Using Parallel Algorithm to Optimize Measuring Pixels and Lengths of Design Patterns

Yunke Zhou

Duong Ngoc Tran

Francis Ma

Mengqian Shen

Min Kyung Lee

Purdue University – Department of Statistics

Stat 598 Spring 2020 Final Project

**Part 1. Background**

“Check patterns” is a classic and widely used technique in fashion or cloth design such as scarves and shirts. Sales is very important in clothing design or garment industry, which benefits companies’ profits and cash flow, and fashion (such as colors, materials, and patterns) is strongly associated with sales.

In this project, our team is mainly focusing on the patterns. Patterns are always repeated, like characters or drawings in simple patterns, where complicated patterns may contain a combination of simple patterns.

Designers usually start with a simple pattern first, and to make adjustments to satisfy customers’ needs. This procedure is often completed by hand, which is time-consuming and may lead to human errors.

In reality, it turns to be a research question that whether the process can be automated with pattern recognition programming technique.

**Figure 1.** Sample patterns in real life

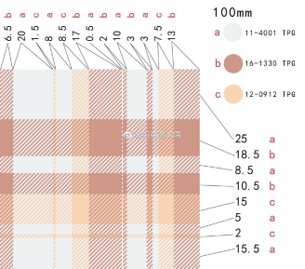
**Part 2. Research Questions and Methods**

**2.1 Research questions**

In the current project, there are four main research questions of interest:

First of all, we will use R programs to recognize the sample check images from PNG files, and auto input the data (such as pixels) from the sample check images.​

Second, each sample check image consists of several sub-patterns (see the pattern a, b, and c in Figure 2, and we will create R functions to detect these sub-patterns from the original check image and measure the side length of these sub-patterns (see the numbers of side lengths in Figure 2.



**Figure 2.** Examples of sample check patterns

Third, we will modify the R function to allow customers to input the overall dimension of the new check pattern, and the R function will output the expected side length of the sub-patterns accordingly.

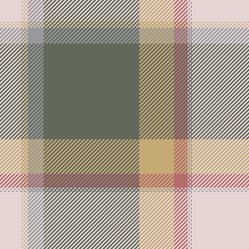
Finally, we will use parallel computing to optimize the multitasking, such as dealing with multiple check images at the same time. We will compare the parallel computing to the sequential tasking with the loops in R.

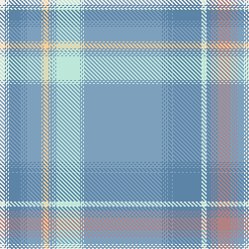
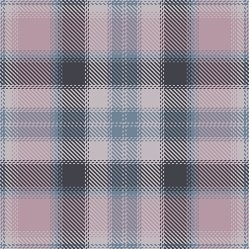
**2.2 Datasets**

In this project, we use a series of check images. These images were created by a designer, and she has authorized us to use these images for the purpose of this project. The sample images are shown in Figure 3.

All the check images are 252 pixels \* 252 pixels, and these images are all diagonally symmetric. Each check image consists of at least three colors.

In the sequential tasking, the four images will be analyzed one by one, and in the parallel computing, all four images will be analyzed in parallel. We will further compare the performance of the sequential tasking and the parallel computing with different numbers of images (ranged between 0 and 100).



**Figure 3**. Sample check images used in the project

**2.3 R packages**

The following R packages are used in this project:

1. Package **“png”** *(Urbanek, 2013)* is used to read the PNG files and measure the original dimensions (such as pixels) from the sample check images.

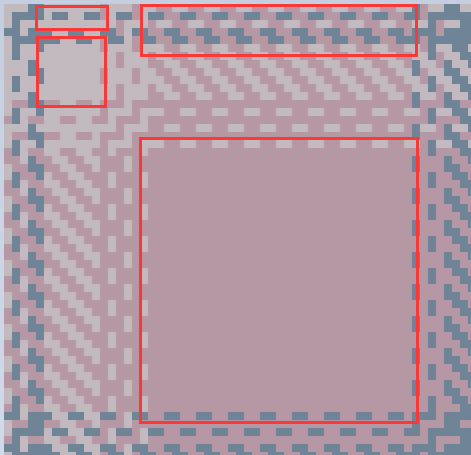
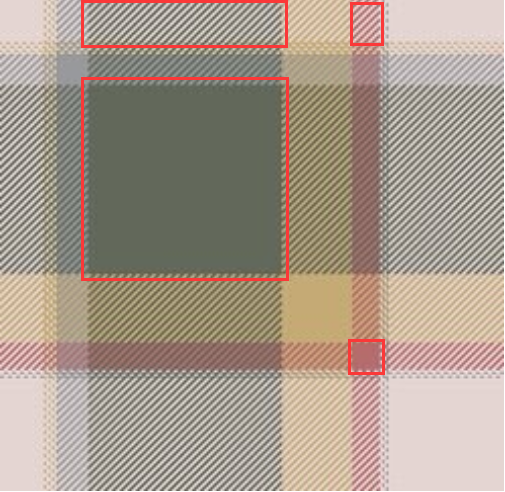
2. Packages **“rlist”** *(Ren, 2016)*, **“pracma**” *(Borchers, 2019)*, and **“base”** *(R Core Team, 2018)* are used for the data manipulation and numerical analysis.

5. Packages **“doSNOW”** *(Microsoft Corporation & Weston, 2019)* and **“parallel”** *(R Core Team, 2018)* are used for the parallel computing.

**Part 3. Sequential Tasking**

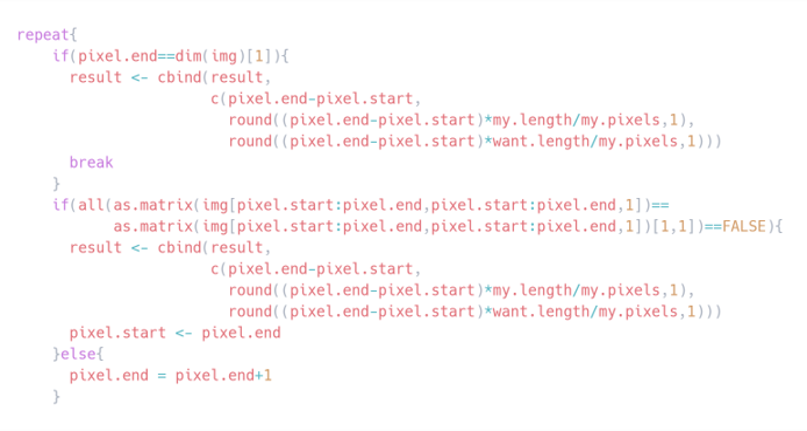
**3.1 Measuring**

Because the image is always the same either in the horizontal or vertical direction (diagonally symmetric), namely, when two identical patterns intersect, there will be a square. Also, these squares are all along the diagonal line of the image. We can see from the following figure 4 that the side length of the squares is exactly the length of the corresponding patterns:



**Figure 4.** Examples of the squares on the diagonal line

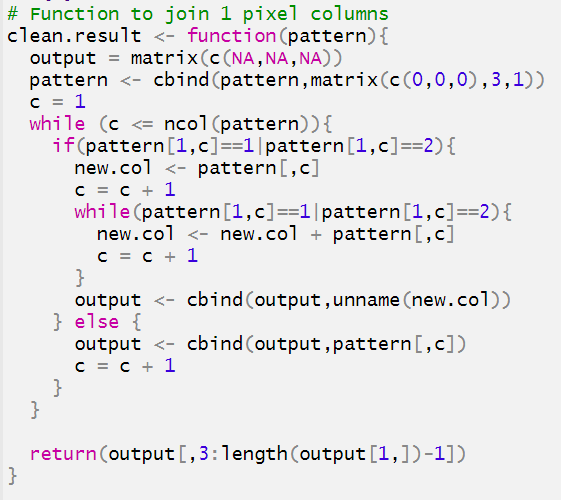
Thus, the general idea to check the lengths of patterns in the image is to check the size of the squares on the diagonal line. This idea can be realized with the following code:

****

**Figure 5.** Square measurement code

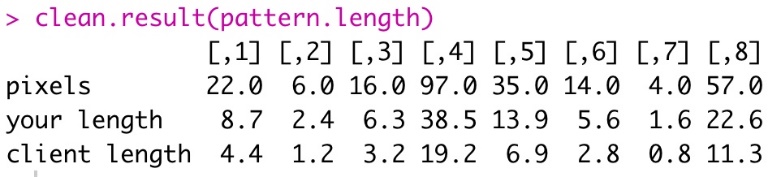
**3.1 Cleaning**

Considering that there are many single- or double-pixel patterns in the images, which may enlarge the size of the result, we decide to combine them together with the following code:



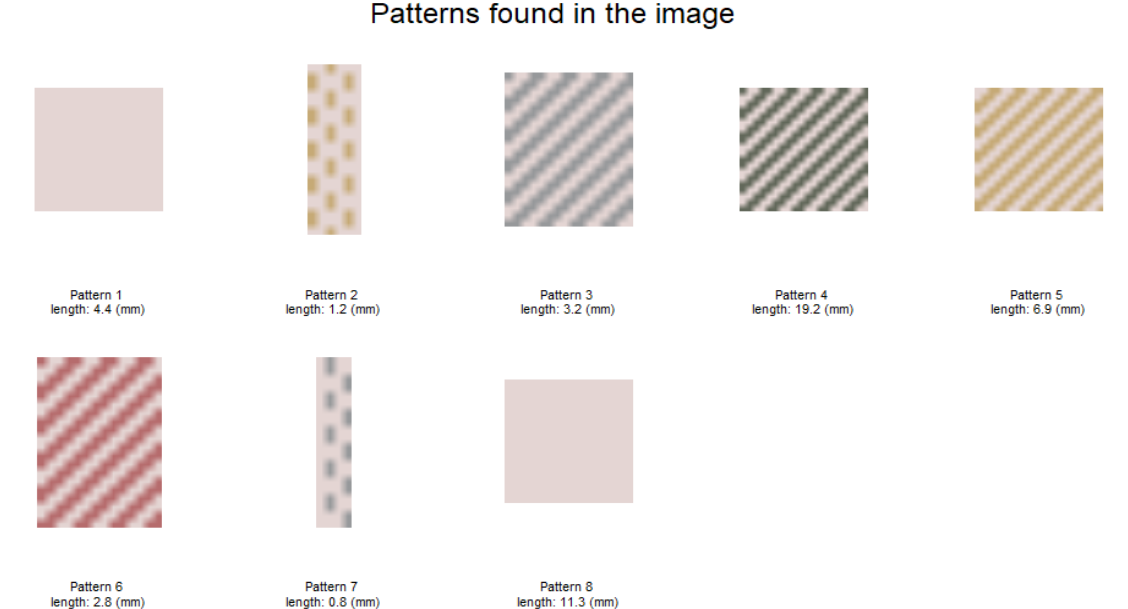
**Figure 6.** Cleaning code

The final results look like this:



**Figure 7.** Result of the measurement

We are able to display the size with the patterns to make them better visible:

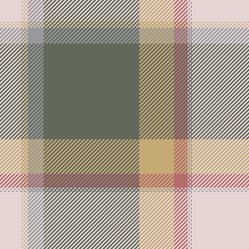


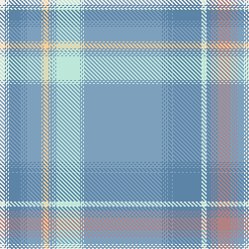
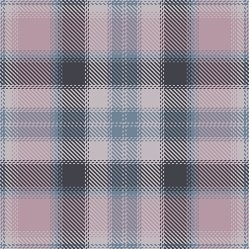
**Figure 8.** Final output of the patterns with lengths

Generally, this part covers all the work that the code does to deal with a single image. However, in real life the designers often need to achieve the information from a set of images instead of one or two. In this situation, a simple idea is to run the process above for the required time, but the parallel computing will be very useful for this work.

In the following part, we will introduce the method to apply parallel computing for this work, and we will also compare the difference of the time elapse between the two different methods.

**Part 4. Parallel Tasking**

**Figure 9.** Sample patterns used in the program

After creating necessary functions to retrieve and display patterns from an image, we increase the scale of the program to be able to read and analyze multiple images. One simple solution is to use a loop to go through images one by one, but it is not efficient. Thus, we implement parallel computing to be able to read images at the same time and reduce the time required to complete the task.

There are several parallel programming packages in R such as: “foreach”, “Snow”, multicore, “doMC”, “parallel”, and “doSNOW”. Each has its own strengths and weaknesses. We use “doSNOW” since it can be used in both Windows and Linux/OSX systems and is easy to implement. The “doSNOW” package is a "foreach" parallel adaptor for the snow package as it provides a parallel backend for the “%dopar%” function.

The codes are shown in detail in Figure 10 in the following pages. Our process for parallel computing is as follows. First, we create a function, “load.img”, to read all files with PNG extension in a folder and add them into a list so we will be able to go through the list in parallel.

Before running codes in parallel, we detect the number of cores in our computer and assign it as the number of clusters. Next, we make and register clusters using “makeClustor” and “registerDoSNOW” respectively. We use SOCK type because it is recommended for multi-core computers and simple Windows clusters, according to “doSNOW’s” documentation. Then, we use “foreach” with “%dopar%” expression to go through images in the list in parallel, apply our “get.patterns” function to retrieve the patterns length in pixel, and in millimeter. The “foreach” returns a list of patterns length for each image. Lastly, if needed, we can show patterns of any images with “display.patterns” function.



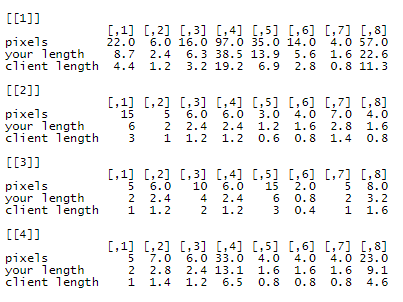
**Figure 10.** Important functions in this study

During our presentation, Professor Liu made a suggestion that we should include the load images function in our parallel part since loading images is a resource heavy part. Hence, we made a modification in our main code. However, our “display.patterns” function requires an image to draw its patterns. If we load images in the main “foreach”, we can’t return image data at the same time as patterns data. To remedy this, we load images in a different “foreach”. The code is in Figure 11.



**Figure 11.** Loading images in parallel

Figure 12 shows the output from parallel computing, only the first eight patterns in each image are displayed since the full results are long. The full results can be found in the appendix.

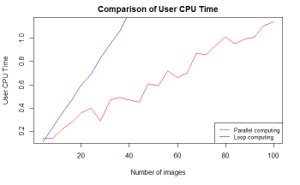


**Figure 12.** Results of parallel computing

**Part 5. Conclusion**

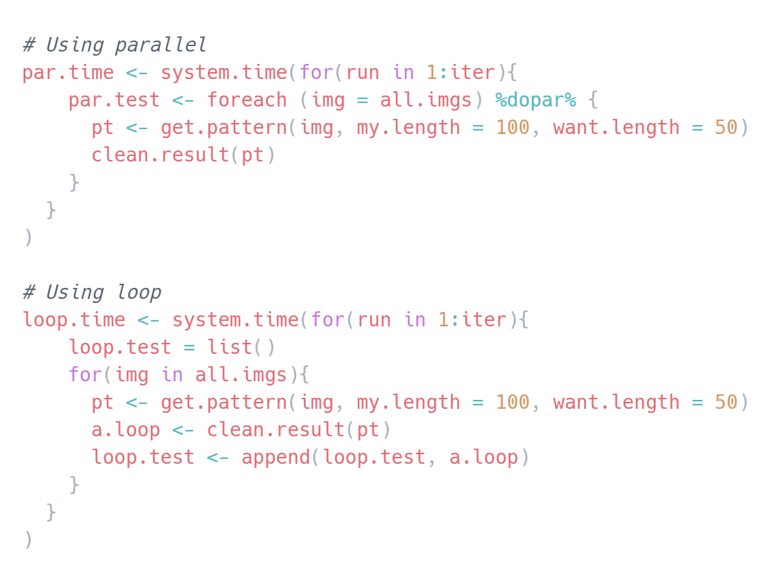
Despite the complexity in writing parallel programs and its high power consumption, parallel computing is predominantly utilized in performing large computational tasks. Parallel computing works by dividing the workload among multiple processors and by simultaneously solving multiple tasks. The team implemented parallel computing to deal with multiple pattern samples.

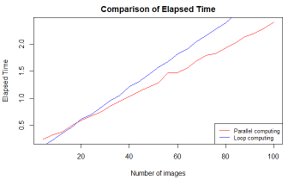
Figure 13 shows the comparison of user CPU time with x-axis being the number of images and y-axis being user CPU time. User CPU time is the CPU time spent by the kernel on behalf of the current process. As the number of images being processed increased, the difference in user CPU time between serial processing and parallel processing increased, with that of parallel computing being always lower.



**Figure 13.** Comparison of user CPU time.

Figure 14 shows the comparison of elapsed time with x-axis being the number of images and y-axis being the elapsed time. Elapsed time shows the real time passed since the process started. As the number of images being processed increased, the difference in elapsed time between serial processing and parallel processing increased, with that of parallel computing being always lower.

****



**Figure 14.** Code and result of comparison of elapsed time

In both cases, parallel computing is proved to be effective for running multiple tasks although parallel computing provides us with opportunities to speed up the process, there are also several limitations to using parallel computing. Apart from its high power consumption and programming complexity, it is also not efficient for a small number of parallel jobs Therefore, it is essential that parallel computing is used as per need. Some examples where parallel computing can be useful would be when there is a massive amount of items to process, or when there are multiple datasets to be processed.

***Reference***

*Microsoft Corporation, & Weston, S. (2019). doSNOW: Foreach Parallel Adaptor for the 'snow' Package. R package version 1.0.18.*

[*https://CRAN.R-project.org/package=doSNOW*](https://cran.r-project.org/package=doSNOW)

*Borchers. H. W. (2019). pracma: Practical Numerical Math Functions. R package version 2.2.9.*

[*https://CRAN.R-project.org/package=pracma*](https://cran.r-project.org/package=pracma)

*R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.*

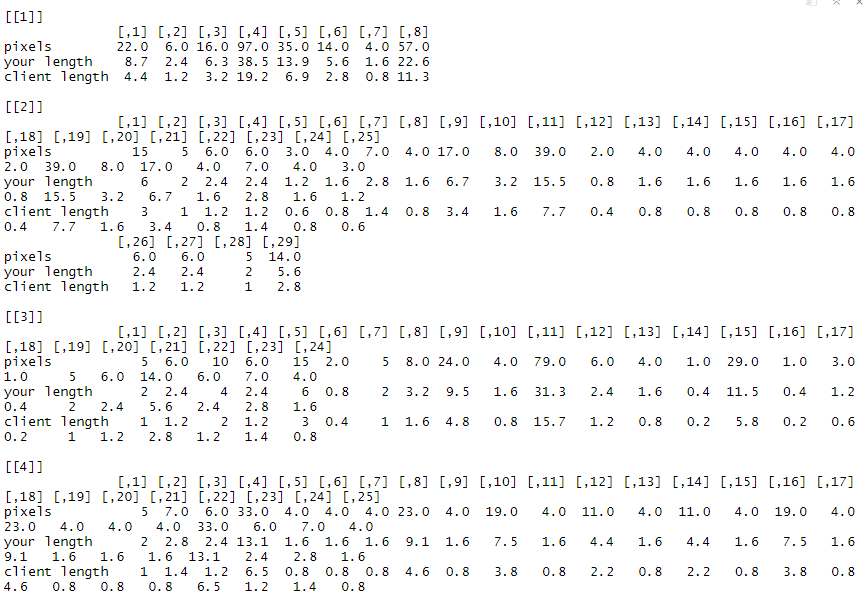
[*https://www.R-project.org/.*](https://www.r-project.org/.)

*Ren, K. (2016). rlist: A Toolbox for Non-Tabular Data Manipulation. R package version 0.4.6.1.*

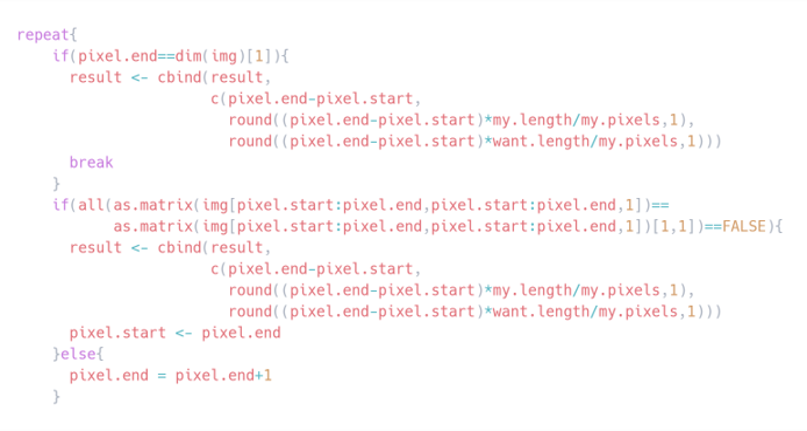
[*https://CRAN.R-project.org/package=rlist*](https://cran.r-project.org/package=rlist)

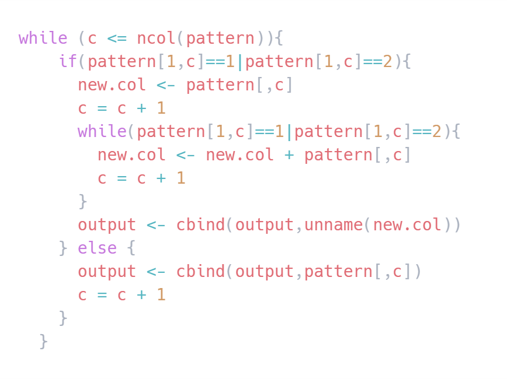
*Urbanek, S. (2013). png: Read and Write PNG Images. R Package Version 0.1–7.* [*https://CRAN.Rproject.org/package=png*](https://cran.rproject.org/package=png)

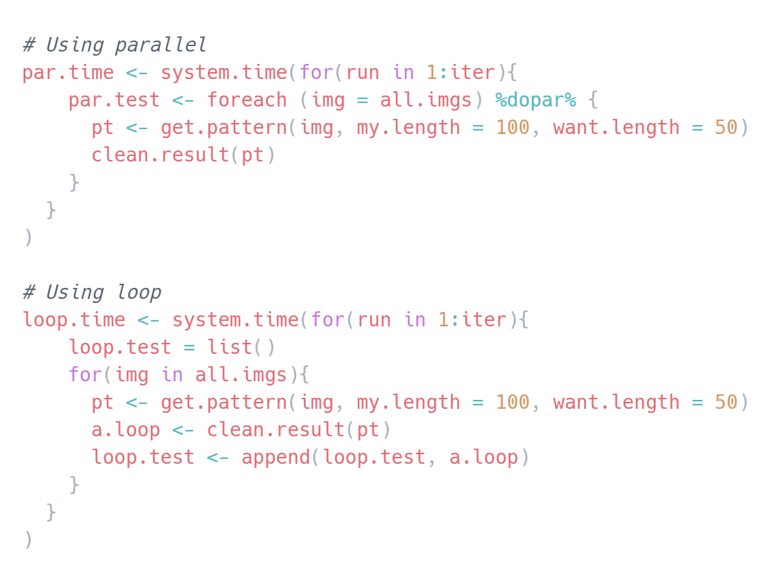
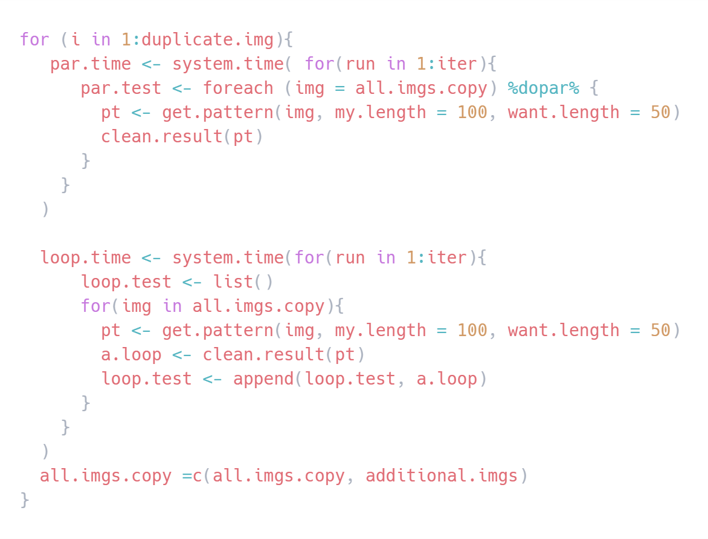
**Appendix A:** Full results



**Appendix B:** Code to generate full results

****

****

****