CIS6930 Fall 2017, Project 4

This is an open project to be finished by groups of up to **three** students. You can propose your own topic or choose one from the topics I recommend (see the bottom of this page for details). These topics listed also give you some examples of the magnitude and flavor of the projects I wish to see. Basically, you are expected to implement a non-trivial data structure or algorithm (either as a standalone program or one integrated into a database management system), or study an area of database research and complete a comprehensive review. For example, you can implement the quad tree in PostgreSQL. Application development on top of a DBMS (e.g., building an online bookstore with a database backend) is not encouraged unless it carries substantial academic prestige.

**Project requirements**

1. Your code should be clearly documented and commented. Code without detailed comments are generally regarded as of little value to people who takes over the project or simply wants to understand your design. You are required to submit all the relevant code and accompanying documents.

2. I expect you to write a report in the format of a publishable paper. In the report, you should clearly state the objective of your project, your design of the data structure or algorithm, and relevant testcases by which the grader can evaluate your code.

3. Your project will be graded according to the functionality you implemented and the details and format of your report. A software demo conducted in the instructor’ office is expected during the last week of class. The report and code should be submitted via Canvas by 11:30pm, December 1, 2017 (Friday).

This is a group project. Once you form a group, please choose a topic to work on and inform the instructor of the topic. I also recommend you choose one person as the team leader to represent your group in communicating with the instructor and coordinate your activities inside the group.

The following are a few sample topics that you can choose from. The following descriptions are very concise; please schedule an appointment with the instructor for more details.

1. Aggregates in PostgreSQL. Supporting complex aggregates (other than the MIN, MAX, COUNT, AVG, SUM kind of things as part of the SQL standard) has always been a functionality of modern DBMSs. The idea is to allow users to write a function of their own in a language of their choice and integrate the code into the PostgreSQL codebase. As a result, users can call the function directly in the SQL statements. In project 2, you wrote such functions in pgplsql. In this project, you will do that in a different language (i.e., C++) in writing a complex function to compute the spatial distance histogram of a set of 3D points. We have the code that handles 2D points, your job is to extend that piece of code to 3D and figure out a way to call the function in SQL. Estimated workload: about 1,000 lines of code to read (but such code is highly modularized and will be much easier to understand than your buffer manager code) and about 300 lines to write.
2. Parallel versions of join algorithm. Join is the most expensive and popular database operator. Due to its high demand on computing resources, parallelization of join algorithms obviously has great potential. This project will focus on implementation and optimization of a join algorithm on a multi-core Intel CPU. You may start from the existing code provided by another group and explore certain ways to optimize it by using the new hardware features of the most recent Intel CPUs. You can even create some locality to explicitly utilize the un-programmable L2 and L3 cache. This project requires basic knowledge in multithread programming.
3. Studying the storage manager of PostgreSQL. This project aims at an anatomy of the storage manager of the PostgreSQL system. Note that the storage manager is the module that takes care of the physical design of your database (e.g., how to organize data in pages, files ...) and it obviously interacts with the hardware layer to accomplish its tasks. Your job is to draw a layout of the design of PostgreSQL storage manager by listing its components and their functionalities. Based on such understanding, you are also asked to manipulate the mechanisms implemented in the storage manager to achieve certain goals. An example of such manipulation is: can we reorganize the tuples in a file at runtime? This project involves a lot of code reading and moderate code writing.
4. Data Stream management. Data streams are data generated continuously from a data source such as a sensor, the stock market. Dealing with such data requires DBMS architectures that are fundamentally different from that of the relational DBMS we saw in class. Existing stream data processing platforms based on Hadoop are available. Your job is to implement, optimize, and test a series of algorithms in processing scientific data generated in molecular simulations on the SPARK streaming platform. A similar project can be developed by using another platform such as the VoltDB system.
5. Streaming platform for traditional database workloads. The aforementioned streaming system can also be used to handle traditional database workload. The idea is to push the data to a compute node using one single streaming thread, and all queries wait for the same data stream to perform the computation towards query results. In this project, you will study the TPC-H benchmark and transform the queries and data in this benchmark into a streaming setup, and evaluate the performance of the new system.
6. Implementation of a **density map** based on quad-tree in PostgreSQL. A density map is a data structure that divides the data space into regions and records some aggregates (e.g., COUNT) of all data points in these regions. Its implementation can be accomplished by augmenting a region point quad tree with such aggregates. In this project, you are provided with some code (e.g., GIST) to implement the quad tree within PostgreSQL, and you will have to augment the regular quad-tree to enable density map functionalities. You will get help from a graduate student who implemented a part of this project in previous semesters. Estimation of the workload: moderate amount of code to read/understand, 400+ lines of code to write.
7. Suggested literature survey topics: processing joins on parallel platforms (e.g., shared everything, share-nothing, FPGAs, Hadoop, Spark); XML database systems; NoSQL systems.
8. **Parallel hash join in Data Streams**. See the following pages of this document for details. This project will be led by Shen Lu (shenlu@mail.usf.edu), please talk to her if you want to join the team.
9. **Development of a database and web server to predict possible drug-drug interactions**. The objective of the research project is to construct a web server to predict/present possible drug-drug interactions using the CYP450-Drug interaction data derived from drugbank. The input of the web server are two drugs. The output is the possible DDIs between these two drugs and related enzyme. For example, if drug A is a CYP3A4 substrate, and drug B is a CYP3A4 inhibitor, drugs A and B may have potential DDI. There is a lot of research literature and information about drugs that are CYP450 substrate, inducer and inhibitor. Information about how a specific drug is interacting with certain CYP450 enzyme can be collected from databases such as DrugBank (<https://www.drugbank.ca/drugs/>) and Supercyp (<http://bioinformatics.charite.de/supercyp/)>. These data will be incorporated into the database/server to predict possible drug-drug interactions.

**Project Title: Parallel Hash Join in Data Streams**

**Background:**

A data stream is an unbounded sequence of stream tuples. Stream join, also called *window join* in the literature, is a class of join algorithms for joining infinite data streams. Window join addresses the infinite nature of the data streams by joining stream data tuples that lie within a *sliding window* and that match a certain join condition. Hence, each tuple in a sliding window should be compared with all tuples existing in the opposite sliding window of the other data stream for the join predicate (see Figure 1). Since the typical rate of such comparisons is significantly large in data stream applications, there is a strong demand to parallelize the stream join on GPUs in order to improve the join operation throughput.

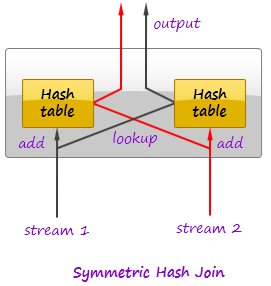
In this project, we focus on parallel computing of *hash-based* stream join (a more efficient algorithm than the nested loop join) on GPUs.

**Project Description:**

Symmetric hash joins (SHJ) is a special type of hash join well-suited for stream join scenarios since it supports the sliding window semantics by its incremental nature. SHJ for two input streams is briefly shown as following (see Figure 2)

* For each input stream create a hash table based on its join attribute
* For each new tuple arrived in an input stream
  + hash and insert into the stream hash table
  + Probe the opposite hash table for finding the matching tuples
* Output the join results

In this project, we would like to design the parallel SHJ algorithm and implement it on GPU architecture.



**Figure 2. Symmetric Hash Join (SHJ)**

**Description: A description...**

**Figure 1. Stream (window) join**