Our progress report will focus on our evaluation of the stability of the OpenCL driver (this is our central hypothesis). In order to assess this, several experiments were conducted, each testing a different hypothesis (i.e. think of it as a series of science experiments).

In our case, an experiment is an OpenCL program running on the Nexus 4 Android platform.

For experiments 1., 2., 3., the OpenCL program (opencl_aes.c) used is mostly the same (only with small modifications). That is, its the OpenCL implementation of an AES encryption algorithm, where all the program does is take a fixed size block of zeroes in memory, then encrypt the zeros using AES as implemented on the GPU. It doesn't even decrypt it, so you are just checking its correctness based on whether consecutive runs of the program returns consistent results.

For experiment 4., I used a different program (helloworld.c) based off an existing prototypical helloworld OpenCL android example. This is probably the easiest first OpenCL program to understand (since I've at least documented what it does). Correct output for this program is easier to understand (it will explicitly say “ERROR” when it fails in the output).

In general, source files are found in the src/ directory of each experiment, and output files are found in data/. The .c files will have an accompanying .sh file, which is what was used to actually run the experiment (e.g. Run the program 100 times consecutively, run the program with 1000 forked instances).

The following experiments were conducted:

1. **Purpose:**
   Based on research by Jiawen into an OpenCL implementation of the AES encryption algorithm, we have been told there was instability in the OpenCL driver. We wish to determine whether this instability exists in a newer version (4.2.2) Android.

   Experiment:
   To do this, we begin by simply running the code 100 times consecutively. Next, we try running the test 100 times consecutively while simultaneously trying to interact with the device by turning on the display.

   Experiment directory:
   android4.2.2_opencl_stability

   Results:
   To access what “crashed”, we ran the command “top -n 1” to get a snapshot of the running processes. We compared the running processes on a freshly booted device (just_booted.top.txt) to the running processes post-failure (63_runs_with_crash.top.txt).

2. **Purpose:**
   See if the same strange OpenCL behaviour is present in Android version 4.3.

   Relavant experimental results:
   Summarized in Preliminary OpenCL Stability Tests (“So Jiawen gave me the code he had used to run a GPU AES encryption algorithm”).

Experiment directory:
android4.3_opencl_stability

The data from the experiment is just show that we are failing to load the /system/libOpenCL.so library, so we won't want to use it really. We should mention what we found out about RenderScript when I speak about this experiment.

Relavant email thread:
Preliminary OpenCL Stability Tests

3. **Purpose:**
   Based on the predictably bad behaviour of OpenCL when turning on the display, we hypothesize this behaviour is due to a lack of coordination between OpenGL calls from other processes using the GPU for graphics tasks, and our OpenCL AES program. This could be due to the AES program not using OpenCL APIs correctly; in particular, not checking error conditions return by OpenCL calls, or not making using of coordination mechanisms.

   Background:
   I independently researched what coordination mechanisms exist for OpenCL code (refer to the GPU TO DO document for sources I used). I wasn't able to find cross-process OpenCL/OpenGL coordination mechanisms (only within the same process), so I did not make use of this. Instead, I tried using exhaustive error checks and clGetDeviceInfo to check the availabilty of a GPU before I used it.

   Experiment directory:
   android4.2.2_mutually_exclusive_access_AES

4. **Purpose:**
   Since after exhaustive error checking to OpenCL API calls and checking GPU availbility has not solved the issue, we next hypothesize that there may be bugs in the AES codebase that we are unaware of. To check for this, we create minimal OpenCL test program from scratch (refer to the top of helloworld.c). We then run the minimal OpenCL program 100 times consecutively and 100 times when activating the display (as done with AES). We also try running 1000 forked instances of the program to determine if there is coordination amongst OpenCL kernels from different processes (i.e. as opposed to just 1 OpenCL kernel and OpenGL operations).

   Background:
   I researched a basic OpenCL tutorial that told me about all the various setup/teardown routines for OpenCL programming (so that I knew the helloworld starter code wasn't missing any relavant API calls that could cause undefined behaviour).

   Experiment directory:
   android4.2.2_mutually_exclusive_access_BASIC

   There's no email thread for this experiment; refer to GPU TO DO document starting at “Figuring out OpenCL programming and mutually exclusive access to the GPU”.