Raspberry PI GPU Auto-Overclocker

Development Project Progress Report – Team Ferrari

*Sajeeb Roy Chowdhury, Caleb Maranian

Colorado State University

Fall 2019

Project Objective:

The goal of this project is to develop a raspberry PI GPU auto overclocker. The objective of the PI GPU overclocker is to achieve higher overclock profiles than that can be attained by auto overclocking programs by GPU manufacturers such as MSI, EVGA and NVIDIA. As of the time of writing this progress report, there are only two auto overclocking programs: Precision X1 from EVGA and Afterburner from MSI. Both of these overclocking programs use a form of a lookup table to apply overclock profiles and stress test for stability. Since these programs use a lookup table, they can only apply discrete overclock profiles that are in the lookup table. This often times does not work very well for low end silicon chips, as the lookup table approach applies a lower profile than that can be achieved by manual overclocking. It is because of this limitation in the lookup table approach, we proposed to develop a sophisticated algorithm that resides on the raspberry PI that approximates and applies incremental overclock profiles to the host machine with a GPU. Our algorithm guarantees to attain a higher overclock profile than what can be achieved through vendor auto-overclocking programs with the only caveat being that it will take longer to reach the optimal overclock profile.

Current progress on required hardware:

In the original proposal, the required hardware was a raspberry pi kit to host the overclocking algorithm, and a desktop/laptop machine with a GPU to test the algorithm on. As of the time of writing this progress, we have acquired the raspberry PI 4. We do not need to purchase a desktop/laptop with a GPU, since we already have a laptop with a GTX 1060 GPU which should suffice for the project.

Figure 1: Proof of acquired raspberry PI

Current progress on software and setup:

As of the time of writing this progress report, we have acquired the raspberry PI and have installed Raspbian on it along with the GreenWithEnvy (GWE) software we initially proposed to use for setting core and memory overclock profiles. Due to certain complications related to the software package and setting NVIDIA coolbits on the host system, we are unable to use this software, and had to resort to writing our own library to abstract the calls to set core clock, memory clock and power limit offset.

As of the time of writing this progress report we were also able to setup the host system with the correct X-server configuration and NVIDIA coolbits to allow our own library to properly set the required offsets necessary for overclocking.

Current progress on the development of necessary algorithms:

As of the time of writing this proposal, we were able to develop an outline of the GPU core clock overclock algorithm. In the initial proposal, we proposed to develop three sets of algorithm to attain a complete overclock. However, the other two algorithms are similar to the core clock overclock algorithm; in that they both incrementally apply OC profiles and evaluate the profiles by stress testing the system with the profile. It is due to this level of similarity, in this progress report we only give an overview of the GPU core clock overclock algorithm.

The memory clock overclock algorithm is exactly like the core clock algorithm, in that each OC profile increments the memory clock by a set increment and stress tests the OC profile. If the profiles caused the system to crash or overheat, a reduced increment is applied to the last known good stable increment to create a new OC profile, if this OC profile also fails, the reduced increment is further reduced. This iterative process continues till the algorithm converges to a stable OC profile.

The core undervolting algorithm is similar to the core overclock algorithm, but this time the core voltage is reduced by 0.5 volts every time and stress tested. If a reduction causes instability in the system, the reduction amount is reduced by half and applied to the voltage value of the last known good OC.

GPU Core Clock Overclock algorithm overview:

Our OC algorithm for GPU core overclock is divided into two parts: the part that resides on the host system and the part that is on the raspberry PI. The part of the algorithm that is on the raspberry PI is the core of the algorithm, i.e. this is the part of the algorithm that calculates the changes to the OC profile and sends the profile to the program in the host system over Ethernet to be stress tested.

The process on the host machine, after receiving and applying the OC profile, will fork the stress process and the temperature monitor process, and let them run for a certain period of time that is determined by how far the core clock in the OC profile is to the base clock of the GPU.

The job of the stress process is to constantly bombard the GPU with render jobs in a loop. The stress process will not be making use of any form of inter process communication protocols to send data to the parent process. The only information the parent process gets to monitor about the stress process is whether or not it has crashed.

The temperature process will constantly monitor the temperature of the GPU over periods of 10 second intervals, take the average of those and return a status to the parent whether or not to kill the stress process. In general if the average temperature at any 10 second interval goes above the set threshold, the temperature monitor process sends a kill signal to the parent to kill the stress child process.

As mentioned earlier, the host process will allow the two child processes to run for a certain period of time, generally this period of time increases as the deviation from the base clock increases. We have decided to keep the increase in the runtime proportional to the factor increase in core clock, for example if the factor increase in core clock is 2, then the increase in runtime would be a factor of 2. To keep things simple, we have decided to start with an initial runtime of 15 minutes for the first increment in core clock speed.

If during the specified runtime period, the host parent process does not receive a kill signal from the temperature process or the stress process has not died, then the parent process sends an 'OK' status to the PI. The program running on the PI then increases the overclock profile, and stresses the system with this new overclock profile.

If the parent process receives a kill signal from the temperature process or the stress process has died, the parent then sends a failure status to the PI along with the failure reason and waits on a new adjusted OC profile from the PI. Once the PI has received this signal, it will discard the current OC profile and generate a new OC profile from the last known good OC profile by incrementing the core clock offset by half of what is was in the OC profile that failed. This new OC profile is then sent to the host system to be stress tested like before.

During the time the program on the host is running the stress and temperature process, it also sends an alive signal to the raspberry PI at every 1 second interval along with the temperature data it collects every 10 second interval. This is done over the Ethernet connection to the PI to let the PI know that the host system has not crashed. If the host system has crashed, then there will be a 1 second interval where the PI did not receive an alive signal from the host - this is how the PI knows that the system crashed.

Once the PI is aware that the system has crashed, it will look at the series of data immediately before the crash. If the data before the crash shows a temperature that is greater than the threshold, it will adjust the OC profile to reduce the power offset by 0.1 volts. If the crash was not caused due to a temperature issue, it will decrement the current core clock by half the increment value.

Once the host system restarts after the crash and the communication program to the PI is loaded, this new adjusted OC profile is sent to the host to be stress tested. This process of adjusting the OC profile continues till the increment in core clock from the last known good OC profile is less than 1MHz.

Evaluation:

We will evaluate our finished project against the following two criterions: power consumption measurements of the overclocked GPU with our algorithm to vendor auto-overclocking algorithms, and accuracy and response time of the overclocked GPU with our algorithm to vendor auto-overclockers.

For the power consumption measurements, we will particularly be looking at the ratio of the summation of core clock and memory clock to power consumption. This is because if our algorithm is able to attain a higher overclock with very little power, then its ratio of summation of core clock and memory clock to power consumption will be high compared to other vendor algorithms that only overclock the core clock, or algorithms that overclock both the core clock and the memory clock but are unable to get these clocks high enough while keeping a low power consumption profile to get a high overclock to power consumption score. In order to gather data for this measurement we will be overclocking two generations of GPUs - GTX1060 and GTX 750Ti - to a stable overclock using both our algorithm and vendor algorithms. Afterwards we will compare the overall overclock to power consumption ratio by running stress tests using a power virus such as FurMark. The stress test will give us the max power consumed by the GPU. We gather these values for both GPUs using the two classes of algorithms, the algorithm that produces the highest ratio is the better algorithm for overclocking in terms of raw performance to power consumption ratio.

In order to assess the accuracy and response time of the overclocked GPU using our algorithm to vendor algorithms we have decided to train a fully connected feed-forward neural network for classification of hand written digits – the dataset for this will be acquired from MNIST. We will first train the neural-network on a non-overclocked GPU, overclock the GPU using the proposed algorithm, and then finally overclock using vendor algorithm. At each of these stages we will train the same neural network with the same initial configuration of starting weights. After training on the non-overclocked GPU for a certain number of iterations, we will store the final configuration of weights in the trained model as a reference point. This is because; the non-overclocked GPU is the most stable and is guaranteed to give the same result given the same initial starting point. We will then train the neural-network with the same initial configuration of weights on GPUs that have been overclocked with our algorithm and on same GPUs that have been overclocked with vendor algorithms for the same set number of iterations.

Once training the model is completed, we will compare the final configuration of weights in the trained model from the overclocked GPUs to the weights from the reference model from the non-overclocked GPU. We access the accuracy of the trained model from the overclocked GPUs by subtracting the weight matrices of the overclocked model and the reference model and taking the 2-norm of the matrix. If this value of 2-norm is less than or equal to 2^{-16}, then the model can be considered accurate and the overclock is not breaking anything crucial. In fact, the lower the 2-norm, the better the overclocking algorithm is at preserving computational accuracy with respect to the non-overclocked model with the non-overclocked model having the highest accuracy.

We can assess the response time by calculating how long it took the overclocked versions of the GPU to finish training the model. The overclocking algorithm that finished training the model earlier is the better one. In general, we can compute the inverse of accuracy multiplied by inverse of computational time (in seconds) for each overclocking algorithm with the same training model. The higher this ratio, the better the algorithm is at overclocking. This approach gives us a single numerical value to compare overclocking algorithms with respect to accuracy and computational time.

Future Work:

As of the time of writing this progress report, we were able to complete the crucial parts of the project necessary to progress further. These crucial parts include: setting up the host system to allow for the GPU to be overclocked by setting the coolbits in the X-server configuration files and changing permissions to configuration files across the system to prevent the GPU-Manager program under Linux Xorg to overwrite these changes at system reboot or crash, setting up the raspberry PI with Raspbian OS, writing the overclock library to be able to apply overclock profiles such as setting the memory offset, core clock offset and power offset. We still need to finish the following tasks: write a library that allows both the raspberry PI and the host machine to exchange information, implement the core clock, memory clock and undervolting algorithm on the PI, write the program on the host machine that reads the OC profile sent from the PI and uses the already written library to apply the overclocks and do stress testing and sending alive signals and temperature data to the PI.