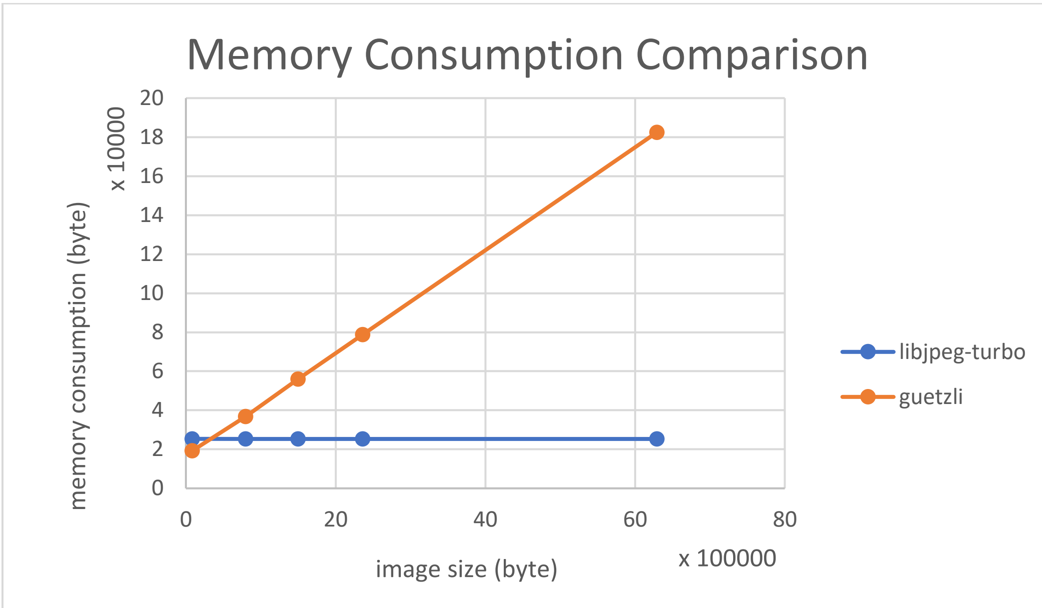


Figure 7 Memory Consumption Comparison

According to Figure 7, The larger the image size, the more memory the guetzli consumes. Libjpeg-turbo consumes less memory and is basically the same.

- Multi-Process Throughout Comparison

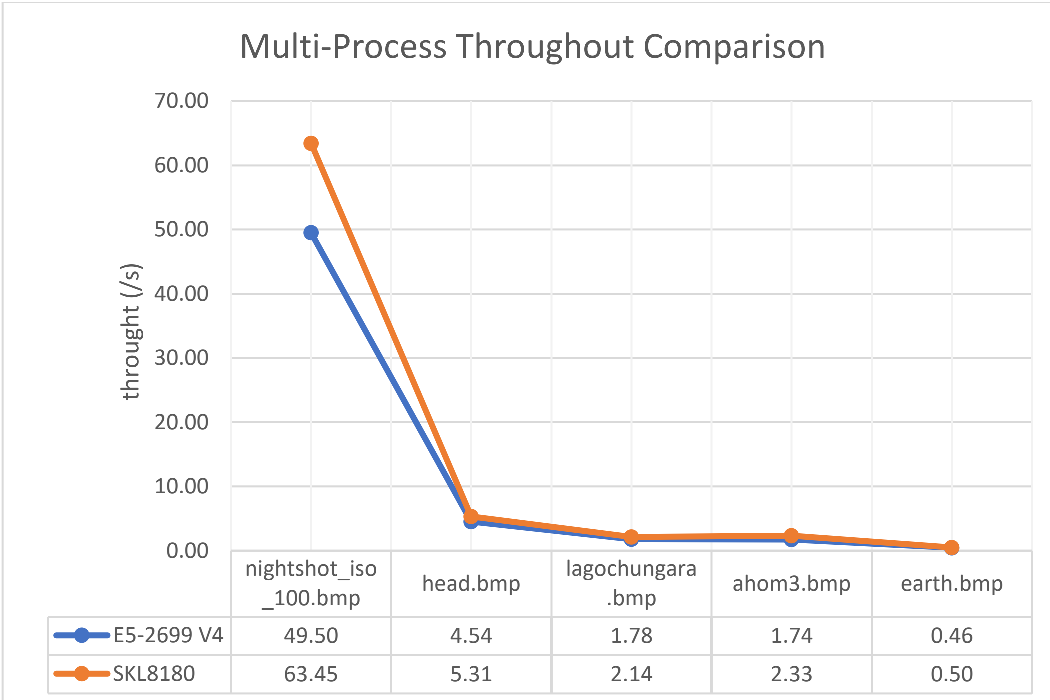


Figure 8 Multi-Process Throughout Comparison

According to Figure 8, Skylake 8180 can increase throughput by about 20%.

4. VTune Analysis

4.1 VTune Installation

https://software.intel.com/en-us/-getting-started-with-intel-vtune-amplifier-xe-2017

See Appendix C.

4.2 VTune Introduction

The VTune Amplifier Performance Analyzer is a product of Intel Parallel Studio and is a commercial application for software performance analysis based on 32-bit and 64-bit x86 machines. It has GUI (graphical user interface) and command line, and provides Linux or Microsoft Windows operating system version.

VTune Amplifier assists in various code analysis, including stack sampling, thread analysis and hardware event sampling. The analyzer results include details such as the time spent in each subroutine.

This paper focuses on analyzing the reason of longtime processing through VTune hotspot analysis.

4.3 VTune Implementation

Use the command line on the root to test, copy the results to spark on the GUI to display.

```
source amplxe-vars.sh
amplxe-cl -collect hotspots bin/Release/guetzli --quality 84 output1.jpg output2.jpg
amplxe-cl -report hotspots
```

4.4 VTune Result

- Summary

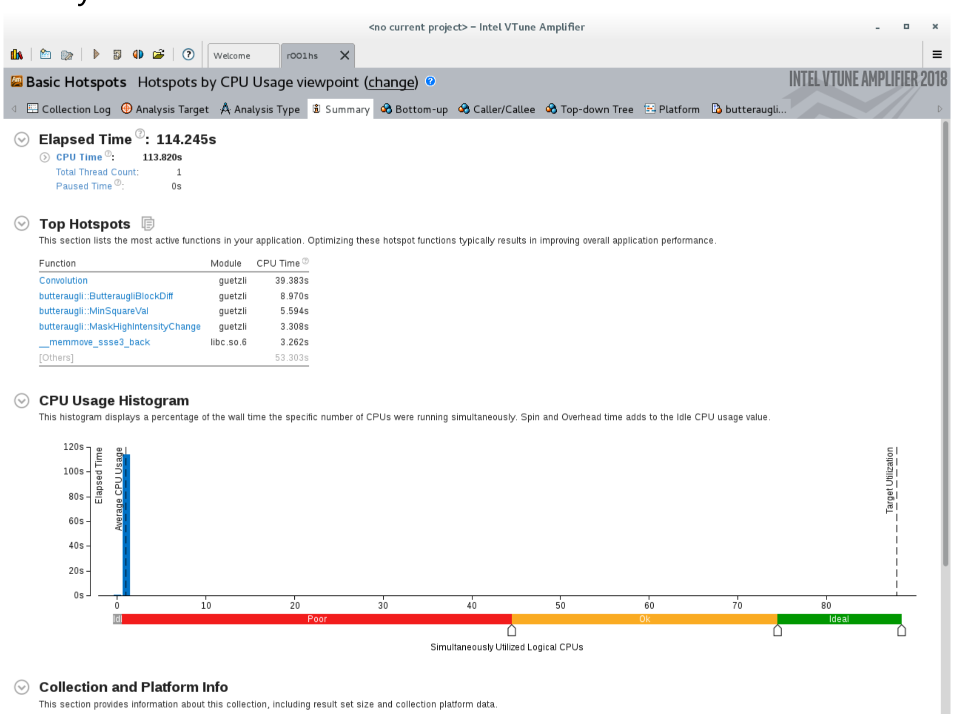


Figure 9 Summary: CPU Usage Histogram

- Bottom-up

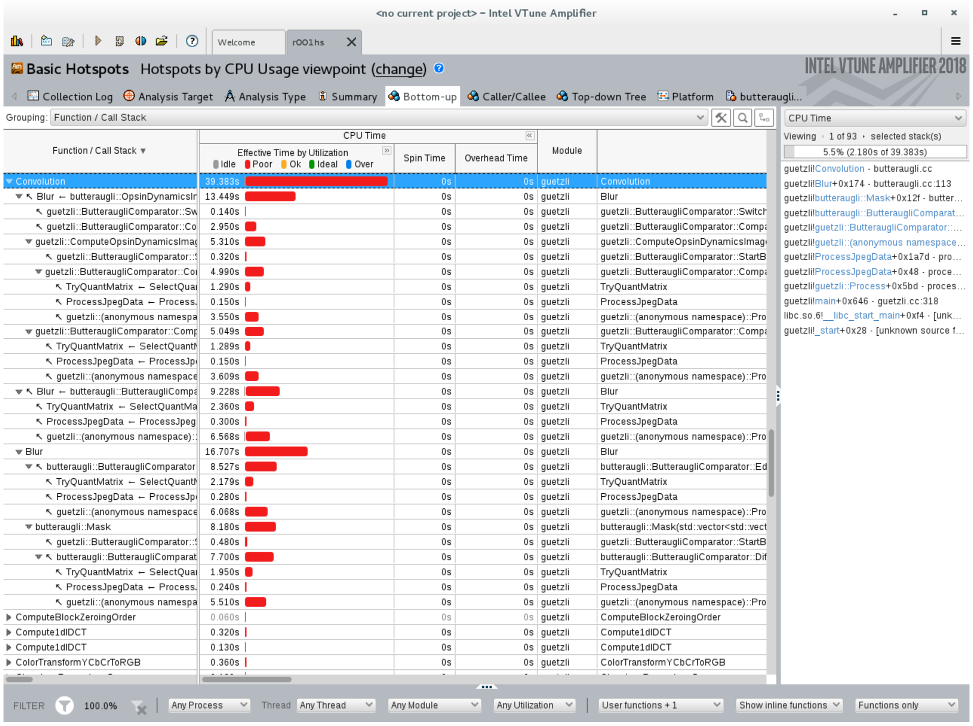


Figure 10 Bottom-up：convolution CPU consumption time

As can be seen from Figure 10, convolution is the function that takes the longest time.

- Caller/Callee

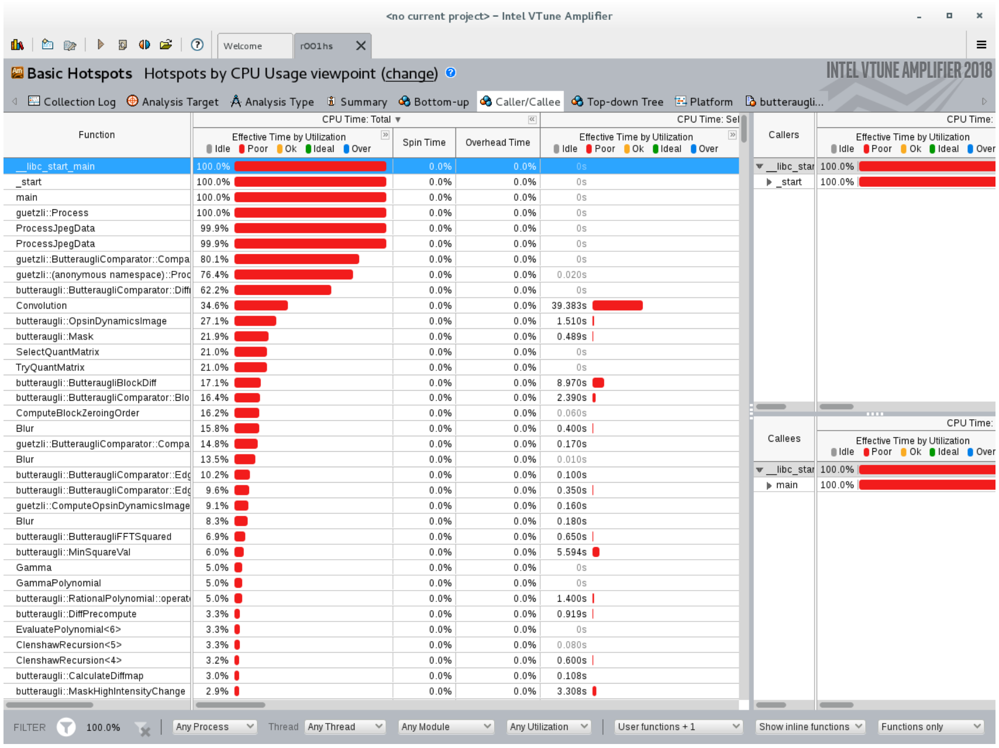


Figure 11 Caller/Callee：the caller of function convolution

- Specific code

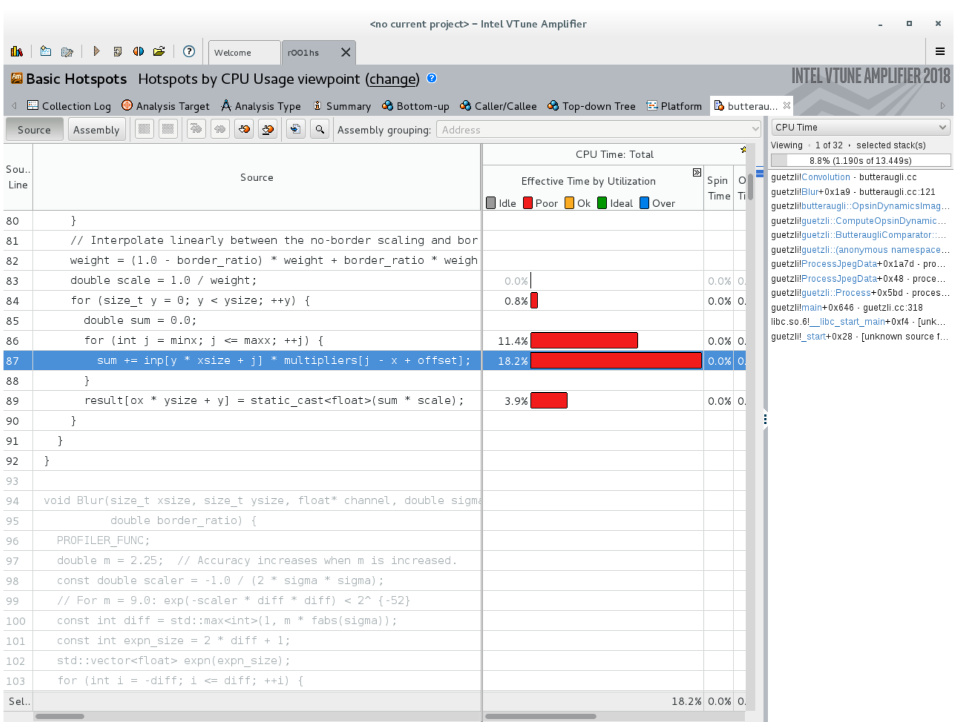


Figure 12 Specific code

5. Conclusion

Guetzli can compress the additional 20% to 30% on ordinary jpeg images, and the the picture quality observed by naked eyes has not changed. But because of its compression time is also lengthened, the performance is difficult to be commercially used. According to the VTune hotspot analysis, if the cost of convolution is decreased, the usability of Guetzli will be enhanced.

6. References

[1] https://arxiv.org/pdf/1703.04421.pdf

7. Appendix

7.1 Guetzli Installation

1. copy source code

git clone git@github.com:google/guetzli.git

2.install libpng

3.make

7.2 Libjpeg-turbo Installation

1.copy source code

git clone https://github.com/libjpeg-turbo/libjpeg-turbo.git

2.install nasm

yum install nasm

3.mkdir build

4.autoreconf –fiv

5.cd build

6.sh ../configure

7.make

7.3 VTune Installation

1. scp spark@dl-bj:/home/jimin/parallel_studio_xe_2018_beta_update1_cluster_edition.tgz .

7.4 Image example

earth.bmp	2048*1024	6291510

original

libjpeg-turbo (75)

guetzli quality 84