Semester Project

Description

You are required to design and implement a customized Database Management System (DBMS) that leverages GPU acceleration through CUDA. Being a competition-based project, it's highly recommended you unleash your creativity to optimise the time and memory requirements for handling different queries according to this programming model and resources. Also, bear in mind that this project is majorly concerned with providing some reasonable and quantifiable results of your work generally, in addition to a comprehensive performance analysis and comparisons with other open-source peers.

Purpose

The goals of this project 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. Practise cuda and gain more hands-on coding experience in more general problems.
3. Assess your work quality by profiling, performance analysis and benchmarking.

System Description

- **Data Representation & Storage:** Data is supplied as a bunch of CSV files with clear headers and hundreds of thousands of data records. Primary keys are followed by **(P)** tag in the header. If the primary key is referenced in another table, the foreign key header name is exactly the same as the primary key, with a naming convention of #PrimaryTableName_id. Data is stored on a persistent auxiliary space, e.g. hard disk.

- **Operations:** Select, project, join (using where clause only), filtering on conditions (general where clause), order by (numerical or string column, ascending (Asc) or descending (Desc)) aggregates (count, max, min, sum, avg), relational operators (<, >, =, !=), logical operators (and, or), or cascaded operations of these ones.

- **Queries**: Queries are submitted in a SQL-like fashion covering the mentioned operations only. Ex:-
    - select s.name, a.address, count(*) as total_count from Students s, Addresses a where s.student_id = a.student_id and s.year > 3 order by s.name Asc;
    - Joins are not limited to two tables, you have to handle multiple joins.
    - A query can include a Max of 1 subquery. Ex:
        - select name, address, count(*) as total_count from (select s.name, a.address from Students s, Addresses a where s.student_id = a.student_id and s.year > 3) order by name Asc;

- **Interface:** You need to develop a CLI interface that parses such queries. If the result is tabular, output a csv file to the working directory only, otherwise, output the result to the interface and in a .txt file as well. (More details on the delivery rules are provided later)

Main Components

- **Design**:You will submit a design document describing your design for the intended approach for each of the basic operations. (chosen algorithm, optimisations, kernels high-level description) alongwith the integrated system and data flow.

- **CPU Demo:** here you should do the 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 full implementation of your cuda kernels and the full program that consumes them, you can either implement all in C or link your C kernels as libraries in python

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

- **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 queries processing a range of different large numbers of records)
    - 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?

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

- **Final Report:** containing:
    - Your final design document
    - Experiments and results, driven the way you think is the most appropriate
    - Performance analysis as described above.

Grading Criteria:

- Fully-functioning GPU implementation passing the automated test cases, CPU implementation, applying streams (40%)
- Full understanding of the system modules and kernels, Report (Design & Performance Analysis) (40%)
- Ranking competition based on average query latency (20%)

Rules

- **Team formation**: a team consists of 3 - 4 students
- **Completeness:** all operations MUST be implemented and fully functioning on GPU.
- **Data Manipulation**: you are NOT allowed to do any data manipulations before query time. Any data handling should be considered within query time for fair comparison.

Deliverables and Timeline

- **Source Code**: Thursday, 1st May, midnight. All of your code files organised in a neat way, along with readme files explaining how to run and reproduce your results. Here, you'll follow all delivery rules (provided later)
- **Final Demo & Report**: Sunday, 4th May. A presentation of 10 ~ 12 mins. MAX. in which you will walk through the final findings and results and run a quick demo.