# Project

## Description

You are required to design and implement a project that leverages GPU acceleration through CUDA. It's highly recommended you unleash your creativity to pick an innovative idea that best suits this programming model and resources. Bear in mind that this project is majorly concerned with providing some reasonable and quantifiable results of your work generally, in addition to comprehensive performance analysis and comparisons with other open-source peers. You will first submit an idea proposal then go straight to plan and design, then once you get an approval, you will pursue your work as per our alignment. Also note that your work is expected to last for 4 weeks, so please plan-ahead and try to fit all in this short duration.

## Purpose

The ultimate goals of this assignment can be summed up as follows:

1. Think parallel, i.e. tweak a traditionally sequential algorithm so that it can run in a parallelized manner and get the ultimate benefit of GPU acceleration.
2. Practice cuda and gain more hands-on coding experiences in more general problems.
3. Assess your work quality by profiling, performance analysis and benchmarking.

## Main Components

- **Proposal**: this should be as simple as a 1-page submission, containing your team info, idea description in a short paragraph, and a simplified block diagram clarifying the workflow through the different kernel modules, in addition to an expected workload division among the team members.

- **CPU Demo:** here you should do your idea's implementation based on CPU only. This should act as a baseline that helps in performance analysis (as described below) and as an initial step towards moving forward to GPU. (You can start with a ready-made version and improve it till it contains the **full** features as the expected GPU one)

- **GPU Implementation**: this is meant with the basic implementation of your cuda kernels and the full program that consumes them, you can either implement all in C, or call your C kernels as bindings in python.

- **Streaming**: as long as we aspire to the maximum allowable throughput, you need to make use of streaming (more on this in the next lectures)

- **Performance Analysis**: this could be the most critical part, you should answer these questions after doing comprehensive experiments against reasonable data sizes:
    - What are the CPU benchmarks for your application (theoretical ones)
    - What about their GPU counterparts (based on your implementation)
    - How much is the speedup of the GPU over the CPU?
    - How does this compare to the theoretical speedup? (try to carefully do your research to answer this)
    - If your speedup is below the theoretical one (this is mostly the case), how do you explain this, what could be changed to achieve a better one?
    - How do your GPU results compare to open-source peers of the same features, if any? (this could be direct APIs used through a framework such as Pytorch, Keras, … etc).

  **N.B**. You are responsible for determining the test data sizes and input patterns that best leverage the parallelism at your code (as comparably small data may not show insightful results)

- **Final Report:** containing:
    - Your final system description (the final form of the proposal)
    - Experiments and results, driven the way you think is the most appropriate
    - Performance analysis as described above.

## Sample Ideas

This section is to be appended with general sample ideas throughout the next couple of weeks. Check it frequently.

- Image processing: Convex Hull
- Video encoding/decoding (H264, … etc)
- Graph processing system (supporting multiple algorithms)
- Parallelizing Linux shell utilities (grep, sort, uniq, … etc)
- Database management system with various operations (joins, select, update, … etc)
- Very High Speed Big Data Analytics

## Rules

- **Team formation**: a team consists of 2 - 4 students (based on your project workload but a kernel per person is expected)
- **Project ideas**: more than a team can work on the same idea but algorithms should be different from algorithms taken in class.

## Deliverables and Timeline

- **Proposal:** Thursday, 13th March
- **Phase 1:** Sunday, 20th April should contains:
    - **CPU Implementation**
    - **At least one kernel**
- **Final Delivery:**
    - **Source Code**: Saturday, 10th May, midnight. Week 14. All of your code files organised in a neat way, along with readme files explaining how to run and reproduce your results.
    - **Final Demo**: Sunday, 11th May. Week 14. A live in-person presentation of 10 ~ 12 mins. MAX. in which you will walk through a final presentation (demonstrating the elements in your report) and show us a quick demo, where you run your program on reasonable test data and show us your results. Note that we'll tolerate extended run times if your application scale is quite large.