PC-2023/24 Alpha-Compositor

Christian Mancini
E-mail address
christian.mancini1@edu.unifi.it

Abstract

We implement a parallel version of the alpha compositor, to compose a foreground over multiples background in a fixed position, starting from a serial version. We also show the speed up we gained of just the compositing part and the overall program with 12 and 7 times of speed up respectively. The tests were made on an Arch Laptop with Linux Kernel 6.7.9, 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 8 cores 16 threads, 23 GB of dual Channel 3200MHz RAM (16 + 8) and max memory bandwidth 51.2 GB/s.

Future Distribution Permission

The author(s) of this report give permission for this document to be distributed to Unifi-affiliated students taking future courses.

1. Introduction

Alpha compositing or alpha blending is the process of combining one image with a background to create the appearance of partial or full transparency. In a 2D image a color combination is stored for each picture element (pixel), often a combination of red, green and blue (RGB). When alpha compositing is in use, each pixel has an additional numeric value stored in its alpha channel (RGBA), with a value ranging from 0 to 1. A value of 0 means that the pixel is fully transparent and the color in the pixel beneath will show through. A value of 1 means that the pixel is fully opaque. With this definition in mind we can produce the effect of drawing the source pixels on top of the destination pixels (foreground over a background) using the following formula:

$$[RGBA]_d = [RGBA]_s + [RGBA]_d(1 - A_s) \quad (1)$$

For our scope the foreground must be completely opaque. Changing the weights leads to different results.

1.1. Compositing Algorithm

The compositing algorithm is straightforward, we just need to apply the 1 on every channel of the background image.

```
for (int color = 0; color < 3; ++color) {
        background.rgb_image[backgroundIndex
        + color] =
        background.rgb_image[backgroundIndex
        + color] * beta
        +
        foreground.rgb_image[foregroundIndex
        + color] * alpha;
}
```

This is just a snippet of the compose function. The program has to performe 3 simple tasks:

1. Read Foreground and a vector of Backgrounds;
2. Compose Foreground on every Background;
3. Save all the Backgrounds on disk.

The most expensive operation is as expected the last one.

2. Parallelization Criteria

There are many ways to parallelize a program. The first step we take is to us Vallgrid [1] to check that we don't have leaks in the serial version of the program.

We can now decide how to parallelize the program using OpenMP [2]. Since we are dealing with small images, parallelize the compositing

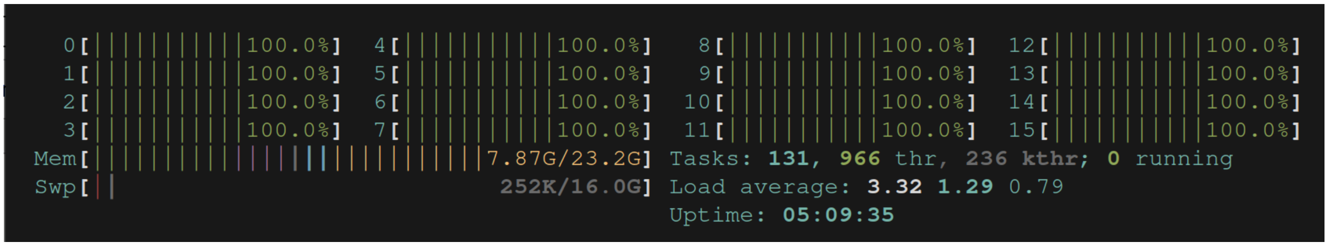

Figure 1. All cores are working

function will lead a slower computing time, instead we let each thread to call its own compositing function to a defined image. Loading and Saving follows the same idea. We got the following results: We can see from figure 1 that all the cores

	Load	Compose	Save	Total
Serial	0.57s	0.77s	40.78s	42,13s
Parallel	0.32s	0.058s	5,90s	6.29s

Table 1. 256x256 Foreground and 564 Backgrounds (roughly same dimention of foreground).

Fase	Speedup
Load	≈ 1.78
Compose	≈ 13.28
Save	≈ 6.91
Total	≈ 6.70

Table 2. Speedup.

are woking during the parallel execution.

References

[1] J. Nichols and N. Nethercote. Valgrind. 1

[2] OpenMP Architecture Review Board. *OpenMP Application Program Interface*. OpenMP Architecture Review Board, 2019. 1