3. Übungsblatt

Ausgabe: 23. Mai 2016 · Besprechung: 30. Mai/6. Juni 2016

1 Row-/Column-Oriented Storage (Besprechung 30. Mai 2016)

In dieser Aufgabe wollen wir den Effekt einer spaltenorientierten Speicherung auf die Cache-Effizienz untersuchen. Dazu sollen Daten aus dem TPC-H-Benchmark verwendet werden. TPC-H ist ein Standardbenchmark zur Evaluation von Data Warehouse-Systemen. Im besonderen wollen wir uns die `lineitem`-Tabelle anschauen, die wir zur Vereinfachung als Array im Hauptspeicher abbilden wollen.

Darstellung im Hauptspeicher

Eine **zeilenorientierte** Darstellung würde dabei einem Array entsprechen, dessen Basis der folgende `struct` ist:

```
typedef struct {
   uint32_t orderkey;         /* identifier */
   uint32_t partkey;          /* identifier */
   uint32_t suppkey;          /* identifier */
   int64_t  quantity;         /* decimal */
   int64_t  extendedprice;    /* decimal */
   int64_t  discount;         /* decimal */
   int64_t  tax;              /* decimal */
   char     returnflag;       /* fixed text, size 1 */
   char     linestatus;       /* fixed text, size 1 */
   uint32_t commitdate;       /* date */
   uint32_t receiptdate;      /* date */
   char     shipinstruct[26]; /* fixed text, size 25 */
   int32_t  linenumber;       /* integer */
   char     shipmode[11];     /* fixed text, size 10 */
   char     comment[45];      /* variable text, size 44 */
   uint32_t shipdate;         /* date */
} lineitem_t;
```

Um die Relation **spaltenorientiert** abzubilden, kann stattdessen für jedes Feld der Relation ein einzelnes Array verwendet werden (sei `N` die Anzahl Zeilen der Relation):

```
/* row store */                 /* column store */
lineitem_t relation[N];         uint32_t orderkey[N];
                                uint32_t partkey[N];
                                ...
                                char     comment[45*N];
                                uint32_t shipdate[N];
```

Beispielanfrage

Wir wollen konkret eine C-Implementation der folgenden SQL-Anfrage untersuchen:

```
SELECT SUM (orderkey + linenumber * shipdate)
  FROM lineitem
```

Teilaufgabe 1: Vorüberlegung

Welches Speicherlayout wird aus den obigen C-Konstrukten entstehen?

Welches Verhalten erwarten Sie, wenn für die gegebenen Speicherlayouts die gegebene Anfrage als Scan ausgewertet wird?

Teilaufgabe 2: Messung

Implementieren Sie nun die gegebene Anfrage einmal für ein zeilen- und einmal für ein spaltenorientiertes System. Dazu liegt auf der Webseite eine Vorlage bereit.

Zeilenorientierte Darstellung. Ergänzen Sie hier den Code für die SQL-Anfrage in der Datei `row_store.c` (aktuell ist dort als Platzhalter ein C-Kommentar, der den SQL-Code enthält). In der Vorlage ist bereits Code enthalten, der TPC-H-Daten laden kann.

Spaltenorientierte Darstellung. Analog finden Sie in der Datei `column_store.c` eine Vorlage für eine spaltenorientierte Darstellung.

Untersuchen Sie beide Varianten in Bezug auf ihre (relative) Laufzeit. Verwenden Sie dazu eine hinreichend große Instanz der `lineitem`-Tabelle, um aussagekräftige Messungen zu erhalten.

TPC-H-Datengenerator

Einen Datengenerator für TPC-H-Daten können Sie auf der Webseite des TPC (`http://www.tpc.org/`) herunterladen. Extrahieren Sie die Quellen, kopieren Sie `makefile.suite` nach `Makefile` und ergänzen Sie am Anfang der Datei die Werte für `CC`, `DATABASE` (wählen Sie einfach '`DB2`'), `MACHINE` und `WORKLOAD`. Danach sollte sich der Code kompilieren lassen mit `make`.

Anschließend können Sie mit `dbgen` Daten automatisch generieren lassen. Der Parameter `-s` gibt dabei den Skalierungsfaktor an.

2 Operator-At-A-Time, Tuple-At-A-Time, Vector-At-A-Time (Besprechung: 06. Juni 2016)

In den Quelldateien zur obigen Aufgabe finden Sie auch eine Codevorlage für einen einfachen *Volcano*-basierten Anfrageprozessor. Konkret ist in `engine.h` eine Menge von Operatoren deklariert. Aus diesen Operatoren wird in `cs_iterator.c` ein Anfrageplan zusammengesetzt, der der Anfrage

```
SELECT  MAX(orderkey)
  FROM  lineitem
```

entspricht.

Die Vorlage sieht dabei bereits ein "Vector-At-A-Time"-Verarbeitungsmodell vor (mit dem sich wiederum durch Wahl der Vektorgröße die anderen Modelle simulieren lassen, siehe Vorlesung).

Aufgaben:

1. Ergänzen Sie die fehlenden Operatorimplementationen (insbesondere die Funktionen `next_*`. Am besten legen Sie dazu eine neue Datei `engine.c` an, die die Implementationen enthält. Wahrscheinlich müssen Sie außerdem einige der `structs` in `engine.h` ergänzen.

2. Auf Vorlesungsfolie 60 wurde der Einfluss der Vektorgröße auf die Ausführungszeit am Beispiel des MonetDB/X100-Systems gezeigt. Erstellen Sie ein entsprechendes Diagramm auch für Ihre Implementation eines "Vector-At-A-Time"-Prozessors.