**User Impatience Application**

User Impatience is an application that caters performance to the desire of the user. It acts as a power governor on a mobile device that allows the user to get more power from the device when they need it or reduce power when they don’t need it. As of now, governors require user interaction to get the performance that they require. The goal of User Impatience is to minimize the requirement of that interaction. As the user interacts with the governor the User Impatience application will record the data. This data will then be used to cater performance to the intent of the user. Machine Learning will be considered for the purpose of optimizing the functionality of the application. Meanwhile, the current system for testing is not ideal and relies on random numbers and power measurements to relay when an app opens. These variables may add to the power consumption in our testing and add error in our current testing framework. Therefore, we will also need to focus on bettering the testing to minimize outside error that we have to account for.

**Virtual User Evaluation System**

A virtual user is a faux user that represents the different type of possible users that would utilize the User Impatience application. The intention of the virtual user is to test for varying user intentions for the purpose of statistical significance. This testing has the goal of optimizing the functionality for all possible users and to develop the application that it can sufficiently learn and, as such, minimize overhead and minimize user involvement with the application.

At present, the virtual user is a simplistic system of random number evaluation. This will be further developed to more accurately capture differing user intentions and to provide more accurate information to the system. Given that the pre-existing virtual user system was designed in C++ and ran on Linux, the same language and system will be used to develop it further.

**Values**

1. **Tolerance Interval**: The amount of time that passes until the system attempts to underclock. This can also be considered the amount of time to evaluate that the user ‘tolerates’ the current performance level.
2. **Impatience Interval**: The amount of time that passes until the user ‘might’ want to increase performance.
3. **Minimum**: Minimum performance
4. **Maximum**: Maximum performance
5. **Current:** The current performance level.
6. **Underclock**: The amount to underclock performance to.
7. **Overclock:** The amount to overclock performance to.

For values 1 and 2, the tolerance interval should be longer than the impatience interval. The system should give the user enough time to determine if that user is satisfied with the current performance level. If the tolerance interval is faster than the impatience interval, then the system will underclock before evaluating the user’s patience and will result in an inaccurate evaluation. Furthermore, if the user has a high impatience level, that is a short impatience interval, then setting an equally short tolerance level would be in conflict to this user’s design.

For values 3 and 4, the minimum value would be initially set to a default value, based on a functional minimum for a given application. However, this value will change according to user evaluation. If the user demonstrates impatience at a performance level, that level will become the new minimum. The maximum value will be initially set to a default value, based on the maximum possible performance of the hardware.

For values 5, 6, and 7, initially the current performance value will be set to the midpoint between the minimum and maximum values. When the system underclocks the performance will be set to the midpoint between the current performance level and the minimum performance value. When the user wants to overclock, the performance level will be set to the midpoint between the current performance level, which is the new minimum, and the maximum value. By setting the performance to a midpoint, you reduce the amount of evaluation that the system must do. This reduced evaluation would result in finding the minimum tolerance level more quickly and would result in the system no longer needing to evaluate for underclocking, only overclocking, reducing overhead.

This project allows each of us to contribute in areas of interest. Professor Mosse and Raphael have walked us through some of the potential additions they would like to make to the existing codebase. These included an opportunity to implement the algorithm in the kernel, which would take out the intermediary steps of running code from the computer to the phone. Another idea was an algorithm that would make the adjustment based on type of user and their previous trends. Therefore, we can all find areas in the project that interest us.

One area that initially interested our group and will be an important aspect of the general project is the user evaluation system. We need to find a way to understand a user’s past trends with the apps they’ve used and how impatient they tend to be. If we are able to create a more efficient system to evaluate users we can minimize the amount of times they will complain about their phone’s speed. If we are able to implement this better we will also know where the starting point should be and the most likely areas for user impatience.

We plan to communicate through a slack channel, email and over text messages. To test the system we currently need to have the physical setup, but we are exploring ideas to set up an ssh connection or having a second setup. Some of the issues we may encounter is our collective lack of experience with android, python and C++. Raphael is currently running the project on linux. We aren’t sure if it will work on our Macs so we will need to find that out and otherwise create a virtual machine or install linux. However, we will have constant communication with each other, professor Mosse, and Raphael. We plan to meet with them weekly or every other week depending on our progress and what times people are available.