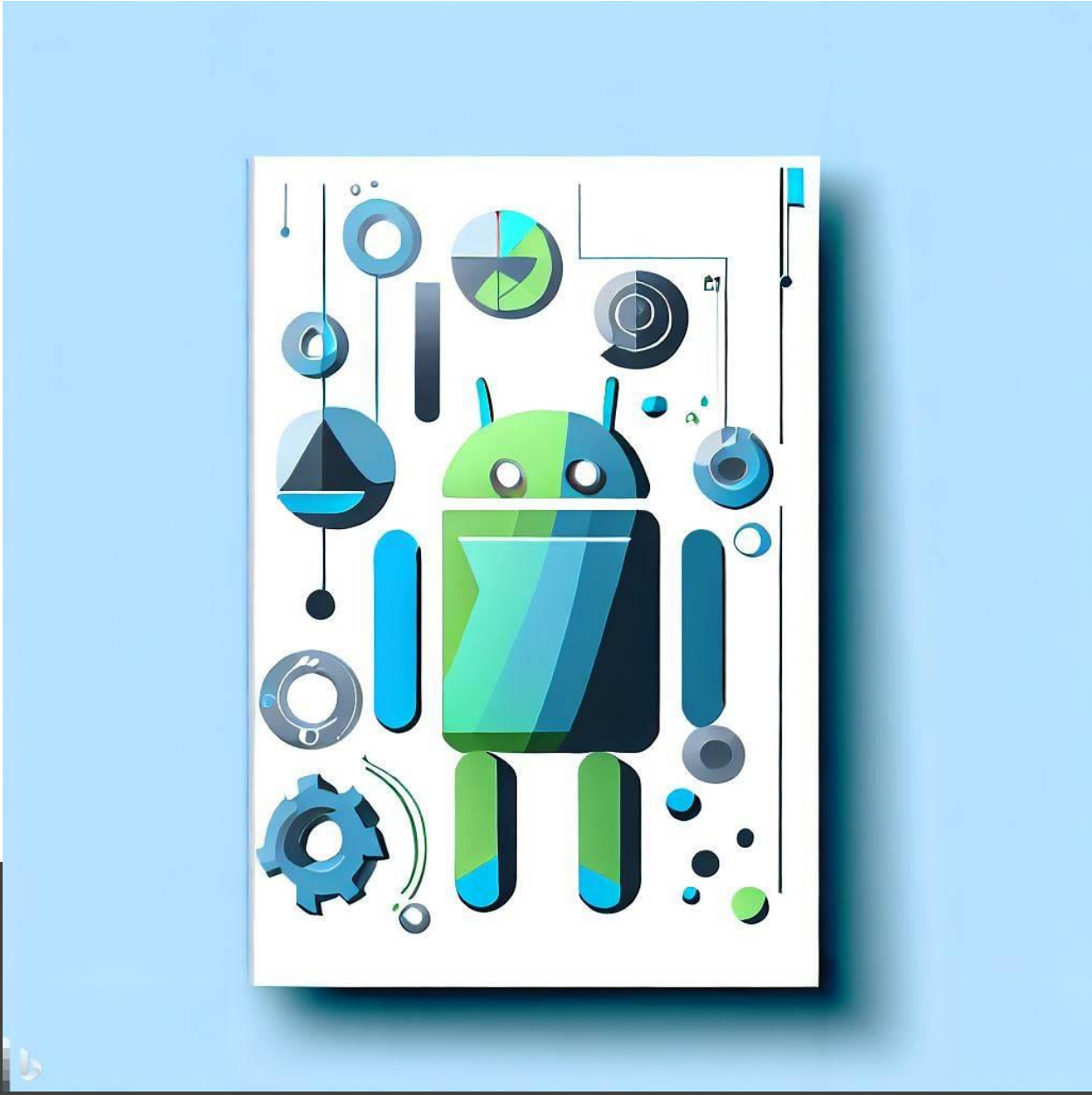
Giacomo Pedemonte - S4861715



Report (Mobile Dev.)

Serial VS Parallel execution in Android

Serial VS Parallel execution

Description

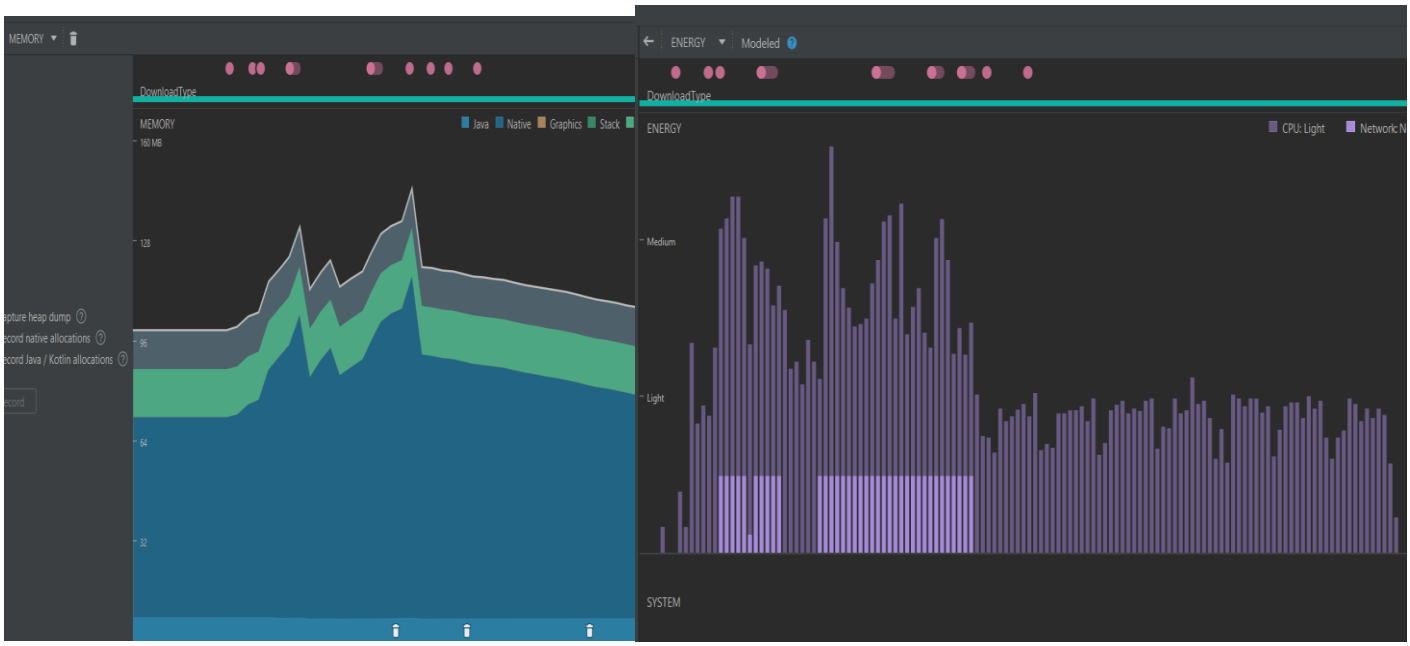
In this report, we have thoroughly analyzed the differences between serial and parallel execution on long-time computation tasks. It is crucial to organize these tasks well, or else they could cause the application of a user to not work as expected. Our findings have shown that parallel execution can significantly increase the speed of computation, but it also requires careful management of resources to avoid overloading the system. On the other hand, serial execution may be slower, but it is simpler to manage and less prone to errors. Ultimately, the choice between serial and parallel execution depends on the specific needs of the application and the available resources.

To assess the performance of serial versus parallel downloads, we will use the Profiler in Android Studio IDE while downloading multiple images. Additionally, we will test the impact of rotating each image by 20 degrees for 87 consecutive times after the download is complete. This will provide a clear distinction in the execution of the two methods.

Serial Execution

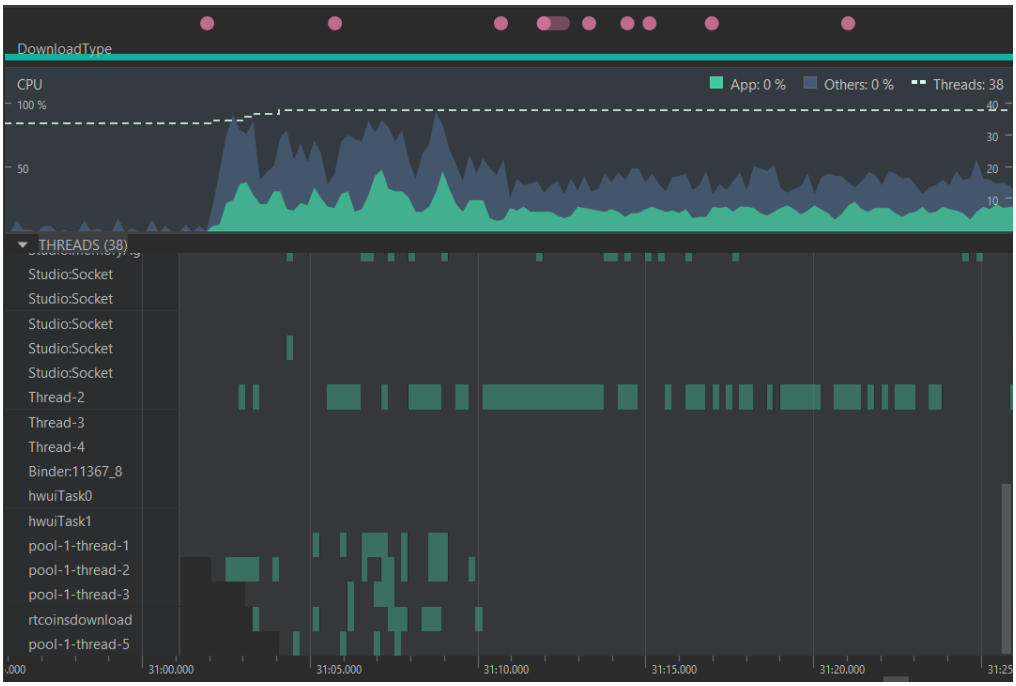
In this part of the report, we will analyze the serial part of the application starting by analyzing the following snapshot of the Profiler, to prove this we try to download all the images(touch one after the other starting from the top left):

The above image shows the CPU consumption in AsyncTask with serial execution. The AsyncTaskThread executes the tasks in serial order, resulting in the rotation of images being performed one after the other. This slows down the execution from the user's side and is not a practical way to organize downloads. To better understand this concept, the following images represent the Memory and Energy usage, allowing for a better comparison with parallel execution.



Parallel Execution

Now we can move to analyze the parallel part to find out the differences:



As seen in the above image, utilizing ThreadPool allows for more effective and efficient handling of downloads. In this instance, 4 threads were employed for both downloading and rotating the images. The AsyncTaskThread has been divided into this thread pool to enhance client-side performance. To confirm this, one can also examine the application's memory usage and energy consumption during the download process of the 9 images indicated by red dots in the profiler). It is clear that improved distribution and performance of memory and energy are important factors like reported in the following: