SUMMARY OF PAPER ON

COMPILING DATABASE QUERIES INTO MACHINE CODE

(Thomas Neumann, Viktor leis)

The paper starts with the introduction of the earlier model for processing queries. It explains how the initial **Iterator model** functioned. The way how it works for only stream of tuples also how it gets overhead when there are occasion where multiple tuples are needed as an output. To solve this issue we can either compile the queries directly to machine code but, this cannot be done easily as the algebraic expressions are very complex to convert. So they introduce the data centric compilation process. The databases system which they use for this purpose is called **HyPer**. Hyper converts the algebraic expressions into machine code using the **Low Level Virtual Machine** (**LLVM**) compiler.

Compilation of the queries are usually driven by algebraic operators where the code is set based on the operator and for each tuple the control passes from one operator to another. Whereas in **HyPer** they don't pass the data between operators rather they hold the attributes in CPU registers as long as possible. They break the query into pipeline and load them with tuples which are then passed to operator to work on them. The source operator access the memory to load the tuples whereas the intermediate operators does its functions but they won't write to the memory. Thus this method make sure that the memory is accessed as much less as possible.

HyPer parses the query then convert them into algebraic expression and then optimizes them. There is no need of fundamental changes since the HyPer uses traditional cost based query optimization. Rather than converting the algebra expression to physical, it compiles them into imperative program. The pipeline is then compiled to code fragment for each before execution. The attributes in each fragment is kept in registers so they don't need to be explicitly passed to the operators. Here instead of pulling tuples from the input operators the tuples are pushed towards the consuming operators. This avoids he unnecessary materialization of tuples and memory traffic.

The unified interface present in this operator runs on two functions *produce* and *consume.* Produce function produces result tuples for the next step and the consume functions consumes the incoming tuples from the child operator. Basically the produce and consume are codes that produce other codes (pseudo-codes). Writing machine code is taken care of by **LLVM** or C--. Abstraction layers are helpful for mapping SQL queries to machine code. LLVM is the low level assembly language that offers well defined interface.

The data types are referenced by 1-bit or 32-bit integer. The 1 bit is for whether it is null or not and 32 bit is for mentioning the actual value. If the multiple values are stored as a structure then it will lead to performance minimization as it needs pointer for traversing through records. The control flow is taken care of by LLVM by providing only conditional and unconditional branches. It uses SSA (static single assignment) where a register can be used only once. The high level IF and FOR helps in constructing the control flow easily. Since there are no tuples in the runtime code the operators directly act on registers. The storage methods are **Compact storage** and **updateable storage.** The former stores the data compactly whereas the latter gives maximum memory for each attribute so they can be edited later. The current method is compared with the iterator model and block-wise processing by executing the same query for all three where the data centric approach overcomes the rest.

Vignesh Kumar Karthikeyan Rajalakshmi
(A20424508)