User Input vs Hardware Events

Nithin Krishnan, Zhonghan Li

Introduction

As designing faster chips and better batteries becomes increasingly difficult, other methods of leveraging performance for power must be examined. Solutions such as increasing core count may be useful for some workloads, but the end goal for any computer is to satisfy the user's needs. However, understanding the happiness of the user during the run of an application is complicated. Although it may be possible to query the user frequently, such explicit interaction will annoy most users. Therefore, it would be beneficial to estimate user satisfaction using implicit metrics. Traditionally, computer architects have used implicit metrics such as instructions retired per second (IPS), processor frequency, or the instructions per cycle (IPC) as optimization objectives. The assumption behind these metrics is that they relate in a simple way to the satisfaction of the user. However, we take a look at Hardware Performance Counters(HPC) that are inherently present in several mobile devices to measure the effect of the user-input on the hardware metrics.

Background

Previous studies have shown that user satisfaction is closely linked to hardware performance and, more importantly, is highly dependent on the user in question. This suggests that a system that is tailored to a user or can learn a user's habits and predict them can increase overall satisfaction. The question then becomes: what information that can be obtained about the user is helpful in determining architecture optimizations? Past research has established a feedback system that monitors the user's physiological changes relative to performance, then employs Dynamic Voltage and Frequency Scaling(DVFS) to manage power consumption while retaining user satisfaction. This requires extra hardware to sense the user's physiological state but may make up for it in more fine-tuned power optimization. Another approach may focus on the user's input, as this information is more readily available. One example is an energy optimization framework that could take advantage of user slack periods: perceptual, cognitive and motor delays between user and computer interface, to reduce CPU frequency.

User-Input

One challenge is determining how user input should be defined for the purposes of drawing correlations. More inputs generally mean more hardware events. It is difficult to see trends in raw user input data and quantitatively decide if a certain input caused a hardware event.
n order to go around this, we ensured that regardless of the method of input, the final state of our system was the same. If different methods of entering the same data leads to the same end result, then it becomes easier to analyse the HPCs based on the input.

Tools and Experiment Setup:

The main tools used for this experiment were:

>System (LG Nexus 5)

> Repetitouch

> Android Debug Bridge(ADB)

> Simpleperf

LG Nexus 5: Our test device was an LG nexus 5 running a 2.26Ghz quad-core Snapdragon 800 processor. We had to enable root access in our system to be able to measure the outputs of the HPCs and record the data.

Repetitouch: Is an application we found on the play store, which enables us to record a set of inputs and replay them. This helps in removing human error in providing the inputs.

Android Debug Bridge(ADB): Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. It is a client-server program that includes three components:

I.A client, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.

II.A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.

III.A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

Simpleperf: Simpleperf is a native profiling tool for Android. Its command-line interface supports broadly the same options as the linux-tools perf, but also supports various Android-specific improvements.

Modern cpus have a hardware component called the performance monitoring unit (PMU). The PMU has several hardware performance counters(HPC), counting events like how many cpu cycles have happened, how many instructions have executed, or how many cache misses have happened.

Setup:

With the help of the repetitouch application, the input sequence is recorded and stored. The particular thread on which the input is meant to run is reset and brought to an initial state. Once the input and application thread is ready, we run the simpleperf command stat to measure all the Hardware Performance Counters supported by simpleperf for a set duration. The amount of time the command has to run should not be too small that all the inputs are not executed, at the same time, it should not be so large that some background processes can majorly affect the HPCs. After the HPC values are returned by simpleperf for the required period of time, We log those values and proceed to repeat the experiment for a total of ten times. The mean of all the HPCs for every experiment is calculated.

Experimental Results and Analysis

After obtaining data from 10 runs of each mode of input for each experiment, we summarized the data and obtained statistics on the number of hardware events per input method for each task. Our results are outlined here.

Experiment 1

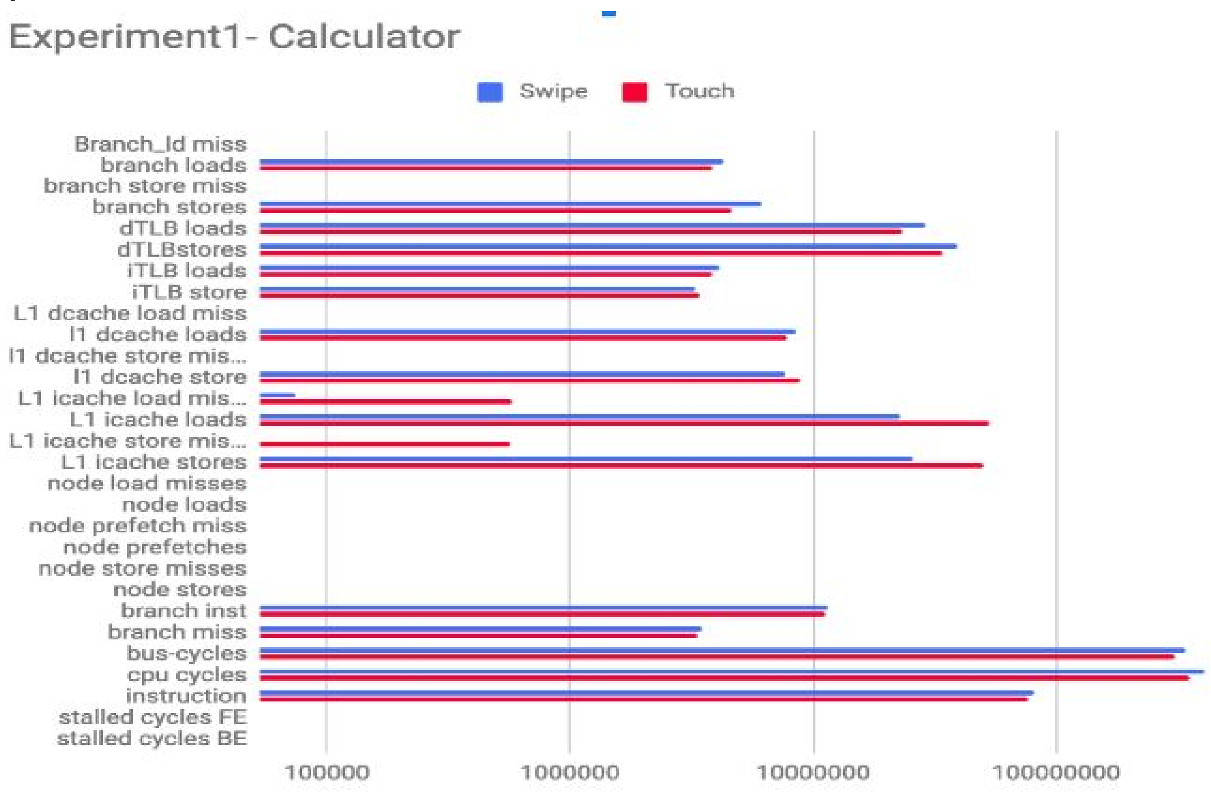


Fig. Means of number of hardware events for the experiment using the native Calculator app.

Our first experiment utilized the phone's native calculator application. One run consisted of mainly touch presses while the other consisted of swipes. To maintain some consistency, the runs performed exactly the same calculations within the same rough timespan. The means of most events seem pretty similar, with the exception of the L1 cache statistics.

Hardware Metric	P-Value
Branch_ld miss	
branch loads	0.4952304759
branch store miss	
branch stores	0.01941314668
dTLB loads	0.04993816053
dTLBstores	0.1641444532
iTLB loads	0.4481493418
iTLB store	0.4813116829
L1 dcache load miss	
l1 dcache loads	0.4139615757
l1 dcache store misses	
l1 dcache store	0.2590417342
L1 icache load misses	0.02070563877
L1 icache loads	0.01368879905
L1 icache store misses	0.0266030801
L1 icache stores	0.05119837902
node load misses	
node loads	
node prefetch miss	
node prefetches	
node store misses	
node stores	
branch inst	0.768657743
branch miss	0.5847102253
bus-cycles	0.2731624408
cpu cycles	0.13273171
instruction	0.560766749
stalled cycles FE	
stalled cycles BE	

Table 1. T-test P values for the calculator experiment.

Although we were able to report on the means of number of hardware events for each experiment, there was much variation between the number of hardware events for each run, despite measures to isolate the application in question such as closing all other applications and turning off network connection to prevent random network packets. We decided it was vital to run some T-tests to see if these average differences are the result of chance.

Table 1. shows the results of a T-Test performed on the two ranges of data in order to determine whether they are statistically different or not. Our null hypothesis is that the two means are not statistically different. As we can see from the above table, most of the p-values are fairly large. Based on an alpha of 0.05, we fail to reject the null hypothesis for most of the runs, meaning there is no significant difference between the means of the number of hardware events resulting from the two different methods of input. The only area that presented a difference is the L1 icache statistics, though the reasoning behind this is difficult to examine.

Experiment 2

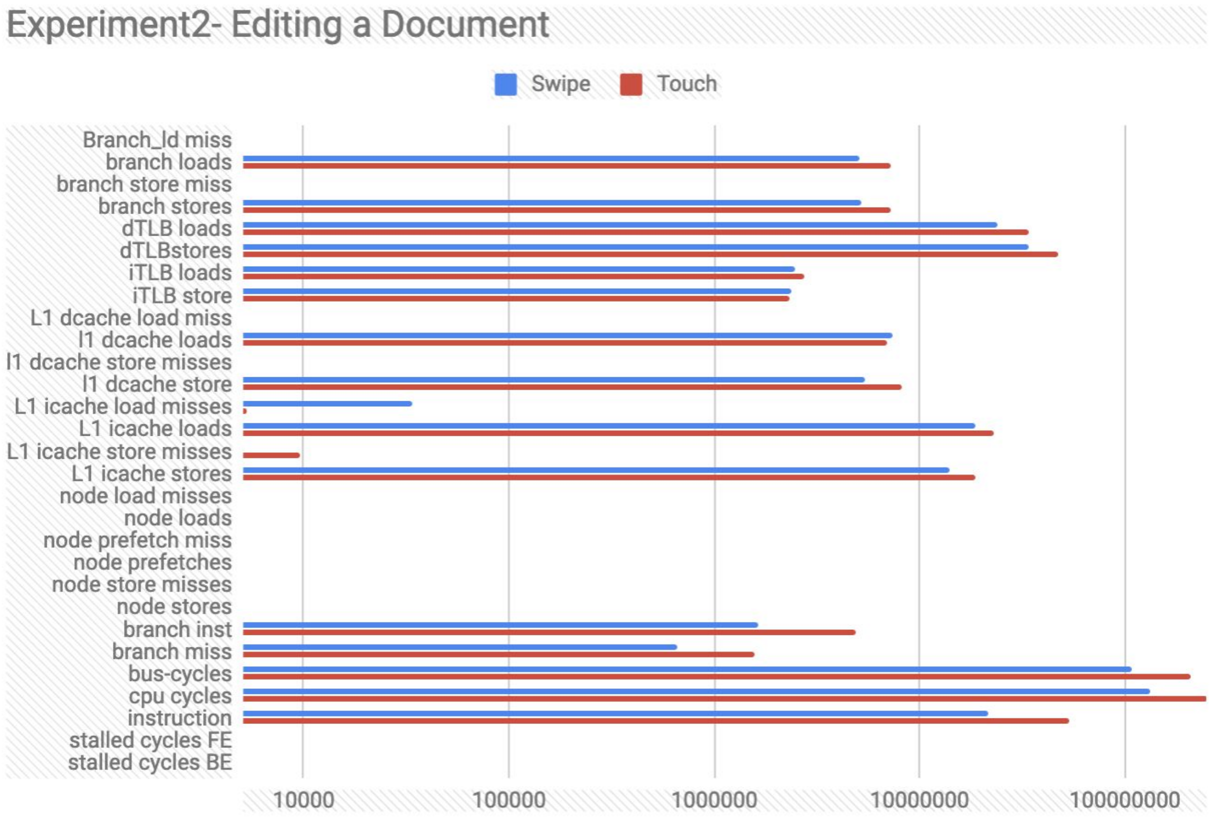
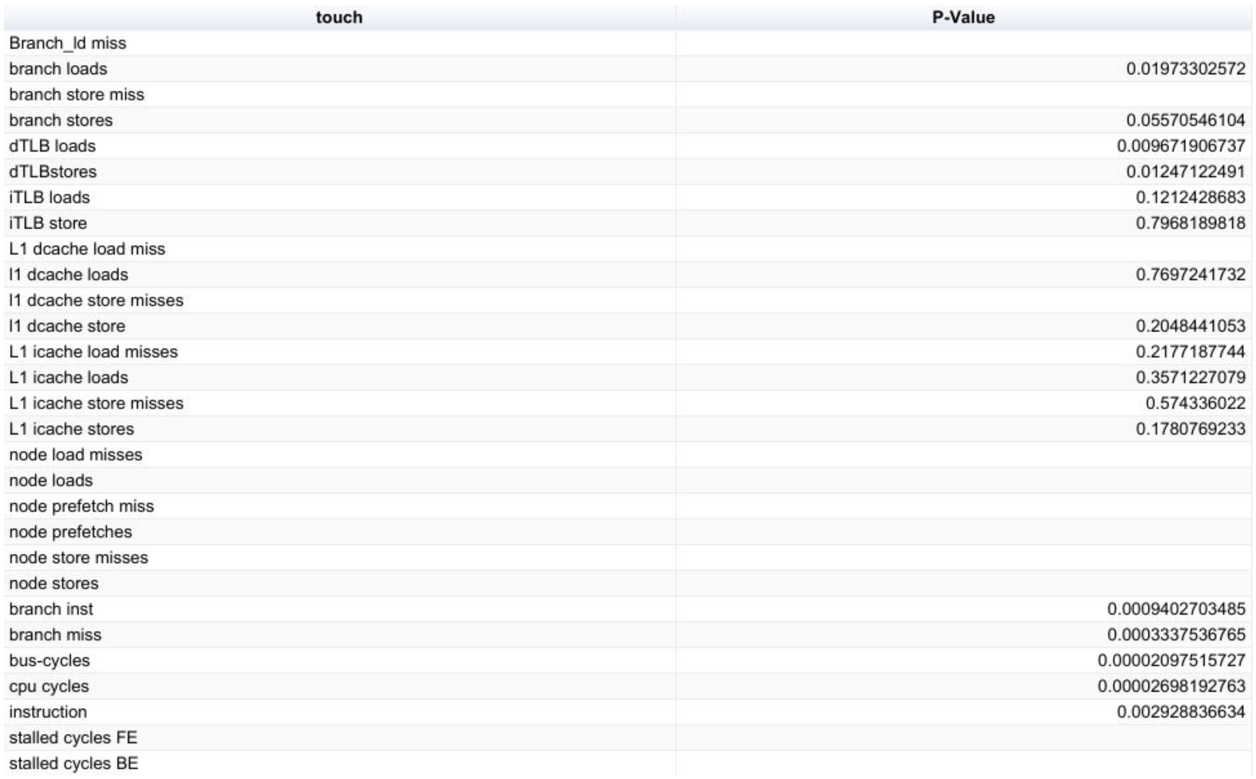


Figure 2. Means of number of hardware events with document editor application.

Our second experiment consists of recording the number of hardware events for different usages of a document editor. The first run is normal touch input, while the second run uses the native android swipe typing functionality, with the same sentence being typed for both runs. The means for the touch typing seem to be higher than those of the swipe typing.



touch	P-Value
Branch_ld miss	
branch loads	0.01973302572
branch store miss	
branch stores	0.05570546104
dTLB loads	0.009671906737
dTLBstores	0.01247122491
iTLB loads	0.1212428683
iTLB store	0.7968189818
L1 dcache load miss	
l1 dcache loads	0.7697241732
l1 dcache store misses	
l1 dcache store	0.2048441053
L1 icache load misses	0.2177187744
L1 icache loads	0.3571227079
L1 icache store misses	0.574336022
L1 icache stores	0.1780769233
node load misses	
node loads	
node prefetch miss	
node prefetches	
node store misses	
node stores	
branch inst	0.0009402703485
branch miss	0.0003337536765
bus-cycles	0.00002097515727
cpu cycles	0.00002698192763
instruction	0.002928836634
stalled cycles FE	
stalled cycles BE	

Table 2. T-test results for the document editor experiment.

The T-test results for this experiment indicated in Table 2 show no significant different in hardware events for cache metrics, but the p-values for instructions, cpu cycles, bus cycles, branch instructions and branch misses are very low, meaning there is a significant difference between the means of the hardware events for these counters.

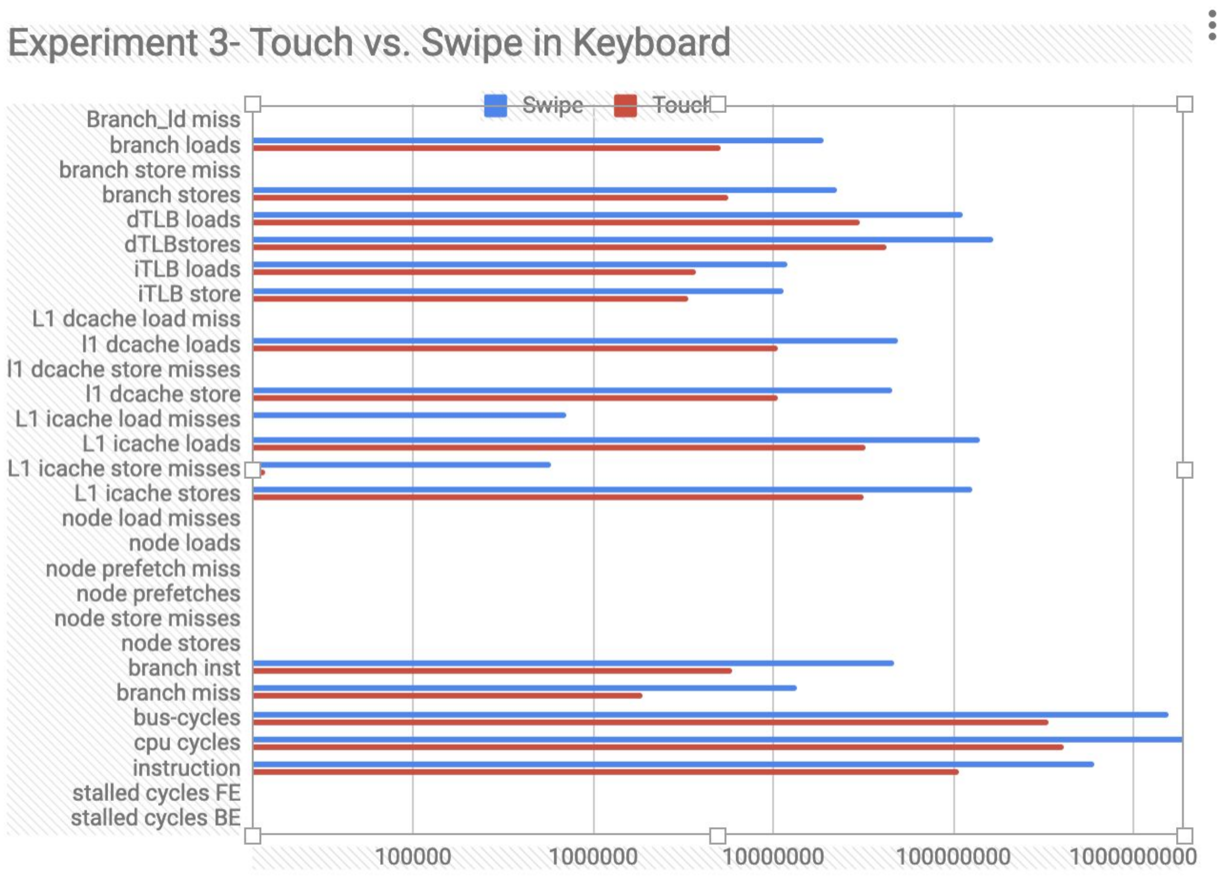


Figure 3. Means of hardware events with keyboard process.

While the second experiment seemed to yield little variation with respect to the number of hardware events generated by the document editor based on method of input, we thought it would be interesting to examine if there was a difference for the keyboard application for the same input patterns. The results indicate that the means for swipes in this experiment seem much higher across the board.

Swipe	P-Value
Branch_ld miss	
branch loads	0.00000006116641221
branch store miss	
branch stores	0.0000000002070768192
dTLB loads	0.0000000007580008976
dTLBstores	0.000000002160528557
iTLB loads	0
iTLB store	0
L1 dcache load miss	
l1 dcache loads	0.0000001546511264
l1 dcache store misses	
l1 dcache store	0.00000000238282489
L1 icache load misses	0.002708166031
L1 icache loads	0.00000002955091169
L1 icache store misses	0.00838325817
L1 icache stores	0.00000005207362509
node load misses	
node loads	
node prefetch miss	
node prefetches	
node store misses	
node stores	
branch inst	0.0000000001466764693
branch miss	0
bus-cycles	0
cpu cycles	0
instruction	0.0000000003563317788
stalled cycles FE	
stalled cycles BE	

Table 3. T-test results for keyboard application experiment.

Our T-tests result in extremely low p-values across the board, meaning there is a statistical difference between the number of hardware events that occur for different usages of the native android keyboard. While the exact reasoning is difficult to pinpoint without examining the source code, one potential cause could be the fact that the keyboard has to do extra computation to match the swipe to a word. The keyboard also has some learning features built in that make it more user friendly, so each word swipe could also be taken into account for future predictions.

Experiment 4

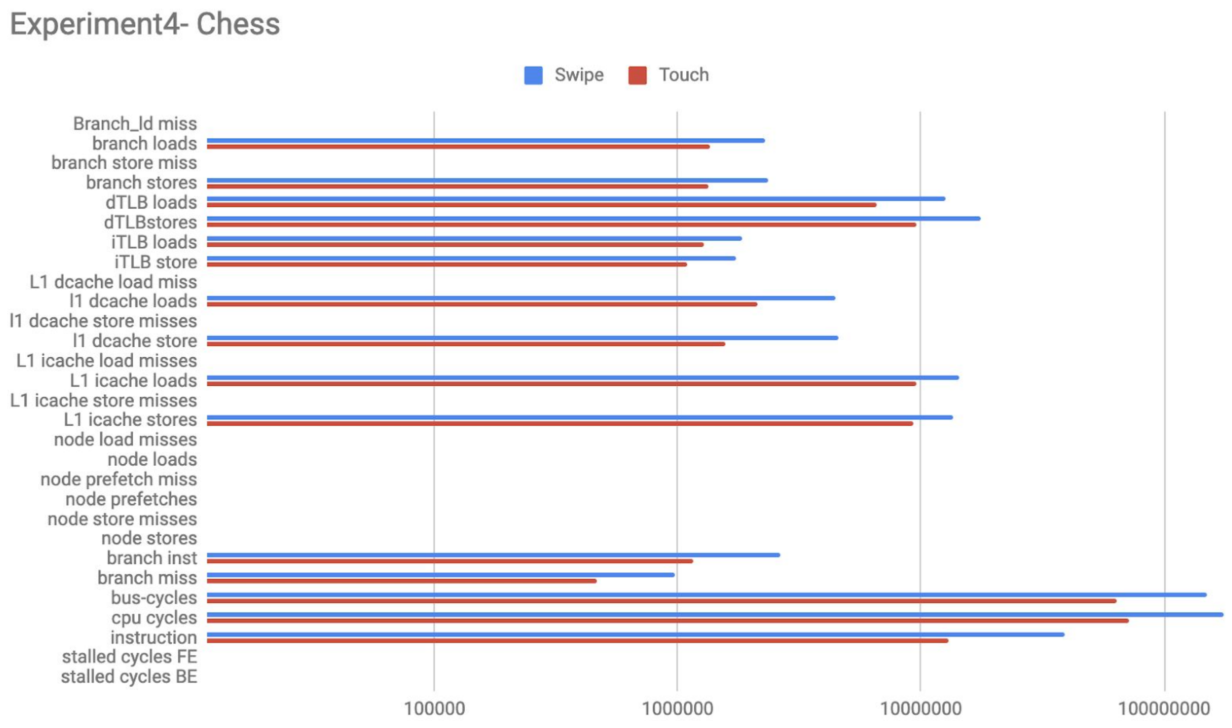
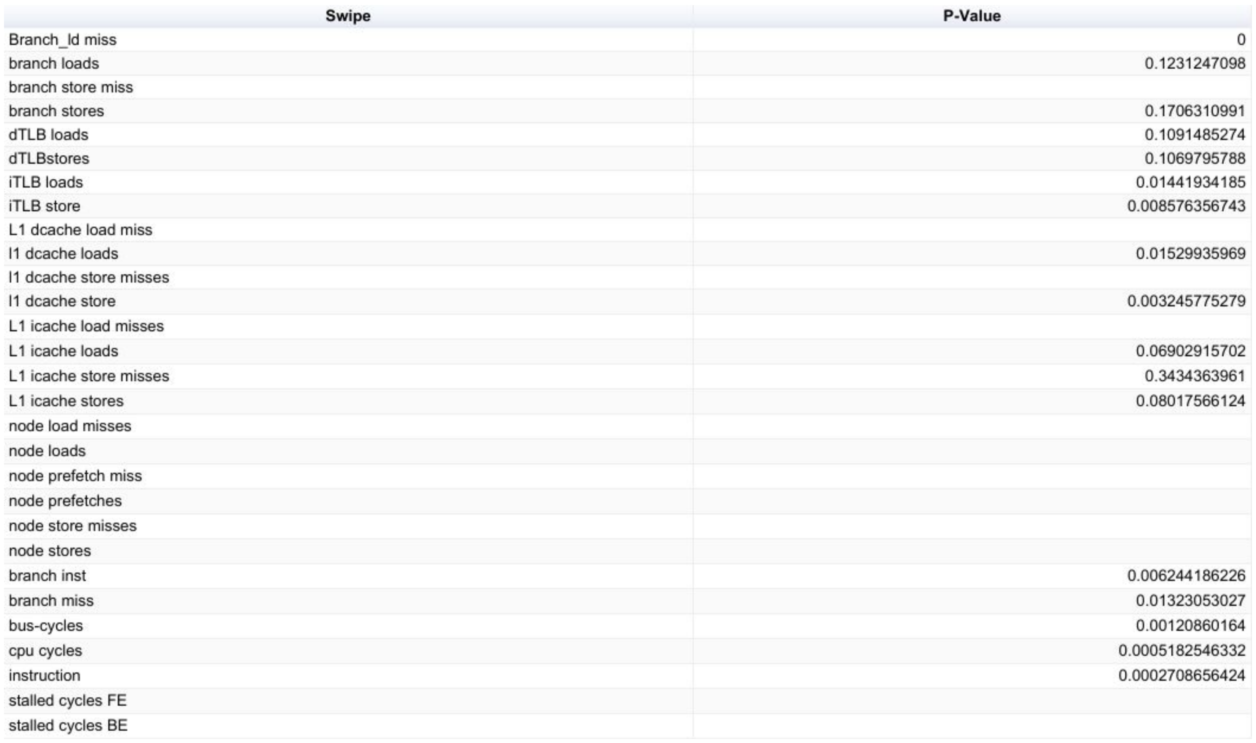


Figure 4. Means of hardware events for game application.

Our fourth experiment involved a game, which we chose because they tend to have many varying possibilities for inputs. After some deliberation, we decided upon a chess game set to two player mode to reduce the possibility of random events occurring and skewing the hardware performance counter readings. The first run was comprised of touch inputs for moves while the second run had swipe inputs. The means indicate higher numbers of hardware events in general for the swipe input.



Swipe	P-Value
Branch_ld miss	0
branch loads	0.1231247098
branch store miss	
branch stores	0.1706310991
dTLB loads	0.1091485274
dTLBstores	0.1069795788
iTLB loads	0.01441934185
iTLB store	0.008576356743
L1 dcache load miss	
l1 dcache loads	0.01529935969
l1 dcache store misses	
l1 dcache store	0.003245775279
L1 icache load misses	
L1 icache loads	0.06902915702
L1 icache store misses	0.3434363961
L1 icache stores	0.08017566124
node load misses	
node loads	
node prefetch miss	
node prefetches	
node store misses	
node stores	
branch inst	0.006244186226
branch miss	0.01323053027
bus-cycles	0.00120860164
cpu cycles	0.0005182546332
instruction	0.0002708656424
stalled cycles FE	
stalled cycles BE	

Table 4. T-test results for game application.

The t-test results for this experiment indicate that there is indeed a significant statistical difference between the means of the number of hardware events for the two modes of input introduced to the chess application. This could be due to the more sophisticated animations when a chess piece is picked up and dragged across the board when swipe inputs occur.

Future Work:

If simpleperf developed a more advanced hardware event logging tool, with timing-logs, a more fine-grained analysis of the effect of user-input on micro-architecture can be performed. Using this we can also test more applications for which we cannot necessarily ensure the same state is achieved after every iteration of the input due to the applications being based on Random Number Generation(RNG).

We can also further our experiments by testing more applications. Some potential applications that could work within the bounds of our test setup include web page browsing(though they should be cached and the network should be turned off to minimize the impact of stray network packets), ebook reading, video player and more games.

Our experiments are primarily concerned with whether or not different user inputs would have an impact on the number of hardware events. While we were able to determine that there was a statistical difference for the applications we tested, these experiments do not show whether or not user input can be impacted by hardware events. We can imagine that a program that performs smoothly would induce the user to have different interactions with the device compared to a program that stutters often. This would be an interesting area to examine if well-designed experiments can be established.

Conclusion

By isolating different methods of input and comparing hardware performance counter readings over many repeated experiments, we have determined that the mode of user input can affect the frequency of hardware events. While the exact reasons for the differences can be difficult to pinpoint, the fact that they exist means that performance can be optimized based on a user's preferred method of input. Having devices adjust to a user's input patterns could prove to be beneficial in energy consumption gains.