HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Datenbanken: Querykompilierung und vektorisierte Ausführung

Dr. Matthias Uflacker, Stefan Halfpap, Dr. Werner Sinzig

3. Juli 2019

Vorlesungsinhalte/-aufbau

- Einführung zu Unternehmensanwendungen (2 Vorlesungen)
- Einführung zu relationalen Datenbanken und Anfrageverarbeitung (2 Vorlesungen)
- Grundlagen des IT-gestützten Rechnungswesens und der Planung (3 Vorlesungen)
- Grundlagen von (spaltenorientierten) Hauptspeicherdatenbanken (5 Vorlesungen)
- **Trends in Hauptspeicherdatenbanken (4 Vorlesungen)**
- Klausur

Literatur und Empfehlung

Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask

Timo Kersten Viktor Leis Alfons Kemper Thomas Neumann Andrew Pavlo[◇] Peter Boncz[⋆]
Technische Universität München Carnegie Mellon University[◇] Centrum Wiskunde & Informatica[⋆]
{kersten,leis,kemper,neumann}@in.tum.de pavlo@cs.cmu.edu boncz@cwi.nl

ABSTRACT

The query engines of most modern database systems are either based on vectorization or data-centric code generation. These two state-of-the-art query processing paradigms are fundamentally different in terms of system structure and query execution code. Both paradigms were used to build fast systems. However, until today it is not clear which paradigm yields faster query execution, as many implementation-specific choices obstruct a direct comparison of architectures. In this paper, we experimentally compare the two mod-

Like the Volcano-style iteration model, vectorization uses pull-based iteration where each operator has a *next* method that produces result tuples. However, each *next* call fetches a block of tuples instead of just one tuple, which amortizes the iterator call overhead. The actual query processing work is performed by primitives that execute a simple operation on one or more type-specialized columns (e.g., compute hashes for a vector of integers). Together, amortization and type specialization eliminate most of the overhead of traditional engines.

Empfehlung

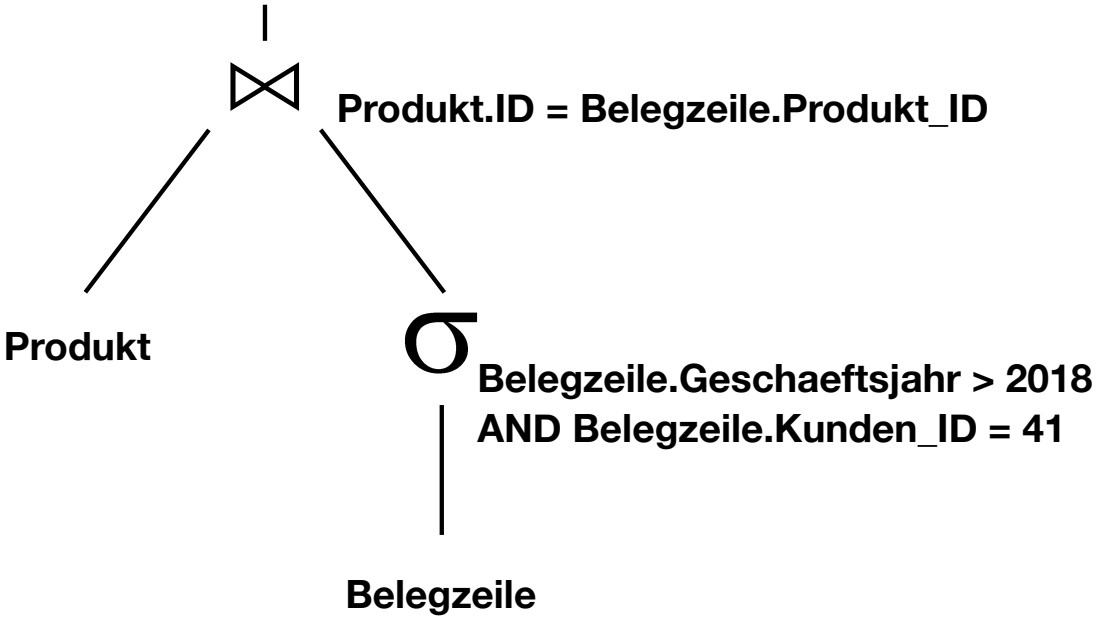
- Andy Pavlo „Database Systems“
  - Query Compilation

    https://15721.courses.cs.cmu.edu/spring2019/slides/19-compilation.pdf
  - Vectorized Execution

    https://15721.courses.cs.cmu.edu/spring2019/slides/20-vectorization1.pdf
  - Vectorization vs. Compilation

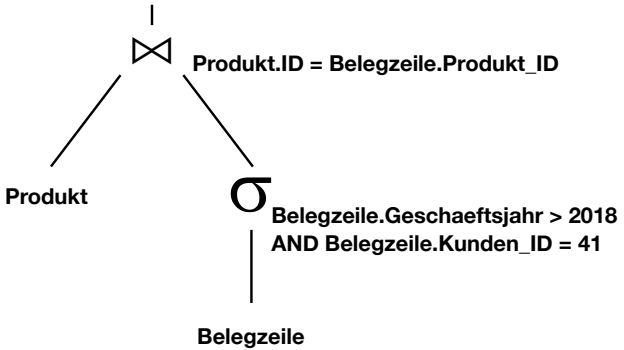
    https://15721.courses.cs.cmu.edu/spring2019/slides/21-vectorization2.pdf

Anfrageausführung



Querykompilierung und vektorisierte Ausführung
Motivation

- Hauptspeicherdatenbanken sind schnell
- Was ist der limitierende Faktor einer DB, wenn Daten nicht mehr von der Festplatte gelesen werden müssen?
  - Hauptspeicherzugriff und Ausnutzung der CPU-Caches
    → Spaltenorientierung für OLAP und „Mixed Workloads“
    → Datenkompression
  - Aufwand von vielen Funktionsaufrufen beim klassischen Tupel-At-A-Time-Iteratorenmodell
    → vektorisierte Ausführung (siehe auch 06_DB_Tabellenrepraesentation_im_Speicher_*)
  - (Anzahl der) CPU-Anweisungen
    → Codespezialisierungen und –generierung/Querykompilierung



Querykompilierung

- Anfrage/Query ist Folge von generischen Operatoren
- Idee: Erhöhe die Performanz durch das Verschmelzen von Operatoren und die Reduktion der CPU-Anweisungen
  - Handgeschriebene Query ist schnell, aber nicht praktikabel/automatisierbar
  - Daher automatisierter Ansatz zur Codegenerierung

- Erste moderne Umsetzung in HyPer

  Neumann „Efficiently Compiling Efficient Query Plans for Modern Hardware “ (2011)

  https://www.vldb.org/pvldb/vol4/p539-neumann.pdf

Querykompilierung
Codegenerierungformen & Vorteile

Codegenerierung

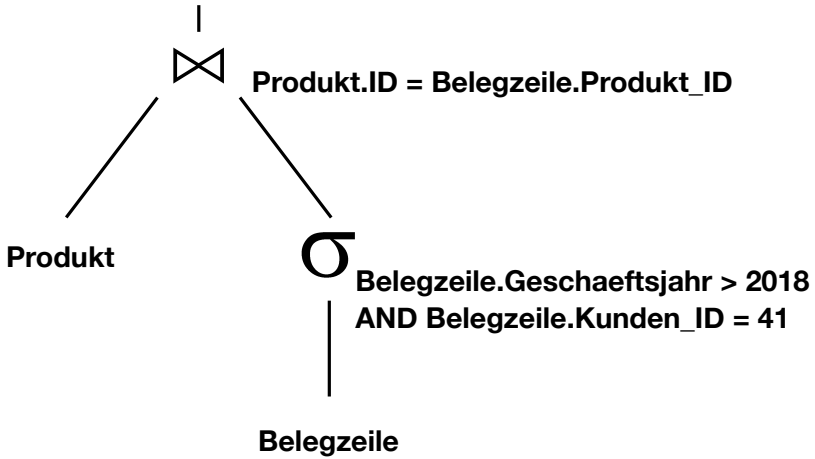
- Generiere C/C++-Programm, z.B. umgesetzt in HIQUE https://15721.courses.cs.cmu.edu/spring2019/papers/19-compilation/krikellas-icde2010.pdf
- Generiere LLVM IR, z.B. umgesetzt in HyPer http://www.hyper-db.de/

Vorteile

- Attributtypen sind bekannt
  → Datenzugriffsfunktionen können in direkte Typumwandlungen (inline casting) umgewandelt werden
- Prädikate sind bekannt
  → Können direkt mit nativen Vergleichsfunktionen ausgewertet werden
- Keine Funktionsaufrufe in Schleifen
  → Verbessertes Caching

Querykompilierung
Beispiel (Funktionsweise)

```
for (int i=0; i<lineitem.size; i++) {
    if (year[i] > 2018 && customer_id[i] == 41) {
        // insert into hash table
        ...
    }
}
```



Vektorisierte Ausführung

- Anfrage/Query ist Folge von generischen Operatoren auf vielen Daten
- Idee: Erhöhe die Performanz durch die Verarbeitung von Datenblöcken/Vektoren mit einfachen Datentypen um Kosten für Funktionsaufrufe und Wertextraktionen zu amortisieren oder vermeiden
  - Austausch von Zwischenergebnisse in Form von Datenblöcken/Vektoren
  - Verwende spezialisierte/typisierte Operatoren

- Erste moderne Umsetzung in MonetDB/VectorWise
  Boncz et al. „MonetDB/X100:Hyper-PipeliningQueryExecution“ (2005)

http://cidrdb.org/cidr2005/papers/P19.pdf

Vektorisierte Ausführung
SIMD, Vektorisierungsformen & Vorteile

Viele moderne Prozessoren unterstützen SIMD(single instruction multiple data)-Befehle, die die gleiche Operation auf vielen (primitiven) Datenelementen gleichzeitig ausführen können

Vektorisierungsformen (/wann werden SIMD-Befehle genutzt)

- Automatische Vektorisierung (durch den Compiler für einfache Schleifen)
- (zusätzliche) Compilerhinweise (, dass Code vektorisiert werden kann)
- Explizite Vektorisierung (durch Aufrufe von SIMD-Befehlen im Programmcode)

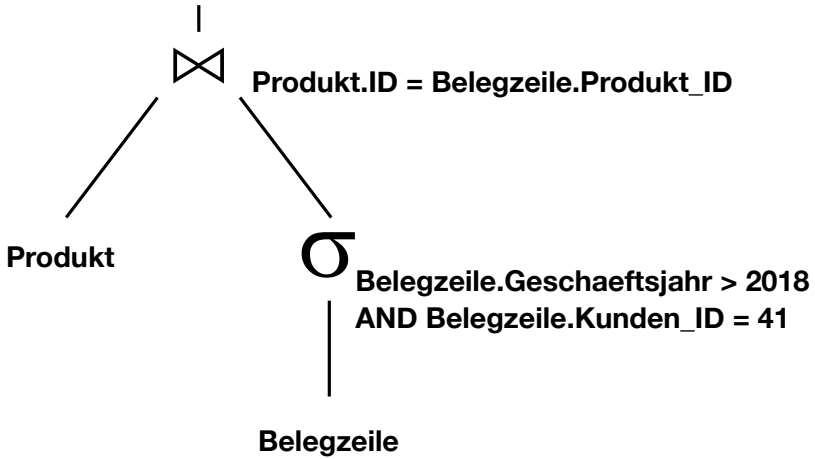
Vorteile

- Ausführung von einfachen/optimierbaren Operationen auf typisierten Spalten
- Profitieren von vektorisierter Ausführung durch den Prozessor (SIMD)

Vektorisierte Ausführung
Beispiel (Funktionsweise)

```
vector_t pos_list;
for (int i=0; i<customer_id.size; i++) {
    if (customer_id[i] == 41) {
        pos_list.append(i);
    }
}
return pos_list

vector_t pos_list;
for (int i=0; i<customer_id_pos_list.size; i++) {
    if (year[customer_id_pos_list[i]] > 2018) {
        pos_list.append(i);
    }
}
return_pos_list
```



Querykompilierung und vektorisierte Ausführung OLAP-Performanz

Generell schwer isoliert zu vergleichen, da meist in verschiedenen Systemen umgesetzt, was weitere Performanzeinflussfaktoren mit sich bringt

(Motivation des Papers von Kersten et al.)

Ergebnisse der Forschungsergebnisse

- Beide Ansätze sind effizient und ähnlich performant
  (haben aber unterschiedliche Stärken und Schwächen)
- **Querykompilierung hat Vorteile bei rechenintensiven Anfragen mit wenigen Cache-Misses**, weil Daten in CPU-Registern gehalten und somit weniger CPU-Befehle ausgeführt werden
- **Vektorisierte Ausführung hat Vorteile bei parallelen Datenzugriff mit vielen Cache-Misses**, die aufgrund einfacher Schleifen besser kompensiert werden können

Querykompilierung und vektorisierte Ausführung Vergleich (weiterer Gesichtspunkte)

- OLTP-Performanz
- Kompilierungszeit
- Profiling
- Debugging
- Anpassungsfähigkeit der Anfrageausführung

Querykompilierung und vektorisierte Ausführung Vergleich - OLTP-Performanz

Querykompilierung

- Kompilierung kann effizienten Maschinencode für alle Anfragearten erzeugen

Vektorisierte Ausführung

- Effizienz durch Verarbeitung von Vektoren mit vielen Werten (OLAP)
- OLTP-Anfragen greifen häufig auf wenige Tupel zu

  → Wenig Vorteil gegenüber Tupel-At-A-Time-Iteratorenmodell

Querykompilierung und vektorisierte Ausführung Vergleich - Kompilierungszeit

Querykompilierung

- Kompilierungszeit kann Ausführungszeit dominieren
- OLTP: Codegenerierung für gespeicherte Prozeduren (stored procedures) im Voraus möglich
- OLAP: Kompilierungszeit steigt mit Codegröße
- Generell: Kompromiss von Kompilierzeit und Ausführungszeit möglich

  Kohn et al. „Adaptive Execution of Compiled Queries “ (2018) https://db.in.tum.de/~leis/papers/adaptiveexecution.pdf

Querykompilierung und vektorisierte Ausführung Vergleich - Profiling

Querykompilierung

- Laufzeitanalyse einzelner Operatoren schwierig, da Operatoren verschmolzen sind

Vektorisierte Ausführung

- Laufzeitanalyse von Operatoren möglich ohne Anfragen zu verlangsamen, da Funktionen auf Vektoren mit vielen Elementen arbeiten

Querykompilierung und vektorisierte Ausführung Vergleich - Debugging

Querykompilierung

- Schwierig zu debuggen, da das Datenbanksystem selbst Compiler ist und damit auch schwieriger zu verstehen ist
- Besonders, wenn der generierte Code maschinennah (low-level) ist
- Kompilierungssysteme können Codegenerierung vereinfachen

Tahboub et al. „How to Architect a Query Compiler“ (2018) https://www.cs.purdue.edu/homes/rompf/papers/tahboub-sigmod18.pdf

Querykompilierung und vektorisierte Ausführung Vergleich - Anpassungsfähigkeit der Anfrageausführung ..

.. um (unvermeidbare) ungenaue Kardinalitätsschätzungen der Queryoptimierung und damit (möglicherweise) schlechte Anfragepläne während der Ausführung anzupassen

Querykompilierung

- Integration schwierig

Vektorisierte Ausführung

- Plan anpassbar, z.B.:
  - Reihenfolge von Filterprädikaten und Joins
  - Alternative Implementierungen von Operatoren z.B. beim Join

Raducanu and Boncz „Micro Adaptivity in Vectorwise " (2013) https://15721.courses.cs.cmu.edu/spring2019/papers/19-compilation/p1231-raducanu.pdf

Querykompilierung und vektorisierte Ausführung Vergleich - Zusammenfassung

Querykompilierung	**Vektorisierte Ausführung**
+ OLTP-Ausführung	- OLTP-Ausführung
- Kompilierungszeit	+ keine Codegenerierung zur Ausführung
- Profiling schwierig	+ Profiling
- Debugging schwierig	+ Debugging
- Anpassungsfähige Ausführung	+ Anpassungsfähige Ausführung

Zusammenfassung

- Die Anfrageausführung moderner Datenbanksysteme basieren auf Vektorisierung oder Anfragekompilierung/Codegenerierung
- Beide Modelle sind effizient, haben aber unterschiedliche Stärken und Schwächen
- Vektorisierung kann Cache-Misses besser kompensieren
- Querykompilierung benötigt weniger CPU-Befehle