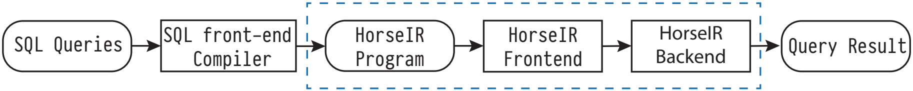
HorseIR : An Array-based Approach to SQL Queries

Hongji Chen

August 30, 2017

What is HorseIR?

- array-based programming language
- uniform intermediate representation(IR)
- optimization framework



What is HorseIR?

In-memory Database System and IR-based Query Engine

- row-oriented or column-oriented systems
  - row-oriented storage: run as few operation as possible on row record (most database use-case, e.g. logging)
  - column-oriented storage: good for set operations for whole table data (e.g. analytics)
- complex primitives set or reduced primitives set
- user-defined functions (UDF)
  - not in SQL standard, but as a language extension in most systems
  - flexibility?
  - optimize potential?
- parallel code generation

Related Work - In-memory Database Systems

- SQLite
  - https://sqlite.org/opcode.html
- MonetDB
  - Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten. 2012. MonetDB: Two Decades of Research in Column-oriented Database Architectures. IEEE Data Eng. Bull. 35, 1 (2012), 40–45.
  - http://sites.computer.org/debull/A12mar/monetdb.pdf
- KDB+ System
  - https://kx.com

SQLite

- SQLite Bytecode
  - scalar based IR design
  - dynamically typed
  - no optimization on bytecode level
  - designed for register based virtual machine

MonetDB

- pioneer in IR based execution engine design
- MonetDB Assembly language(MAL)
  - scalar based IR design
  - statically strongly typed, without sub-typing
  - encapsulate atomic relational algebra operation in each instruction
  - frontend translate queries into MAL
  - backend interpret MAL to generate result
  - optimizer optimize MAL to MAL

KDB+ System

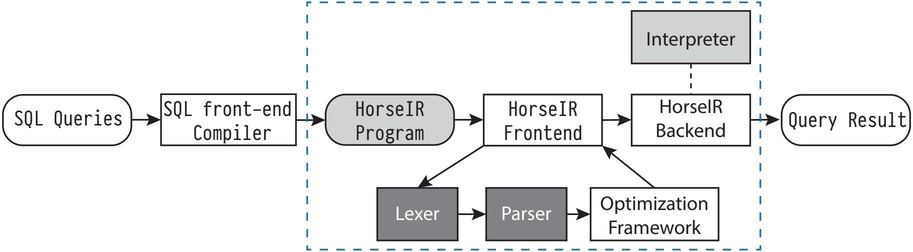
- K and Q programming language
  - fully functional programming (not IR-based)
  - array-based language design (array/vector are first-class type)
  - focus on data analytics
  - powerful and efficient primitives implementation
  - limited intra-procedural or inter-procedural optimization

Related Work Summary

Database System	SQLite	MonetDB	KDB+ System
Language Design	scalar-based	scalar-based	array-based
Storage Structure	row-oriented	column-oriented	column-oriented
Primitives	reduced	complex	reduced
UDF Support	external	MAL, external	native
Parallel Code	limited	explicit declared	implicit (primitives)

HorseIR - Summary

- syntax design
- type system
- fully functional interpreter connected to Hanfeng's backend



HorseIR

- array-based IR design for database system
    - reflects column-data storage in database system
        - homogeneous data type storage
        - vector-based column
    - explore data parallelism easily
        - a rich set of array-based built-in functions
    - data compression friendly
- statically typed, with sub-typing

HorseIR - A Quick Example

```
SELECT LastName FROM EmployeeTable;
```

```
module default {
    import Builtin.*;
    def main() :table {
        t0    :table       = @load_table(‘EmployeeTable) ;
        name  :sym         = check_cast(
            @column_value(t0, ‘LastName),
            sym
        ) ;
        t1    :list<sym> = @enlist(‘LastName) ;
        t2    :list<sym> = @enlist(name) ;
        t3    :table       = @table(t1, t2) ;
        return t3 ;
    }
}
```

HorseIR

- column-oriented system (use vector as a abstract view of column)
- reduced primitive set
  - efficient implementation
  - expose details to optimizer
- UDF implementation:
  - write UDF in arbitrary language
  - compile into HorseIR
  - participate in optimization
- parallel code generation:
  - at primitive level
  - at intra-procedural level
  - at inter-procedural level

HorseIR

```
CREATE TABLE Employee
(LastName varchar(99), DepartmentID int);
CREATE TABLE Department
(DepartmentID int, DepartmentName varchar(99));

select * from Employee, Department
where Employee.DepartmentID = Department.DepartmentID;
```

Types in HorseIR

- designed for database systems
  - string and character
  - numerics (integer and floating-point)
  - date and time
  - functions
  - table and keyed table
- parametric polymorphism
  - list (list<?>)
  - dictionary (dict<?, ?>)
  - enumeration (enum<?>)

Types in HorseIR

- statically checked and inferred at compile time
- resolve overloading and polymorphic dispatch at compile time
    - minimize runtime overhead
- downcast guard elimination

```
x  :?
castedX  :i32 = check_cast(x, i32); (*)
avgX  :i32 = @Builtin.avg(castedX);
```

Is it possible to safely remove this cast? use static analysis

Types in HorseIR - Overloading

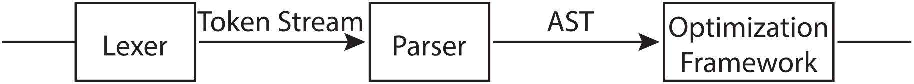
- support more flexible UDF
- choose the most efficient primitive implementation at compile time
    - function overloading

```
def foo(x: ?, y :?) :bool {
    avgX :? = @Builtin.avg(x) ;
    result :bool = @Builtin.lt(y, avgX);
    return result;
}
@Builtin.avg(?) @Builtin.avg(f64) @Builtin.avg(i32)
```

- choose which implementation?
    - in foo(i32, i32),
    - in foo(f64, i32),
    - in foo(list<i32>, i32)

HorseIR Frontend

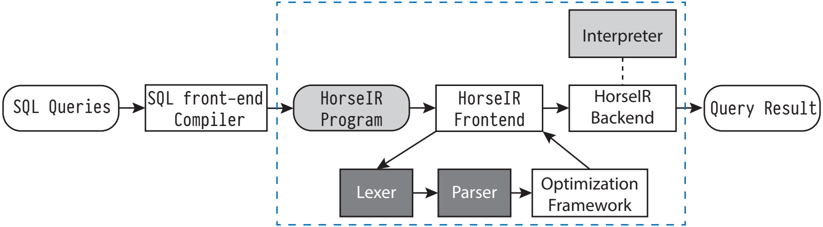
- a lexer and parser
- interpreter
- customized abstract syntax tree
- optimization framework
  - peephole
  - static analysis



Lexer → Token Stream → Parser → AST → Optimization Framework

Conclusion

- HorseIR
  - array-based IR for database system with powerful primitives
  - programming language optimizations for databases
  - extensible framework for user-defined functions (UDF)
  - efficient parallel code generation



- complete working system
- future experiment on TPC-H benchmarks

Questions?