

Lecture #10

Carnegie Mellon University
ADVANCED DATABASE SYSTEMS

Vectorization vs. Compilation

Andy Pavlo // 15-721 // Spring 2023

ADMINISTRIVIA

Project #1: Sunday February 26th

Project #2: Sunday April 30th

Project #3

- Proposals: Wednesday March 1st
- Updates: Monday April 3rd
- Final Presentations: TBA

OBSERVATION

Vectorization can speed up query performance.

Compilation can speed up query performance.

We have not discussed which approach is better and under what conditions.

Switching an existing DBMS is difficult, so one must make this design decision early.

VECTORWISE – PRECOMPILED PRIMITIVES

Pre-compiles thousands of "primitives" that perform basic operations on typed data.

→ Using simple kernels for each primitive means that they are easier to vectorize.

The DBMS then executes a query plan that invokes these primitives at runtime.

→ Function calls are amortized over multiple tuples.

→ The output of a primitive are the offsets of tuples that satisfy the predicate that the primitive represents.

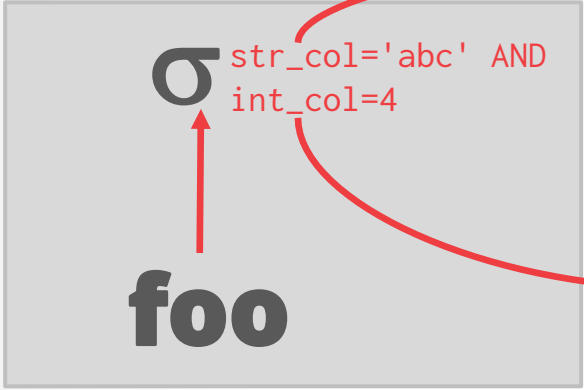


VECTORWISE – PRECOMPILED PRIMITIVES

```
SELECT * FROM foo
WHERE str_col = 'abc'
AND int_col = 4;
```

σ `str_col='abc' AND
int_col=4`

foo



```
vec<offset> sel_eq_str(vec<string> batch, string val) {
    vec<offset> res;
    for (offset i = 0; i < batch.size(); i++)
        if (batch[i] == val) res.append(i);
    return (res);
}
```

```
vec<offset> sel_eq_int(vec<int> batch, int val,
                      vec<offset> positions) {
    vec<offset> res;
    for (offset i : positions)
        if (batch[i] == val) res.append(i);
    return (res);
}
```

HYPER – HOLISTIC QUERY COMPILATION

Compile queries in-memory into native code using the LLVM toolkit.

Organizes query processing in a way to keep a tuple in CPU registers for as long as possible.

- Bottom-to-top / push-based query processing model.
- Not vectorizable (as originally described).

HYPER – HOLISTIC QUERY COMPILATION

```
SELECT * FROM foo
WHERE str_col = 'abc'
AND int_col = 4;
```

σ `str_col='abc' AND
int_col=4`

foo

```
vec<offset> scan_row(vec<string> str_col, string val0,  
                    vec<int> int_col, int val1) {  
    vec<offset> res;  
    for (offset i = 0; i < str_col.size(); i++)  
        if (str_col[i] == val0 && int_col[i] == val1)  
            res.append(i);  
    return (res);  
}
```

TODAY'S AGENDA

Vectorization vs. Compilation

Project #2

Project #3

VECTORIZATION VS. COMPILATION

Test-bed system to analyze the trade-offs between vectorized execution and query compilation.

Implemented high-level algorithms the same in each system but varied the implementation details based on system architecture.

→ Example: Hash join algorithm is the same, but the systems use different hash functions (Murmur2 vs. CRC32×2)

IMPLEMENTATIONS

Approach #1: Tectorwise

- Break operations into pre-compiled primitives.
- Must materialize the output of primitives at each step.

Approach #2: Typer

- Push-based processing model with JIT compilation.
- Process a single tuple up entire pipeline without materializing the intermediate results.

TPC-H WORKLOAD

- Q1:** Fixed-point arithmetic, 4-group aggregation
- Q6:** Selective filters. Computationally inexpensive.
- Q3:** Join (build: 147k tuples / probe: 3.2m tuples)
- Q9:** Join (build: 320k tuples / probe: 1.5M tuples)
- Q18:** High-cardinality aggregation (1.5m groups)



TPC-H ANALYZED: HIDDEN MESSAGES AND LESSONS LEARNED
FROM AN INFLUENTIAL BENCHMARK
TPCTC 2013

TPC

Q1: Fixed-point ar

Q6: Selective filter

Q3: Join (build: 14

Q9: Join (build: 32

Q18: High-cardin

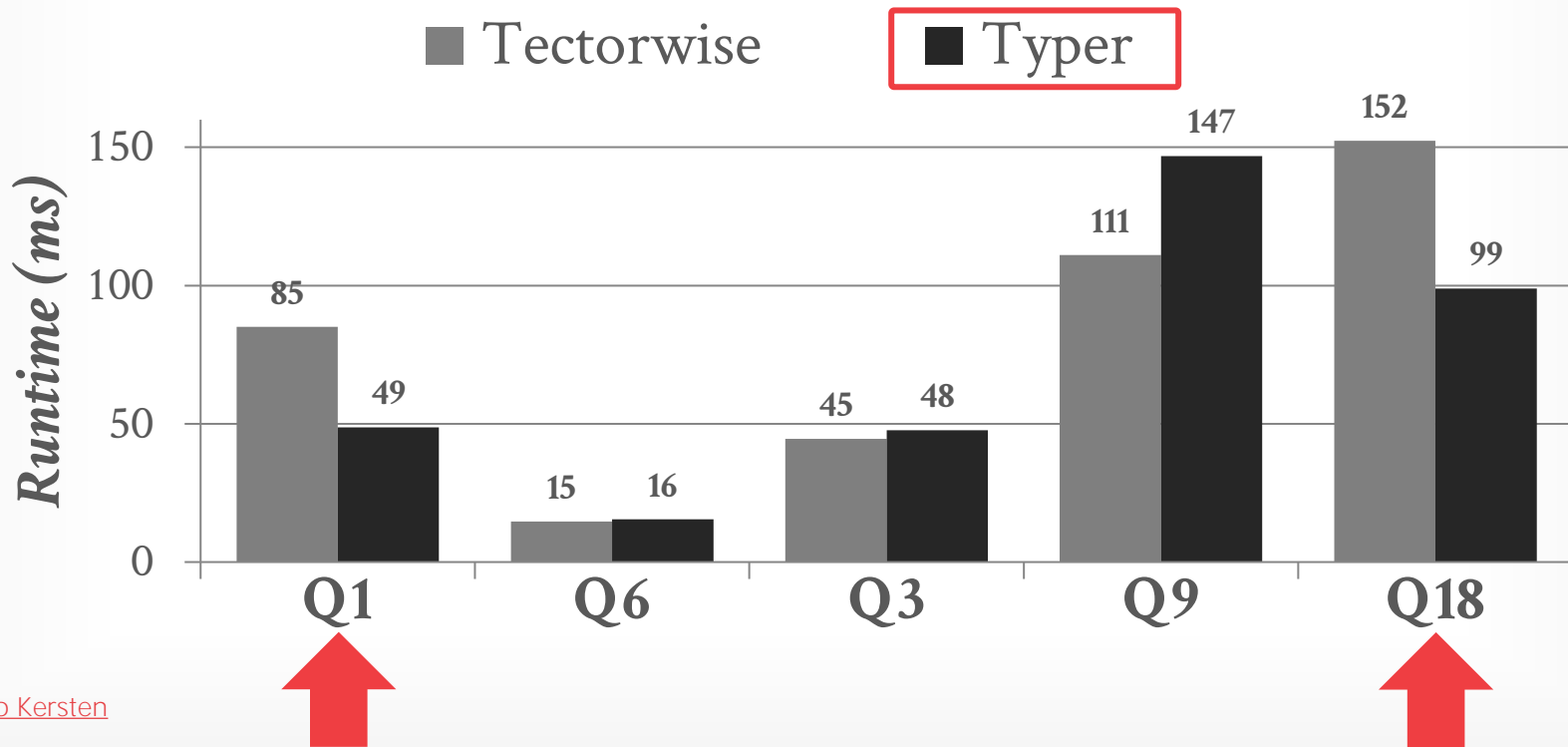
The screenshot shows the GitHub repository for `cmu-db / benchbase`. The repository is public and has 98 forks and 230 stars. The current view is the `main` branch, showing the file structure: `benchbase / src / main / java / com / oltpbenchmark / benchmarks / tpch / procedures /`. A commit by TheoVanderkooy titled "Fix bugs with TPCH implementation (#204)" is highlighted, dated July 7, 2022. Below the commit, a list of files is shown, each with a link to the commit and a timestamp of "7 months ago".

File	Commit	Timestamp
GenericQuery.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q1.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q10.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q11.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q12.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q13.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q14.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q15.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q16.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q17.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q18.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q19.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q2.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q20.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q21.java	Fix bugs with TPCH implementation (#204)	7 months ago
Q22.java	Fix bugs with TPCH implementation (#204)	7 months ago

TPC-H ANALYZED: HIDDEN MESSAGES AND LESSONS LEARNED
FROM AN INFLUENTIAL BENCHMARK
TPCTC 2013

SINGLE-THREADED PERFORMANCE

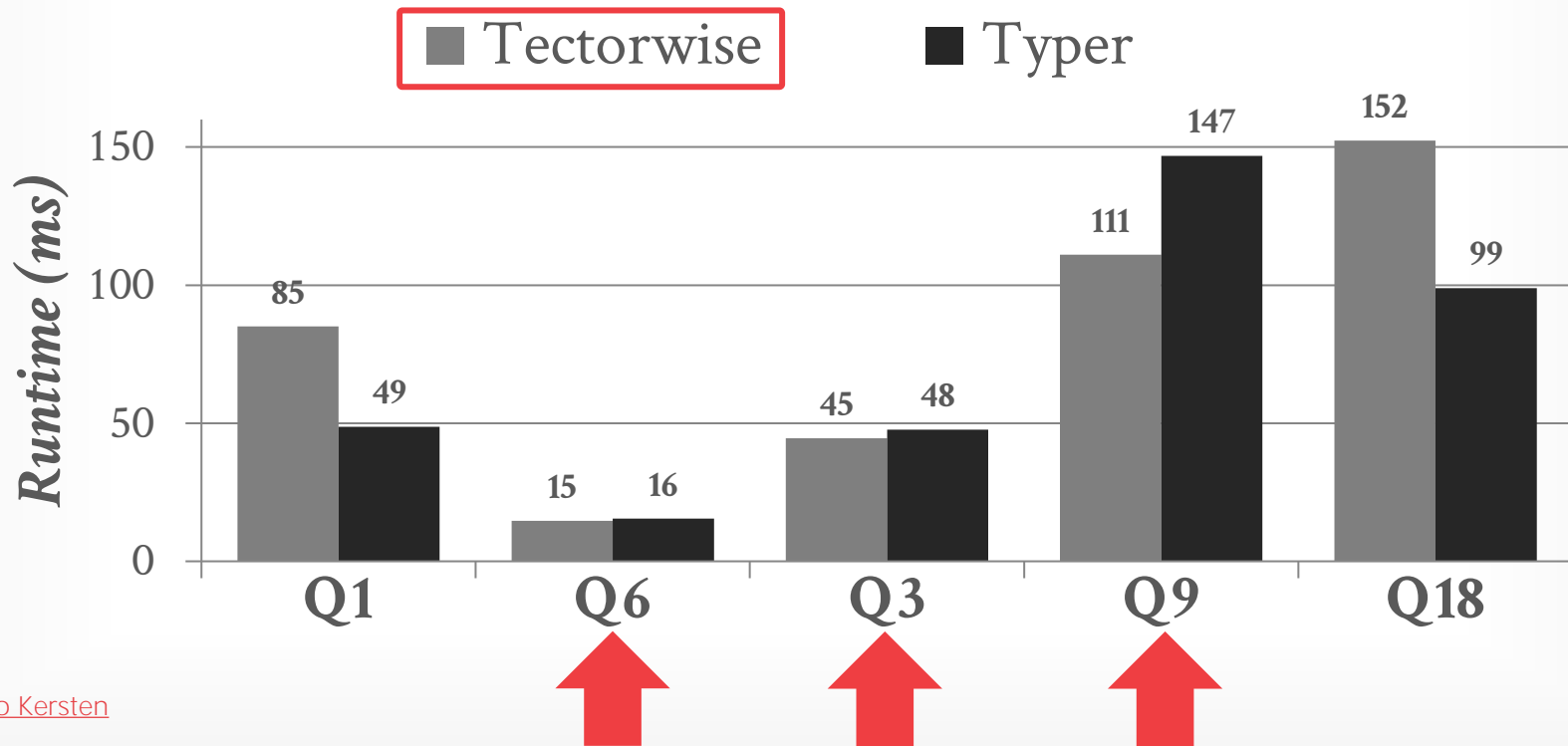
Intel Core i9-7900X (10 cores × 2HT)
TPC-H Queries (Scalefactor=1)



Source: [Timo Kersten](#)

SINGLE-THREADED PERFORMANCE

Intel Core i9-7900X (10 cores × 2HT)
TPC-H Queries (Scalefactor=1)



Source: [Timo Kersten](#)

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	★68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	★46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	★2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	★2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	★ 1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	★ 21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	★ 0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	56	★ 1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	★ 42	1.7	0.46	★ 0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

SINGLE-THREADED PERFORMANCE

		<i>Runtime</i>	<i>Cycles</i>	<i>IPC</i>	<i>Instr.</i>	<i>L1 Miss</i>	<i>LLC Miss</i>	<i>Br. Miss</i>
Q1	TWise	85	59	2.8	162	2.0	0.57	0.03
	Typer	48	34	2.0	68	0.6	0.57	0.01
Q6	TWise	15	11	1.4	15	0.2	0.29	0.01
	Typer	16	11	1.8	20	0.3	0.35	0.06
Q3	TWise	45	24	1.8	42	0.9	0.16	0.08
	Typer	48	25	0.8	21	0.5	0.16	0.27
Q9	TWise	111	★ 56	1.3	76	2.1	0.47	0.39
	Typer	147	74	0.6	42	1.7	0.46	0.34
Q18	TWise	152	48	2.1	102	1.9	0.18	0.37
	Typer	99	30	1.6	46	0.8	0.19	0.16

MAIN FINDINGS

Both models are efficient and achieve roughly the same performance.

→ 100x faster than row-oriented DBMSs!

Data-centric is better for "calculation-heavy" queries with few cache misses.

Vectorization is slightly better at hiding cache miss latencies.

SIMD PERFORMANCE

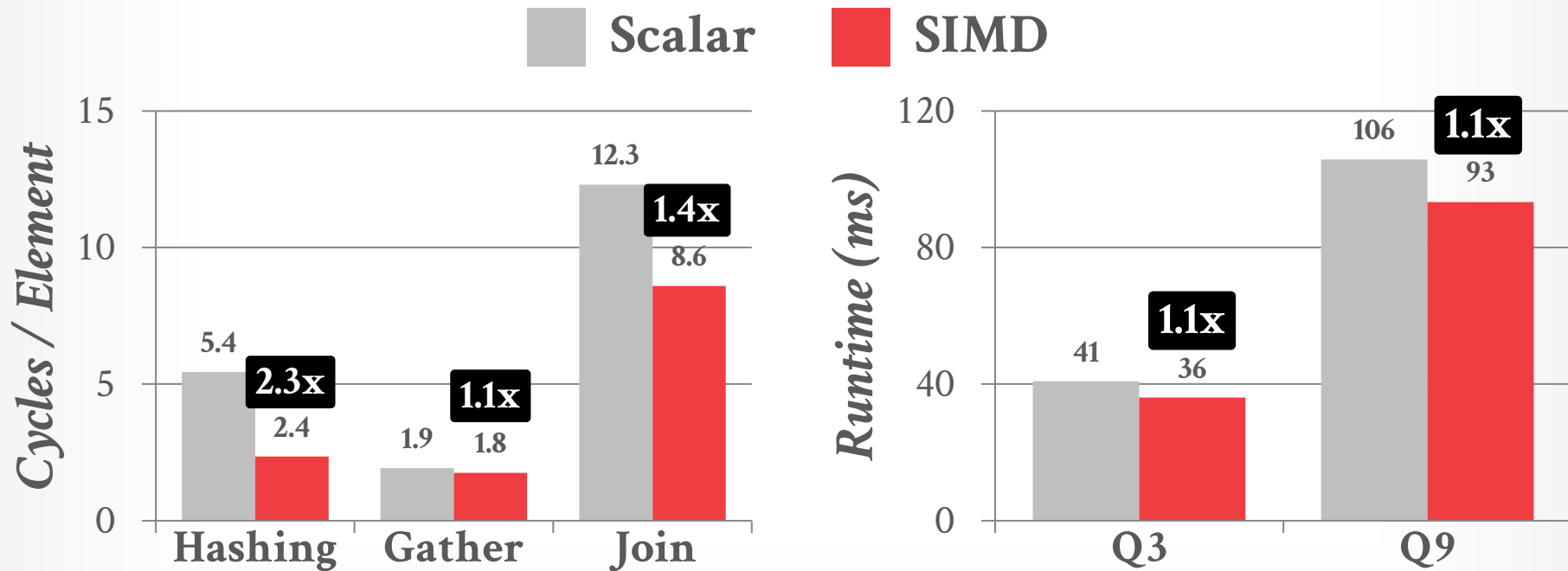
Evaluate vectorized branchless selection and hash probe in Tectorwise.

Use AVX-512 because it includes instructions to make it easier to implement algorithms using vertical vectorization.

→ Selective operations using bitmask registers.

SIMD EVALUATION`

Intel Core i9-7900X (10 cores × 2HT)
TPC-H Queries (Scalefactor=1)



Source: [Timo Kersten](#)

AUTO-VECTORIZATION

Evaluate how well the compiler can automatically vectorize the Vectorwise primitives.

→ Targets: GCC v7.2, Clang v5.0, ICC v18

ICC was able to vectorize the most primitives using AVX-512:

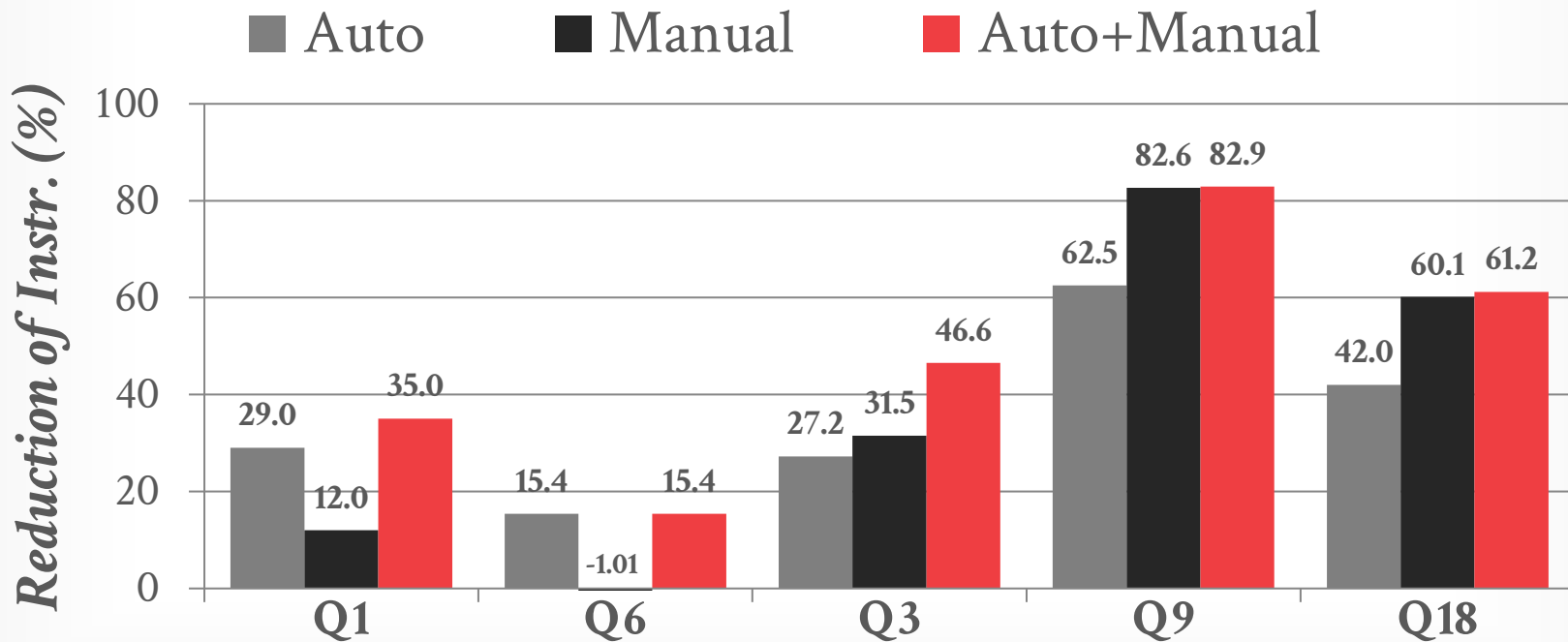
→ Vectorized: Hashing, Selection, Projection

→ Not Vectorized: Hash Table Probing, Aggregation

AUTO-VECTORIZATION

Intel Core i9-7900X (10 cores × 2HT)

Compiler: ICC v18

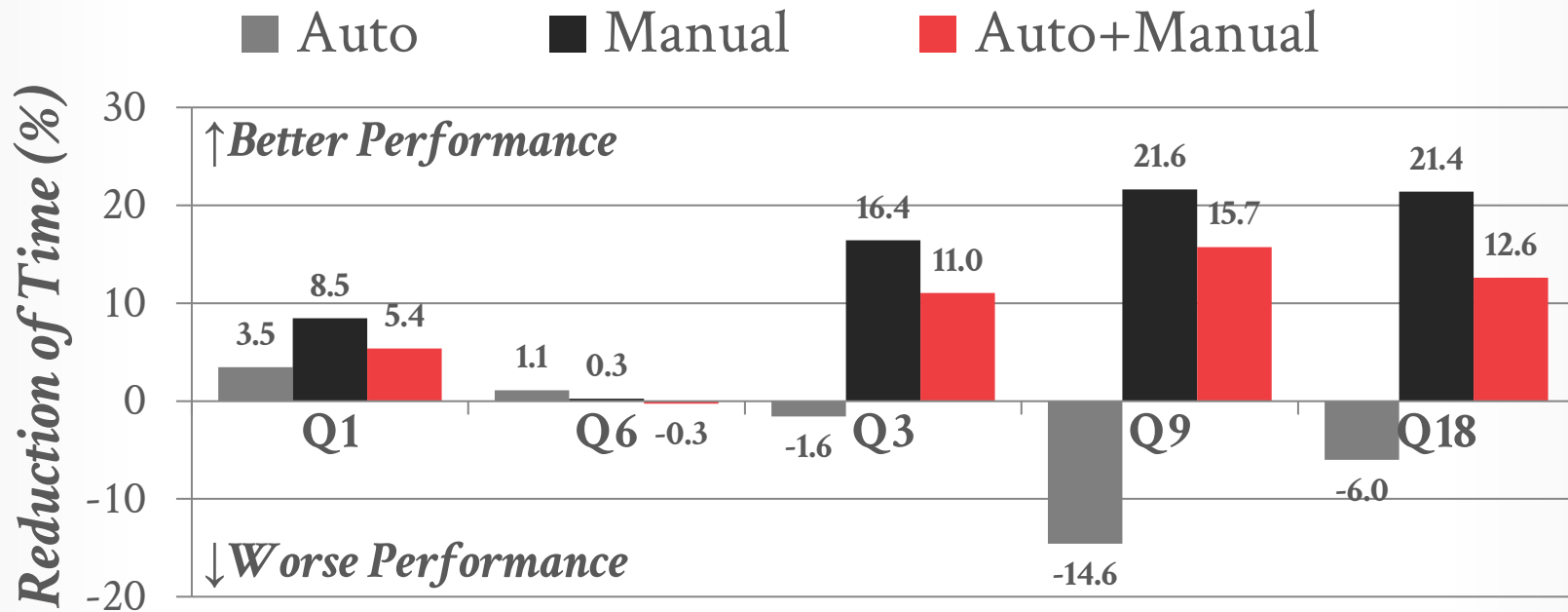


Source: [Timo Kersten](#)

AUTO-VECTORIZATION

Intel Core i9-7900X (10 cores × 2HT)

Compiler: ICC v18



Source: [Timo Kersten](#)

VECTORIZATION VS. COMPILATION

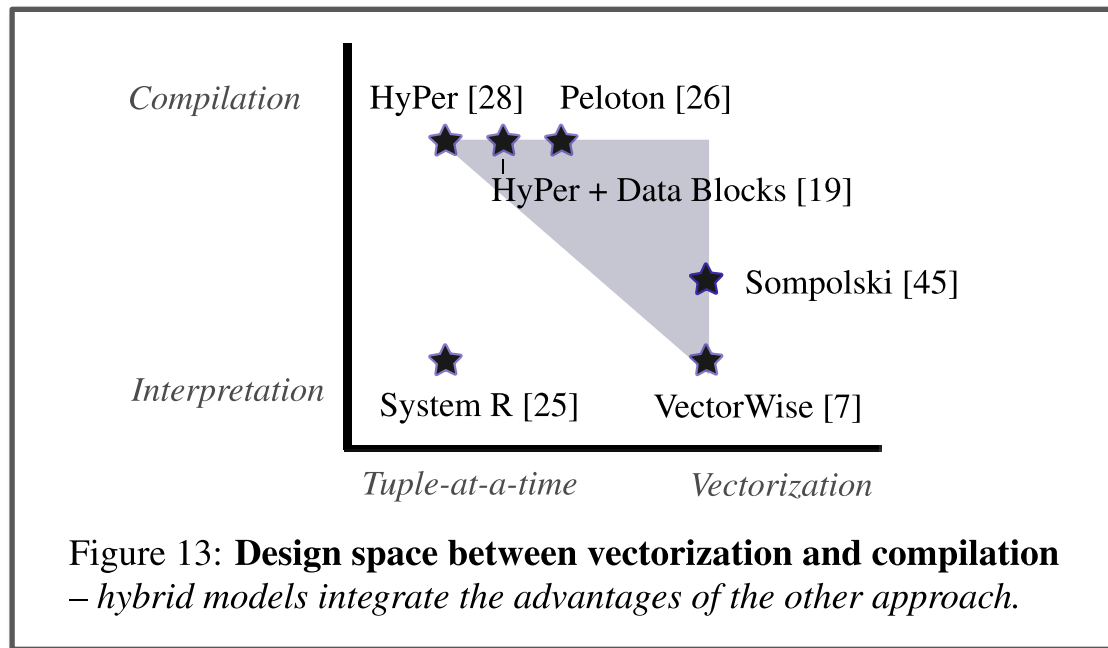


Figure 13: **Design space between vectorization and compilation**
– *hybrid models integrate the advantages of the other approach.*

PARTING THOUGHTS

No major performance difference between the Vectorwise and HyPer approaches for all queries.

NEXT CLASS

Hash Join Implementations

PROJECT #2 – DATABASE SYSTEM REPORT

Each student will write an encyclopedia article about the internals of a real-world DBMS.

→ We will target OLAP systems that implement topics from this semester.

Andy will post a sign-up sheet for you to pick what DBMS you want.

Feedback Due Date: April 1st

Final Due Date: May 1st



Prompt: An aerial photograph of Carnegie Mellon University but with all the buildings replaced with giant database drums.

<https://15721.courses.cs.cmu.edu/spring2023/project2.html>

DBDB.IO

Students will write their articles on CMU's online Database of Databases encyclopedia.

Each article will use a standard taxonomy.

- For each feature category, you select pre-defined options for your DBMS.
- You will then need to provide a summary paragraph with citations for that category.

DBDB.IO

Database of Databases | Browse | Leaderboards | Recent | Create | EmptyFields | pavlo

Database of Databases

Refine by

Start Year

End Year

Country

- ☐ Armenia
- ☐ Australia
- ☐ Austria
- [Show more](#)

Compatible With

- ☐ Access
- ☐ Accumulo
- ☐ BonDB
- [Show more](#)

Embeds / Uses

- ☐ BadgerDB
- ☐ Berkeley DB
- ☐ Boost
- [Show more](#)

Derived From

- ☐ Accumulo
- ☐ Adabas
- ☐ Adaptive Server Enterprise
- ☒ PostgreSQL
- [Show more](#)

Inspired By

- ☐ ArangoDB
- ☐ Aurora
- ☐ Berkeley DB
- [Show more](#)

Operating System

- ☐ AIX
- ☐ All OS with Java VM
- ☐ Android
- [Show more](#)

Programming Languages

- ☐ ActionScript
- ☐ Assembly
- ☐ Bash
- [Show more](#)

Tags

- ☐ Blockchain
- ☐ Cache

Search name, keywords, features...

AGENS Graph Database

Last updated Jan. 6, 2022, 12:19 p.m.

AlloyDB

Last updated May 15, 2022, 9:48 p.m.

AntDB

Last updated Feb. 15, 2022, 8:46 p.m.

MARIPOSA

Last updated June 27, 2022, 10:37 p.m.

TransLattice

Last updated June 27, 2022, 10:37 p.m.

PostgreSQL

PostgreSQL is an object-relational database based on Postgres, developed from University of California at Berkeley. It is ACID-compliant and supports materialized view, stored functions, triggers, and foreign keys. PostgreSQL is a free and open source software under the PostgreSQL License, still often referred to as Postgres by many people. Postgres has been continuously maintained since 1996 by the PostgreSQL Global Development Group, which consists of many companies as well as many individual contributors.

History

PostgreSQL is derived from Postgres, developed by University of California at Berkeley. Postgres was created by Michael Stonebraker as a follow up project to Ingres and released in 1984 (the name is meant to be "Post Ingres"). Two Berkeley Ph.D. students, Andrew Yu and Jody Chen, later brought a subset of SQL to Postgres and renamed the system to PostgreSQL. The system was then maintained and developed in the open source world outside of Berkeley, and finally renamed as PostgreSQL.

Checkpoints

PostgreSQL supports [pg_dump](#) to extract a consistent database into a file while concurrent transactions are running. While [pg_dump](#) only outputs a single database, [pg_dumpall](#) is used to dump all databases. For checkpoints, PostgreSQL supports periodic checkpointing as well as immediate checkpointing, which both guarantee that previous changes would be written to disk.

Concurrency Control

Multi-version Concurrency Control (MVCC)

PostgreSQL applies Multi-version Concurrency Control for data consistency. For MVCC, not only the current status but also previous values of data are visible to the transaction, which provides transaction isolation. The primary advantage of MVCC over locking is that the writing operation won't conflict with the reading operation on the same block of data. Thus, MVCC reduces the lock contention to achieve high throughput.

Data Model

Object database

PostgreSQL is an object-relational database (i.e., similar to relational database with an object-oriented database model). This means objects are supported in database schemas and PostgreSQL can be created, including: conversion, cast, functions, data types, domains, procedure languages and indexes.

Foreign Keys

Foreign key

PostgreSQL allows user to define a foreign key constraint, which means the values in that column match the keys in other tables. PostgreSQL also supports a foreign key to contain multiple columns. Besides, a label is allowed to define multiple foreign key constraints in PostgreSQL, which is usually used in many-to-many relations among the tables. When the table contains the primary key, which is referenced as foreign key in another table, is deleted, PostgreSQL provides several options, including a) Disallow deleting the referenced table to Delete the foreign key table as well c) something else.

Website
<https://www.postgresql.org/>

Source Code
<https://www.postgresql.org/source/>

Tech Docs
<https://www.postgresql.org/docs/>

Twitter
[@postgresql](#)

Developer
University of California-Berkeley

Country of Origin
US

Start Year
1985

Former Name
Postgres

Project Type
Academic, Open Source

Written in
C

Operating Systems
AIX, HP-UX, Linux, OS X, Solaris, UnixWare, Windows

Licenses
PostgreSQL License

Wikipedia
<https://en.wikipedia.org/wiki/PostgreSQL>

Revision #22 | Updated 06/20/2022 10:30 a.m.

Compatible Systems

- yugabyteDB
- Yellowbrick



PLAGIARISM WARNING



This article must be your own writing with your own images. You may **not** copy text/images directly from papers or other sources that you find on the web.

→ This includes **both** your submission for review and submission for your grade.

Plagiarism will **not** be tolerated.

See [CMU's Policy on Academic Integrity](#) for additional information.

PROJECT #3 – FINAL PROJECT

Group project to implement some substantial component or feature in a DBMS.

Projects should incorporate topics discussed in this course as well as from your own interests.

Each group must pick a project that is unique from their classmates.



Prompt: "A woman with a database drum for a head" by Johannes Vermeer

<https://15721.courses.cs.cmu.edu/spring2023/project3.html>

PROJECT #3 – DELIVERABLES

Proposal Presentation: March 1st

Status Update Presentation: April 3rd

Design Document: TBA

Final Presentation: TBA

PROJECT #3 – PROPOSAL

Five-minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:

- Architecture and implementation overview of the project.
- How you will test whether your implementation is correct.
- What workloads you will use for your project.

PROJECT #3 – STATUS UPDATE

Five-minute presentation to update the class about the current status of your project.

Each presentation should include:

- Current development status.
- Whether your plan has changed and why.
- Anything that surprised you during coding.

PROJECT #3 – DESIGN DOCUMENT

As part of the status update, you must provide a design document that describes your project implementation:

- Architectural Design
- Design Rationale
- Testing Plan
- Trade-offs and Potential Problems
- Future Work

PROJECT #3 – FINAL PRESENTATION

10-minute presentation on the final status of your project during the scheduled final exam.

You should include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too...

PROJECT TOPICS

Fast Fixed-Point Decimals (Standalone)

Database Proxy Acceleration (PostgreSQL)

Adaptive Query Opt. (PostgreSQL)

UDF Inlining (DuckDB)

FAST FIXED-POINT DECIMALS

The Germans claim that fixed-point decimals are faster than floating point decimals.

Project: Complete our implementation and integrate into PostgreSQL as UDT.

- Verify that the calculations are correct.
- Add support for vertical vectorization.
- Benchmark against PostgreSQL's built-in **NUMERIC**.

FAST FIXED-POINT D



LIBFIXEYPOINTY

We couldn't use the name "libfixedpoint" because it would be terrible for SEO...

PASSED

This is a portable C++ library for fixed-point decimals. It was originally developed as part of the [NoisePage](#) database project at Carnegie Mellon University.

This library implements decimals as 128-bit integers and stores them in scaled format. For example, it will store the decimal 12.23 with scale 5 1223000. Addition and subtraction operations require two decimals of the same scale. Decimal multiplication accepts an argument of lower scale and returns a decimal in the higher scale. Decimal division accepts an argument of the denominator scale and returns the decimal in numerator scale. A rescale decimal function is also provided.

$$2^{2^5} + 1 = 641 \cdot 6700417$$

$$2^{2^6} + 1 = 274177 \cdot 67280421310721$$

$$\text{avg}(x, y) = (x \& y) + ((x \oplus y) \gg 1)$$

$$x - y = x + \bar{y} + 1$$

$$\lfloor a \rfloor + \lfloor b \rfloor \leq \lfloor a + b \rfloor \leq \lfloor a \rfloor + \lfloor b \rfloor + 1$$

$$\text{pop}(x) = -\sum_{i=0}^{31} (x \ll i) \& 1$$

$$\sqrt{11111111} = 1111$$

$$(x \neq 0) = (x \mid -x) \gg 31$$

$$\text{mux}(x, y, m) = ((x \oplus y) \& m) \oplus y$$

$$A(n, d) = A(n-1, d-1), d \text{ even}$$

$$-\bar{x} = x + 1$$

Hacker's Delight

SECOND EDITION

$$1111^2 = 11100001$$

$$n = -2^{31}b_{31} + 2^{30}b_{30} + 2^{29}b_{29} + \dots + 2^0b_0$$

$$\lceil x \rceil = -\lfloor -x \rfloor$$

$$f(x, y, z) = g(x, y) \oplus zh(x, y)$$

$$\text{Num factors of 2 in } x = \log_2(x \& (-x)), x \neq 0$$

$$\text{rjust}(x) = x \gg (x \& -x), x \neq 0$$

$$p_i = 1 + \sum_{n=1}^i \left[\sum_{k=1}^n \left[\cos^2 \pi \frac{(k-1)(i+1)}{k} \right] \right]^{1/n}$$

$$x \oplus y = (x \mid y) - (x \& y)$$

$$x + y = (x \mid y) + (x \& y)$$

HENRY S. WARREN, JR.

DATABASE PROXY ACCELERATION

We have been working on optimizing network operations in PostgreSQL proxies.

Project: Extend pgCat (Rust) to support user-bypass and/or kernel-bypass.

- User-bypass: eBPF
- Kernel-bypass: io_uring
- Matt has existing benchmark scripts to compare against his proxy and Odyssey.

ADAPTIVE QUERY OPTIMIZATION

We want to be able to change a query plan during execution without stopping the query.

Project: Create a PostgreSQL extension that swaps a plan node in the tree with a "dummy" node.

- New node can either halt execution or generate fake data.
- An easier approach might be to wrap nodes with "control" nodes that determine whether to call inner node.

UDF INLINING

We want to compare methods for compiling UDFs into machine code versus UDF inlining.

→ We will cover this in [Lecture #14](#)

Project: Add support for PL/pgSQL UDF inlining in DuckDB.

- PostgreSQL's query optimizer is too primitive.
- DuckDB supports nested query decorrelation, which is needed for the Microsoft Froid technique.
- Potentially in collaboration with (different) Germans.

HOW TO START

Form a team. Sign-up on class spreadsheet.

Meet with your team and discuss potential topics.

Look over source code and determine what you will need to implement.

I am able during Spring Break for additional discussion and clarification of the project idea.