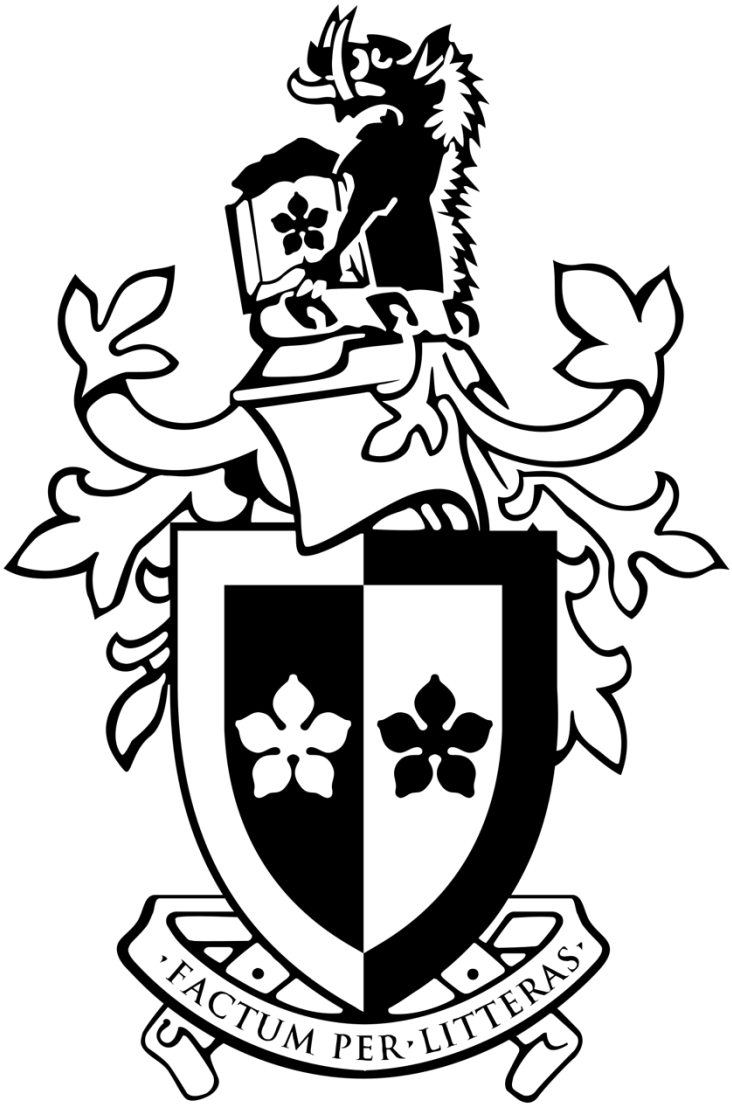
COS 30017
EXTENSION ON PERFORMANCE



Student name: Bill Pham
Student ID: 103826322

Introduces experiment.

Nowadays, mobile device is become a important part of our life. Almost everyone has for their own a mobile device, so the demand for efficient and user-friendly mobile app is becoming more and more crucial than ever. In term of efficient, the mobile application needs to take into account the performance aspect. Therefore, we will be going to analyse and monitor the performance of an app by profiler. This app is designed base on recycle view along with three scenarios which will result in different performance. This report will going to demonstrate the performance of each scenario then point out the best one and the worst one. Moreover, we will discuss how the performance can enhance by utilizing concurrency

Describe experiment

In this app, I will set the list up to 1000 lines also, I'm using the pixel 3a as the device to run on.

Profiler observation

Scenario 1: A constraint icon

For this first scenario, as we can see from the figure 1, the app starting up with relatively lower memory consumption, around 74 mb. CPU using the power at first for 15% to at the creation of the app.

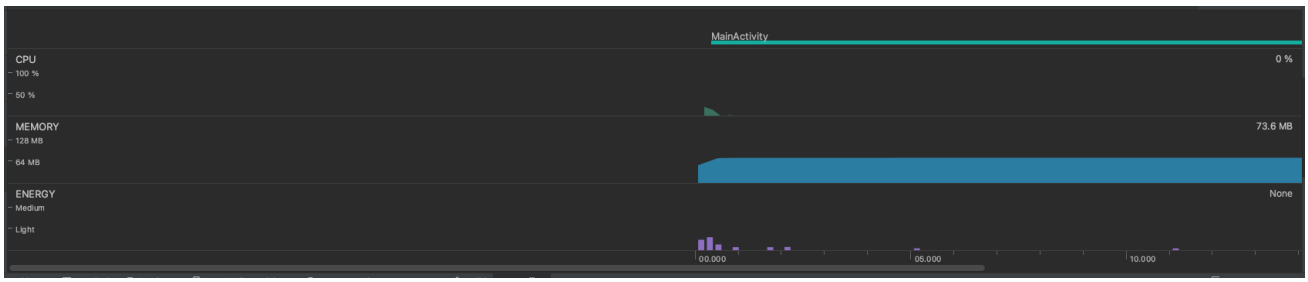


Figure 1 App starting

When we press the FAB for the first time, the recycle view automatically scroll down to the bottom of the apps. Meanwhile, the CPU will raise up to 20% max and keep consuming the power at the time the app automatic scrolling down. At the time pressing the button, the memory start rising up to max of 117mb and then reduce and stay stable at around 92mb. The energy is fluctuations during this action being carry out. It is noticeable that the memory is now allocated more than at the point we starting the apps

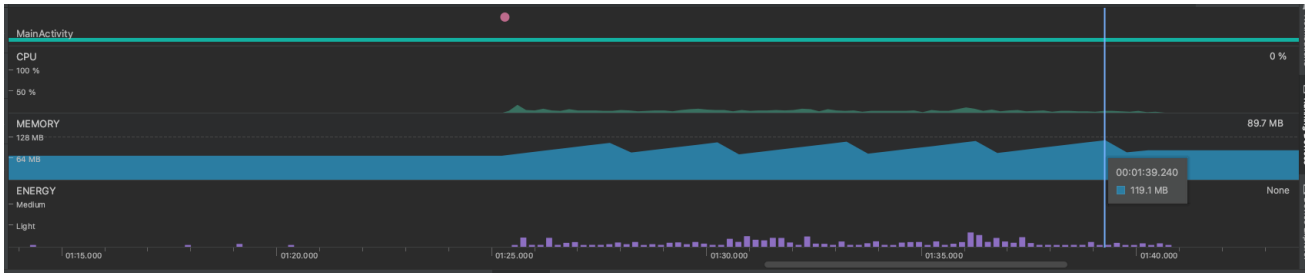


Figure 2 Pressing FAB for the first time

As we continuing to pressing the button multiple times after the first click. Right now, compared to the first click, the apps just use a bit of CPU power. Meanwhile, the memory is stable at 92mb. The energy that the app consuming is relatively low.

Figure 3 Pressing FAB serverall time

Scenario 2: A generated icon created on bind

As the same with the first scenario, the app starting up with low memory, just around 76mb

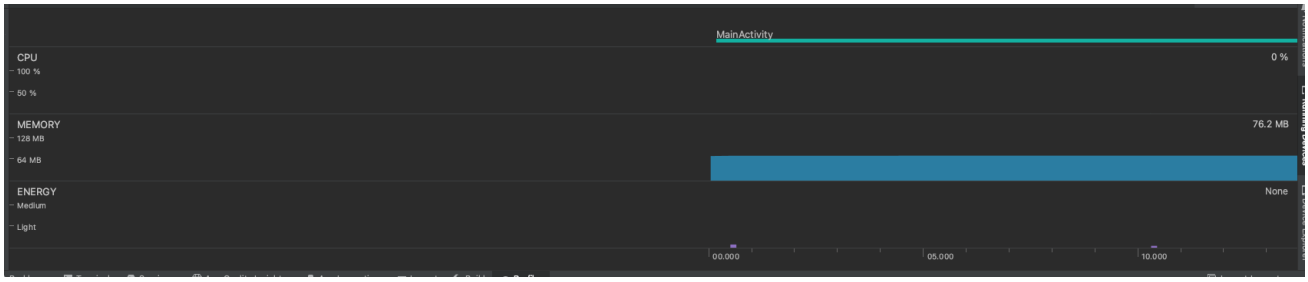


Figure 4 App starting

After pressing the floating action button for the first time of this scenario. Noticeably, the same things happen as scenario 1. The CPU consumption only at 13% at peak, however, compared to the first scenario, the CPU consumption at the first click is lower than the first scenario. Beside it, the memory of this scenario behaves the same as scenario 1 and keep stable at around 85 mb.

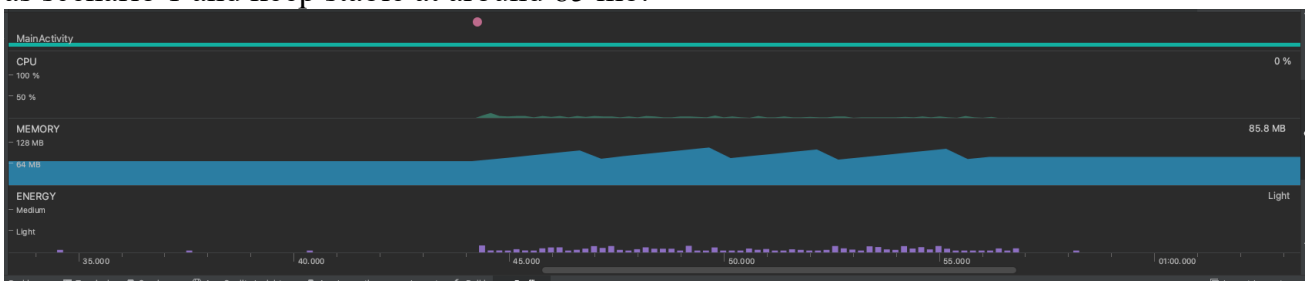


Figure 5 Pressing FAB for the first time

As when pressing the button for several time, the memory allocation gradually increase to around 93 mb and then stay stable at that. The CPU consumption is also similar to the first scenario. At this point, each time when we clicked, the CPU consume a bit, compared to the frist time pressing the button.

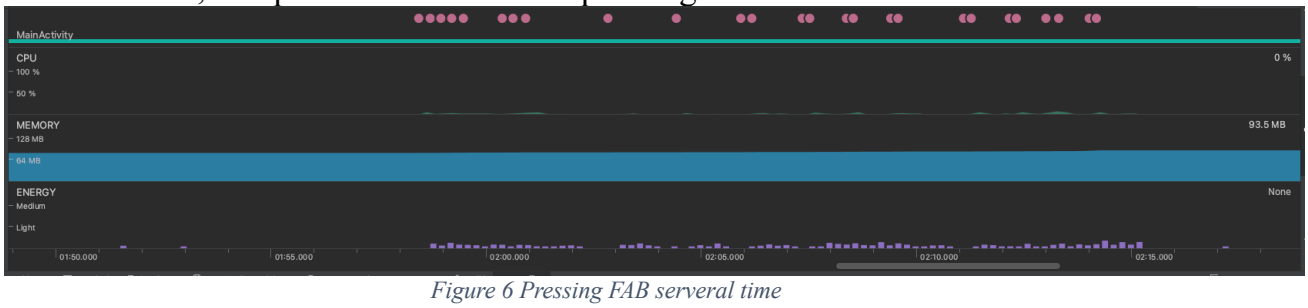


Figure 6 Pressing FAB serveral time

Scenario 3: A generated icon but created on initialisation:

As we start measuring using profiler, a instantly change happen. The memory stay at much higher level compare to first and second scenario, at around 223mb. Meanwhile, the CPU and energy consumption for at the start-up time stay the same with the two other scenario.

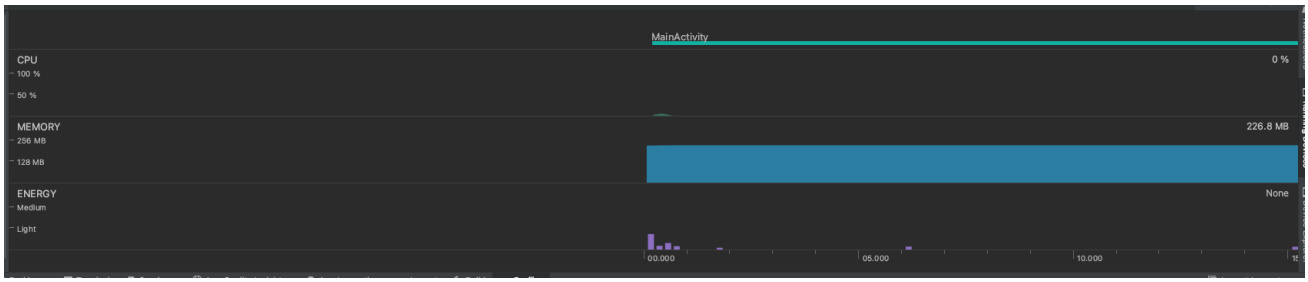


Figure 7 App startup

As we press the button for the first time. The CPU behave similar to two other scenario above. However, instead of the dynamic changes of the memory like other scenario, the memory gradually decrease and gradually increase again to around 223 mb, similar when starting up the app after the list is finish scrolling and add one more item to the list.

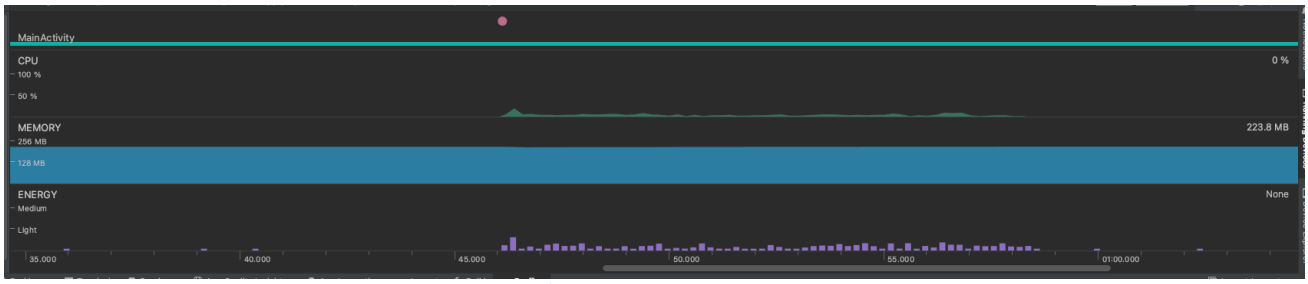


Figure 8 first time pressing FAB

When clicking multiple time and the screen is full of item adding by the FAB button, the memory slightly increase to 230mb and stay stable. CPU and energy consumption remain the same behaviour at this step as the two scenario above.

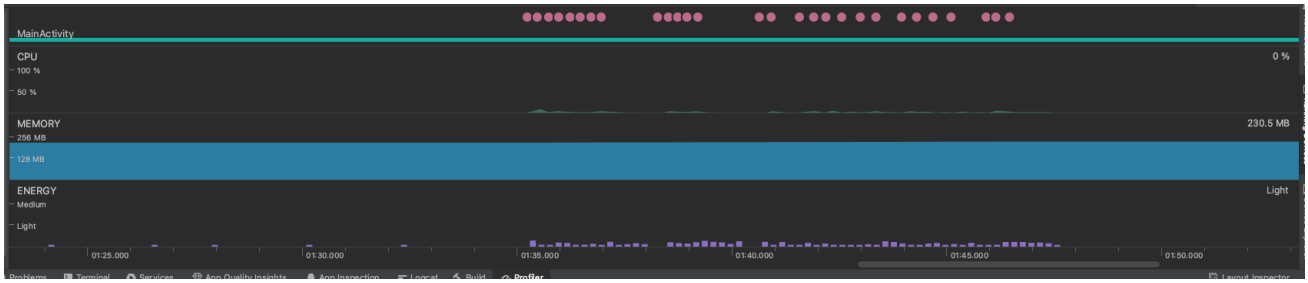


Figure 9 Pressing FAB button several time

Performance Analysis

Base on the observation of the profiler for each scenario performance. The first two scenarios performed much more better than the third scenarios in term of RAM usage. Both scenario 1 and 2 remain relatively low memory consumption rather than scenario

three. Due to generate the icon during initialization, the scenario three resulting in much higher memory usage.

Both scenario 1 and 2 have similar performance. However, a slightly better in memory usage of scenario 2 is being observed. Because of the icon is generated dynamically during binding, which lead to lower initial memory usage. The main reason for the low memory usage due to item.icon in my Unit is null, which indicate that the application is not require to process the image for the recycle view. Moreover, the memory is able to release it and come back to it initial state (with one more item add it resulting in the memory release is a bit higher than the start up memory)

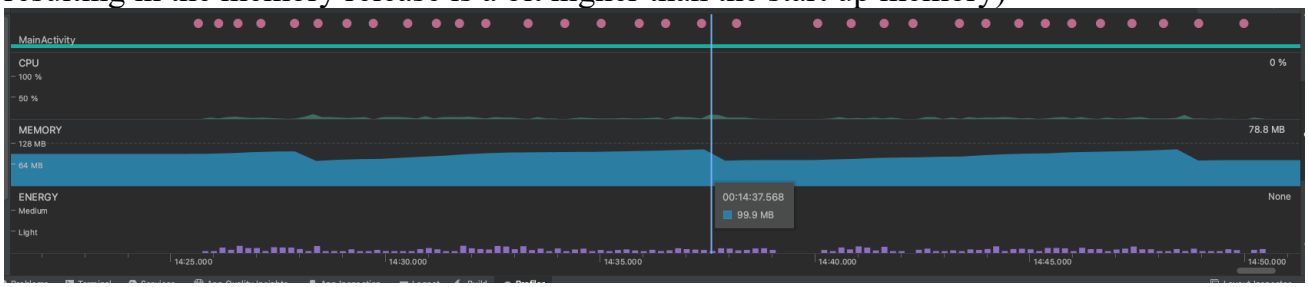


Figure 10 Scenario 2 memory release

In contrast, the wost performance will definitely the third scenario, which utilize too much of memory. Meanwhile, the application cannot automatically release the memory. As we observe above, the memory will constantly increase without being released. This in practical could lead to memory leak and related memory problem. This might be consider as a bad practice for any case.

As we experiment though three scenarios with different types of approach to generate a image (icon). The primary takeaway from this task is how we manage the performance of an apps and how important to manage it carefully. Moreover, testing and profiling of the application can provide a deeper look in to the performance of the app. This tool can determine various of issues related to memory, CPU usage, and energy consumption.

In case of concurrent approach, concurrency can significantly improve the performance of loading a large number of item into a list. Concurrency allow the application to parallel processing the task, enabling multiple items can be loaded concurrently from the file. The app can load the data in the background by utilizing multiple threads which ensure the UI still running. Overall, by using concurrency approach, we can take advantage of the available hardware resource, enhance the application's performance, improve data retrieval speeds, and provide users with a more responsive and seamless experience

Conclusion

In conclusion, mobile software is not only a working application, but we also need take into account the performance of the apps. With the limited resources on the mobile, developer need to optimize the performance of the app as well as ensure a seamlessly experience for the end user. Take in to experiment three scenarios in this report, we have found out the best scenario and the worst scenario regarding application performance. Concurrency, which is like doing multiple things at once, plays a key role in making apps not only function effectively but also operate efficiently, meeting the expectations of a dynamic and ever-evolving user base.