VectorWise

Michał Świtakowski
Kamil Anikiej

6 października 2011

Agenda

- Czym jest VectorWise?
- Przetwarzanie zapytań
- Projekty magisterskie
- Projekty do zrealizowania
- Podsumowanie

vectorwise

Analityczna baza danych

- Istnieją 2 główne rodzaje systemów bazodanowych: OLTP (Online Transaction Processing) i OLAP (Online Analytical Processing)
- Systemy OLAP:
  - Przechowują historyczne dane
  - Są zoptymalizowane pod duże wolumeny danych
  - Są wykorzystywane przy badaniu tendencji rynkowych, długoterminowych danych statystycznych itp.

vectorwise

VectorWise

VectorWise (www.vectorwise.com) to nowa, innowacyjna analityczna baza danych.

- Początkowo projekt badawczy na CWI w Amsterdamie (MonetDB/X100)
- Główni twórcy: Peter Boncz, Marcin Żukowski
- W tej chwili kompletny produkt znajdujący coraz więcej klientów i osiągający świetne wyniki w benchmarkach (www.tpc.org/tpch) – najszybsza nierozproszona baza danych
- Zespół liczy 12 programistów, w tym 7 Polaków

vectorwise

Motywacja

TPC-H Query 1

- Przetwarza 6 milionów krotek wykonując proste projekcje i agregacje
- `SELECT A, B, SUM(C), AVG(D*(1-E)), ...`
  `FROM LINEITEM WHERE X < 100 GROUP BY A, B ORDER BY A, B`
- Wydajność:
  - MySQL: 28.1s
  - DBMS „X”: 28.1s
  - program w C: 0.2s

vectorwise

Motywacja

Dlaczego tak wolno?

- Nieefektywny model przetwarzania zapytań
- Nieefektywny sposób przechowywania i ładowania danych

vectorwise

Przetwarzanie zapytań

- Volcano pipelened model – np. MySQL
- Operator at a time – MonetDB
- Zwektoryzowany model Volcano – VectorWise

vectorwise

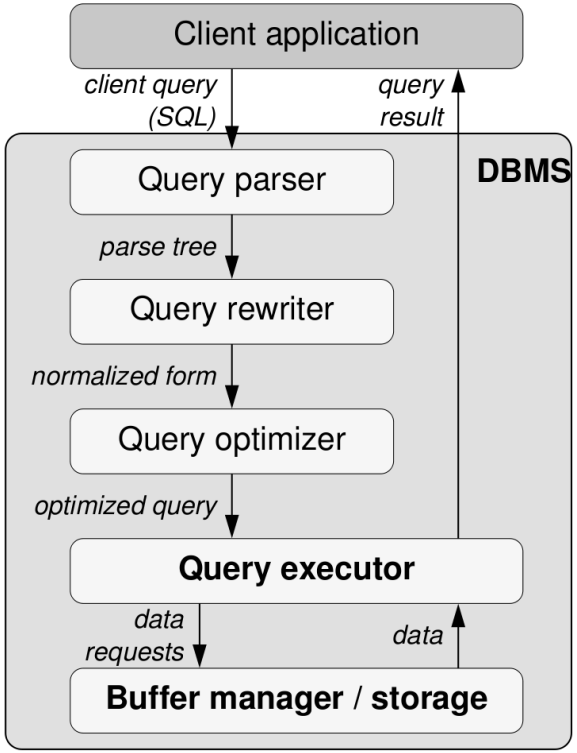
Volcano model

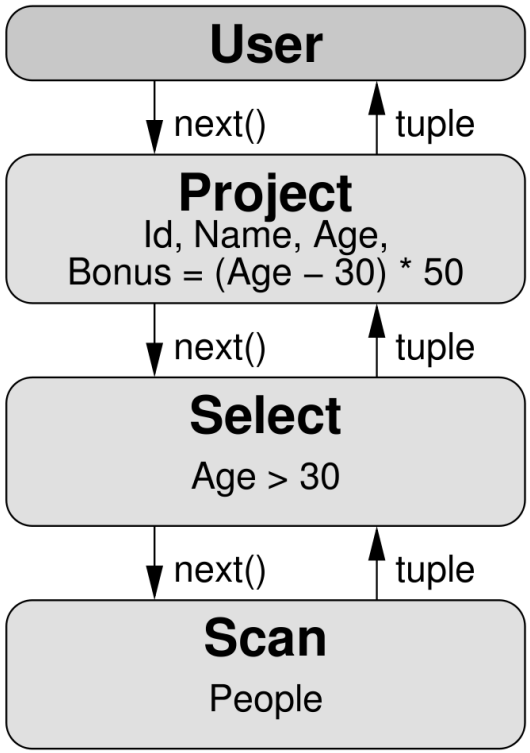
W modelu Volcano:

- Zapytanie jest reprezentowane jako drzewo operatorów
- Każdy operator implementuje interfejs: open(), next(), close()
- Krotki wędrują w górę drzewa przy każdym wywołaniu next()

vectorwise

Volcano model





vectorwise

Volcano model – wady

- Dużo niepotrzebnego, powtarzającego się kodu
  - Logika operatorów
  - Wywołania funkcji
  - Dostęp do atrybutów
- Większość instrukcji interpretuje zapytanie
- Niewiele instrukcji przetwarza faktyczne dane
- Wysoki wskaźnik IPT (instruction per tuple)

vectorwise

Współczesne procesory

Należy także uwzględnić charakterystykę współczesnych procesorów

- Wolny dostęp do RAM-u, szybki do cache (cache L1 – 4 cykle, L2 – 11 cykli, L3 – 38 cykli, RAM – 50-300 cykli)
- Przetwarzanie potokowe
- Superskalarność
- Instrukcje SIMD (MMX, SSE)

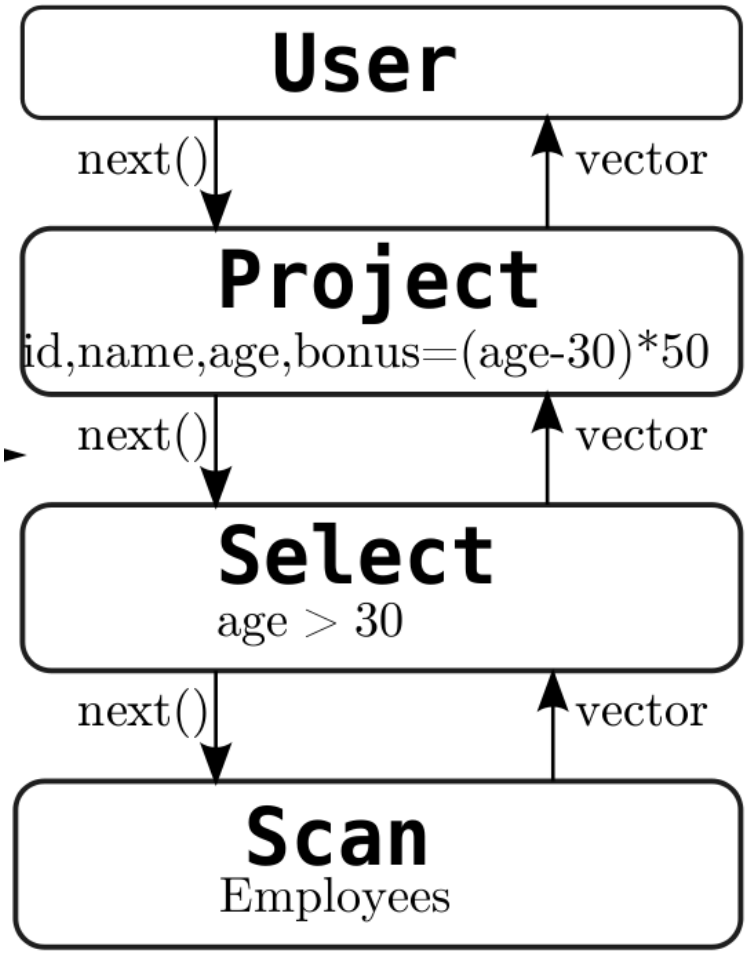
vectorwise

Rozwiązanie – model wektorowy

- Pomysł: zamiast 1 krotki przetwarzamy cały wektor krotek (100 - 10000 elementów)
- Co możemy zyskać:
  - Wielokrotnie niższy narzut na interpretację
  - Lepsze wykorzystanie cache
  - Kod prostszy do optymalizacji dla kompilatora
  - Użycie instrukcji SIMD

vectorwise

Vectorized model



id	name	age	bonus
1	adam	39	450
~~2~~	~~eva~~	~~25~~	
3	tom	31	50
4	marc	60	1500

sel: 0, 2, 3

id	name	age
1	adam	39
~~2~~	~~eva~~	~~25~~
3	tom	31
4	marc	60

sel: 0, 2, 3

id	name	age
1	adam	39
2	eva	25
3	tom	31
4	marc	60
5	chris	42

...

vectorwise

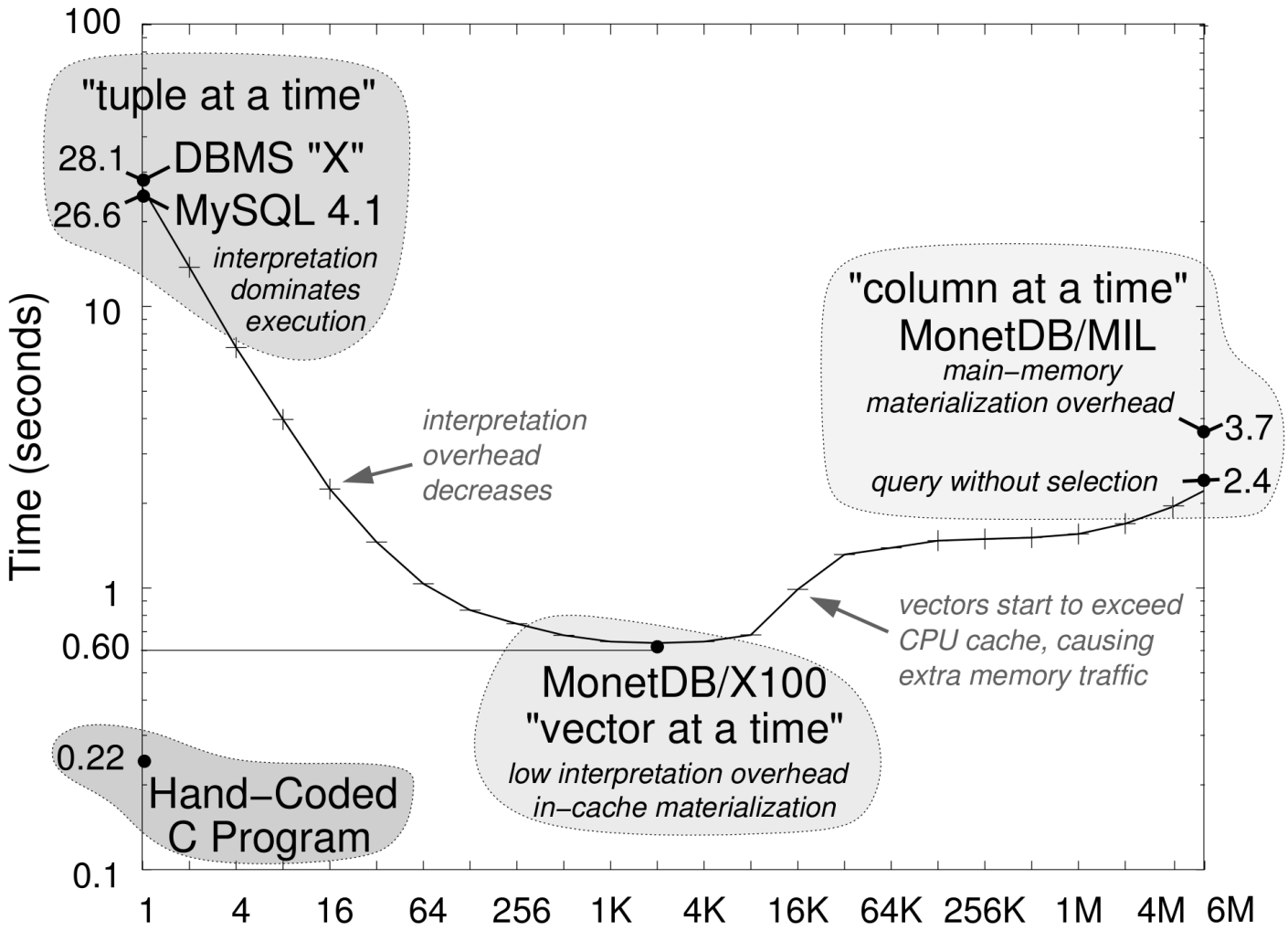
Prymitywki

- Prymitywki to wyspecjalizowane funkcje przetwarzające wektory
- Generowane są makrami z uwzględnieniem rodzaju operacji, typów itp.

```
int map_mul_flt_col_flt_col(int n, flt* res, flt* col1, flt* col2, int *sel)
{
  for(int i=0; i<n; i++)
      res[sel[i]] = col1[sel[i]] * col2[sel[i]];
  return n;
}
```

vectorwise

Model wektorowy – wydajność



vectorwise

Rozwiązania w VectorWise

Podsumowując, najważniejsze techniki poprawiające wydajność w VectorWise to:

- Przetwarzanie wektorowe w cache
- Wykorzystanie SIMD
- Kolumnowe przechowywanie danych (DSM)
- Zapis skompresowanych danych, dekompresja w cache
- Aktualizacje w pamięci w postaci drzew PDT (Positional Delta Trees)

vectorwise

Projekty magisterskie

vectorwise

Projekty magisterskie

- Kamil Anikiej – Parallel execution
- Fabian Nagel – Recycler
- Alicja Łuszczak – Compressed execution
- Juliusz Sompolski – Just in Time Compilation
- Michał Świtakowski – Cooperative Scans

vectorwise

Compressed Execution

vectorwise

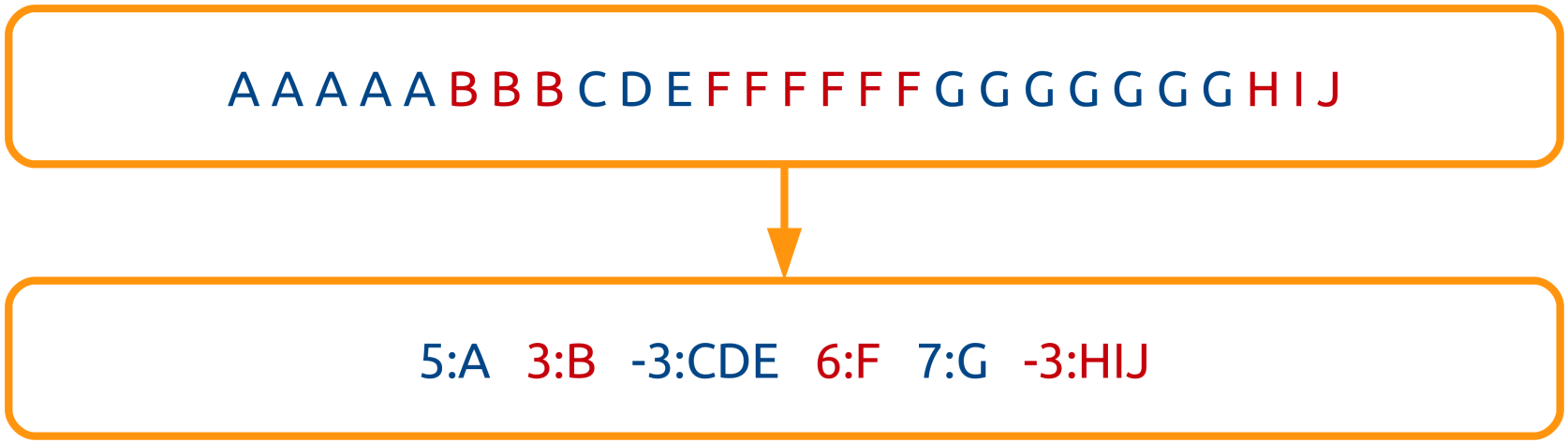
Compressed execution – idea

VectorWise przechowuje w postaci skompresowane, aby wystarczająco szybko dostarczać dane procesorowi z dysku.

- Tradycyjne podejście polega na wykonaniu dekompresji przed przetwarzaniem
- Niektóre metody umożliwiają przetwarzanie danych bez konieczności dekompresji
- Cel: wykorzystać możliwości przetwarzania skompresowanych danych bez dużych zmian w systemie
  - Run length encoding (RLE)
  - Kompresja słownikowa

vectorwise

RLE



vectorwise

RLE w przetwarzaniu wektorowym

- Pomysł: dodać flagę do wektora mówiącą o tym czy zawiera identyczne wartości
- Ustawianie flagi: podczas ładowania bloku skompresowanego za pomocą RLE
- Flagę można wykorzystać przy wyborze prymitywki

vectorwise

RLE – prymitywki

Prymitywka oryginalna:

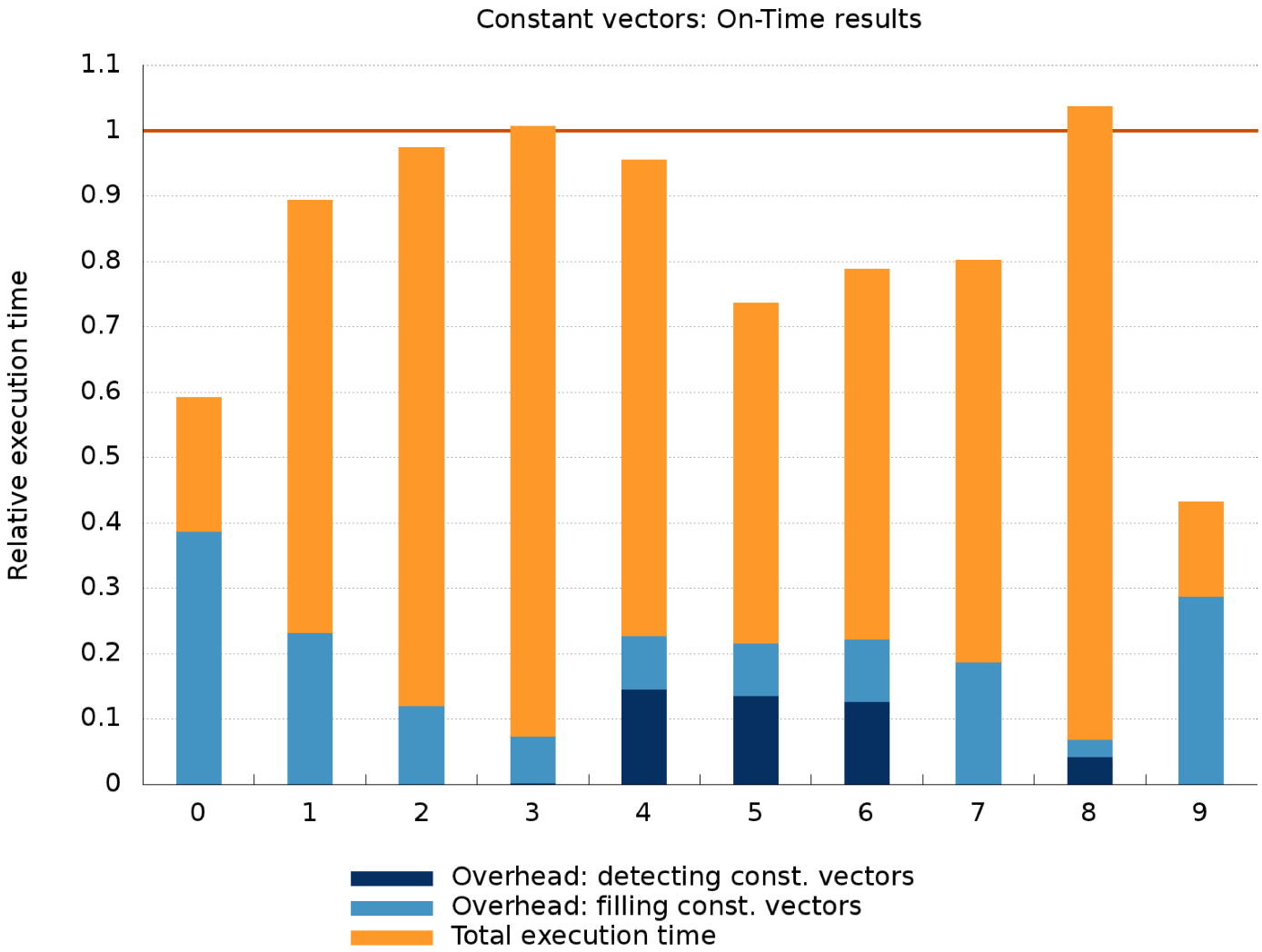
```
map_add_sint_col_sint_col (n, res, arg1, arg2) {
        for (i = 0; i < n; i++)
                res[i] = arg1[i] + arg2[i]
}
```

Prymitywka zoptymalizowana:

```
map_add_sint_col_sint_val (n, res, arg1, arg2) {
        for (i = 0; i < n; i++)
                res[i] = arg1[i] + *arg2
}
```

vectorwise

Compressed execution – wyniki („on time” benchmark)



vectorwise

Kompresja słownikowa – idea

vectorwise

Kompresja słownikowa – przetwarzanie

Skompresowane wartości umożliwiają szybsze:

- sprawdzanie równości
- porównywanie zakresów (jeśli kompresja zachowuje porządek)
- eliminowanie duplikatów
- obliczanie wartości funkcji hashujących

Podstawowym wymogiem jest jednolite kodowanie całej kolumny.

vectorwise

Słowniki

TableA	
...	*City*
...	‘Atlanta’
...	‘Boston’
...	‘Boston’
...	‘Chicago’
...	‘Atlanta’

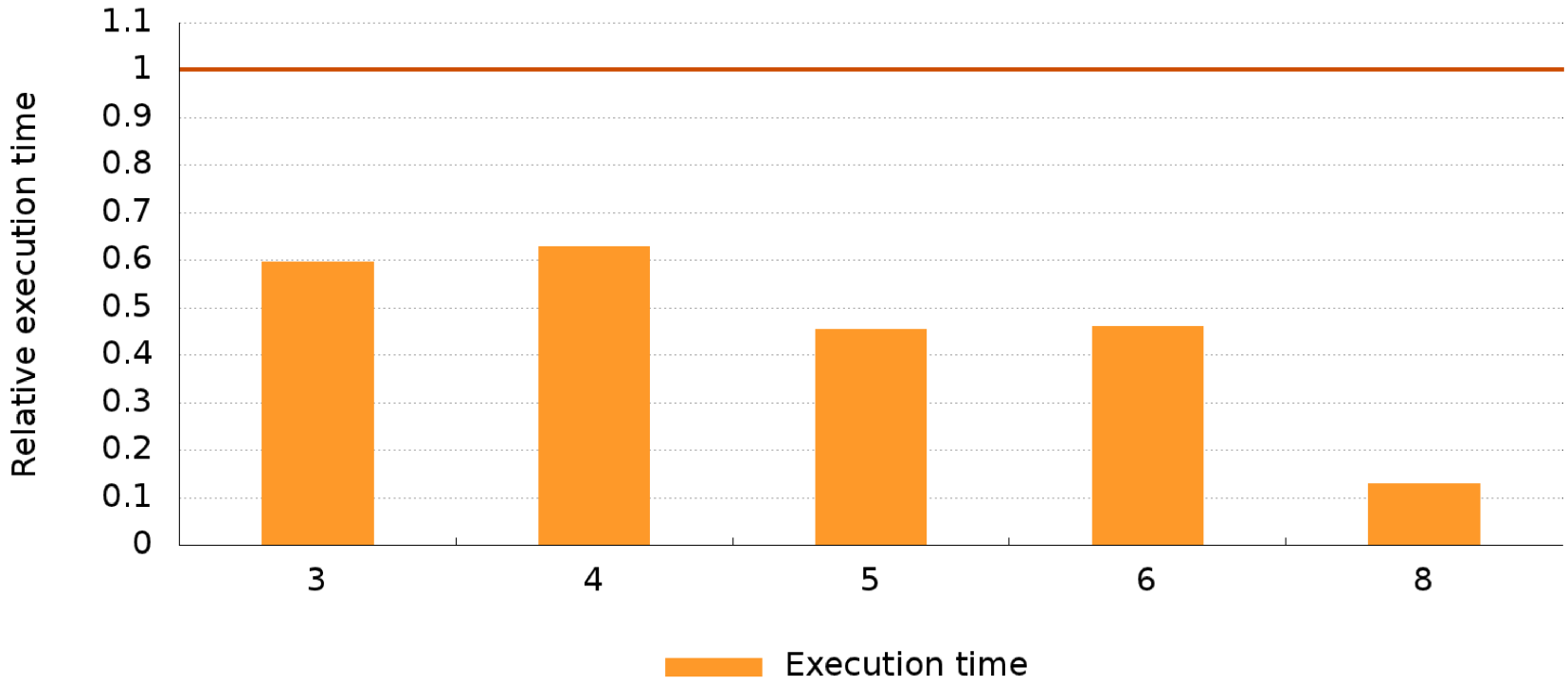
TableA	
...	*City*
...	1
...	2
...	2
...	3
...	1

+

CityDictionary	
‘Atlanta’	1
‘Boston’	2
‘Chicago’	3

vectorwise

Kompresja słownikowa – wyniki



vectorwise

JIT Compilation

vectorwise

JIT Compilation

Idea: przy zapytaniach, które trwają kilka sekund może opłacać się poniesienie kosztu kompilacji.
Operatory, które mogą wykorzystać kompilację:

- Project
- Select
- HashAggregation, HashJoin

vectorwise

JIT Compilation

```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem
```

- l_extprice – 4 bajty szerokości, pozostałe 1 bajt
- (1-l_discount)*(1+l_tax) – 2 bajty szerokości
- ostateczny wynik – 4 bajty szerokości

vectorwise

Realizacja w VectorWise aktualnie – dane wejściowe

```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem
```

```
// input
i32 l_extprice[N];
i8 l_discount[N],l_tax[N];
i8 v1 = 100, v2 = 100;
// output
u32 res[N];
// intermediates
i8 tmpi8_1[N], tmpi8_2[N];
i16 tmpi16_1[N], tmpi16_2[N], tmpi16_3[N];
i32 tmpi32_1[N];
```

vectorwise

Realizacja w VectorWise aktualnie – dane wejściowe

```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem

// 1+l_tax
map_i8_add_i8_col_i8_val(N, tmpi8_1,
                         l_tax, v1, sel);
// 1-l_discount
map_i8_sub_i8_val_i8_col(N, tmpi8_2, v2,
                         l_discount, sel);
// (1-l_discount)*(1+l_tax)
map_i8_to_i16_i8_col(N, tmpi16_1, tmpi8_1, sel);
map_i8_to_i16_i8_col(N, tmpi16_2, tmpi8_2, sel);
map_i16_mul_i16_col_i16_col(N, tmpi16_3,
                            tmpi16_1, tmpi16_2, sel);
// l_extprice*(1-l_discount)*(1+l_tax)
map_i32_to_i32_i16_col(N, tmpi32_1,
                       tmpi16_3, sel);
map_i32_mul_i32_col_i32_col(N, res, l_extprice,
                            tmpi32_1, sel);
```

vectorwise

Realizacja projekcji w VectorWise bez rzutowania

```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem

// Optimal vectorized code with primitives
   for more combinations of types:
map_i8_add_i8_val_i8_col(N, tmpuchr1,
                            &v1, l_discount, sel);
map_i8_sub_i8_val_i8_col(N, tmpuchr2,
                            &v2, l_tax, sel);
map_u16_mul_i8_col_i8_col(N, tmpi16_1,
                             tmpi8_1, tmpi8_2, sel);
map_i32_mul_i32_col_i16_col(N, res,
                               l_extprice,
                               tmpi16_1, sel);
```

vectorwise

Realizacja projekcji w VectorWise z compound primitive

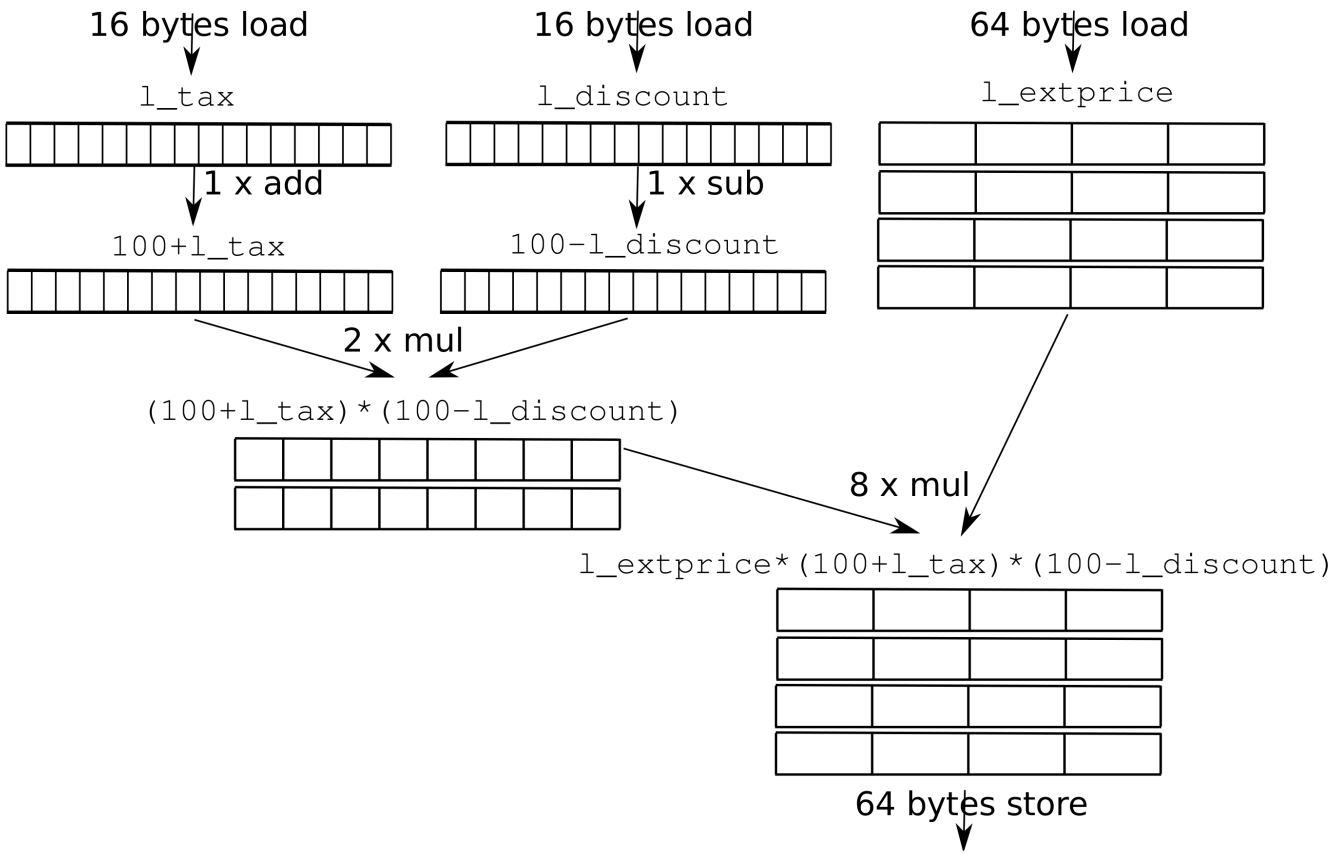
```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem


// Compiled equivalent of this expression:
uidx map_compiled(uidx n, i32 *res, i32 *col1,
                  i8 *col2, i8 *col3) {
  for(uidx i=0; i<n; i++)
    res[i]=col1[i]*((100-col2[i])*(100+col3[i]));
}
```

vectorwise

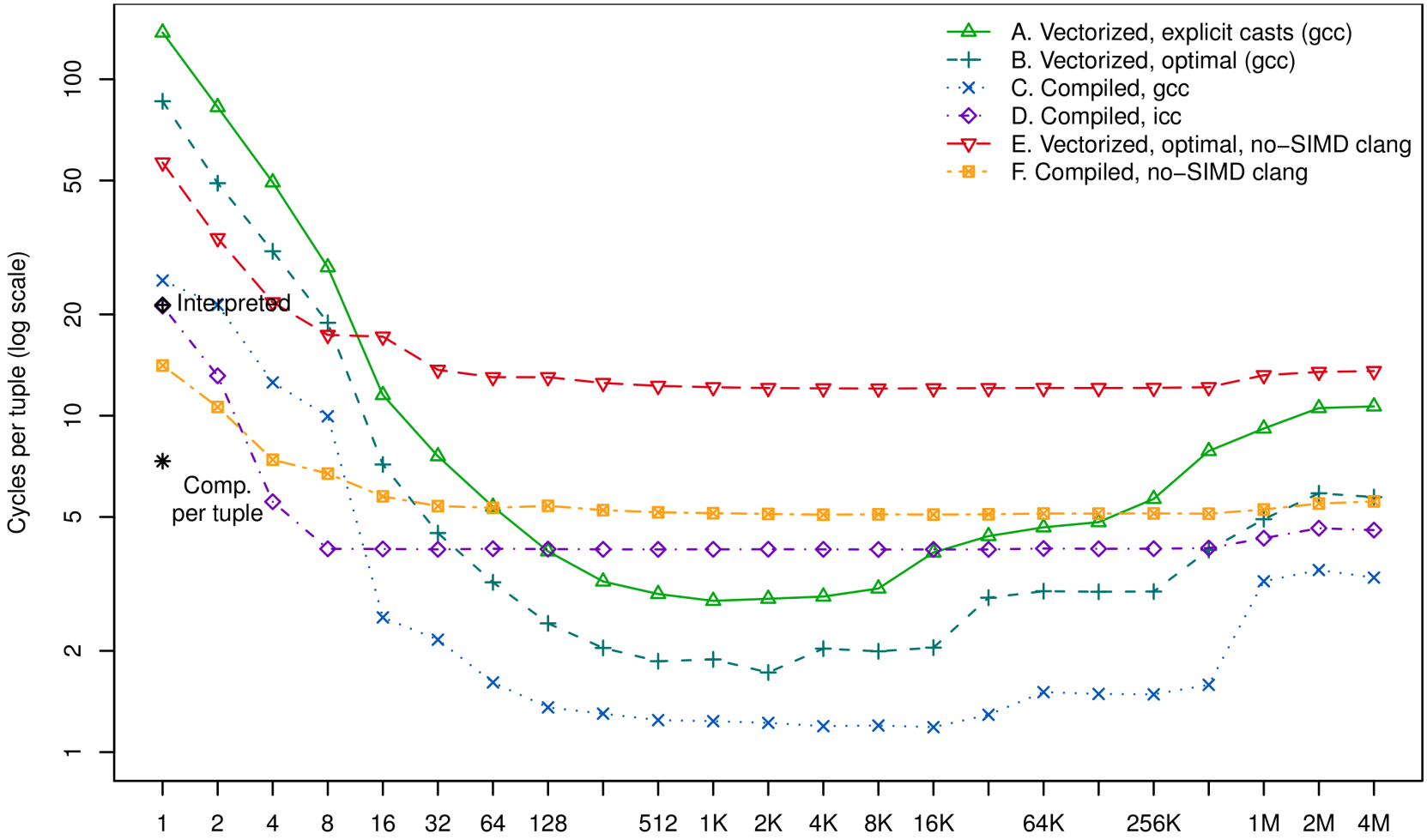
Realizacja projekcji w VectorWise z SIMD

```
SELECT l_extprice*(1-l_discount)*(1+l_tax)
FROM lineitem
```



vectorwise

JIT Compilation – wyniki



vectorwise

Cooperative Scans

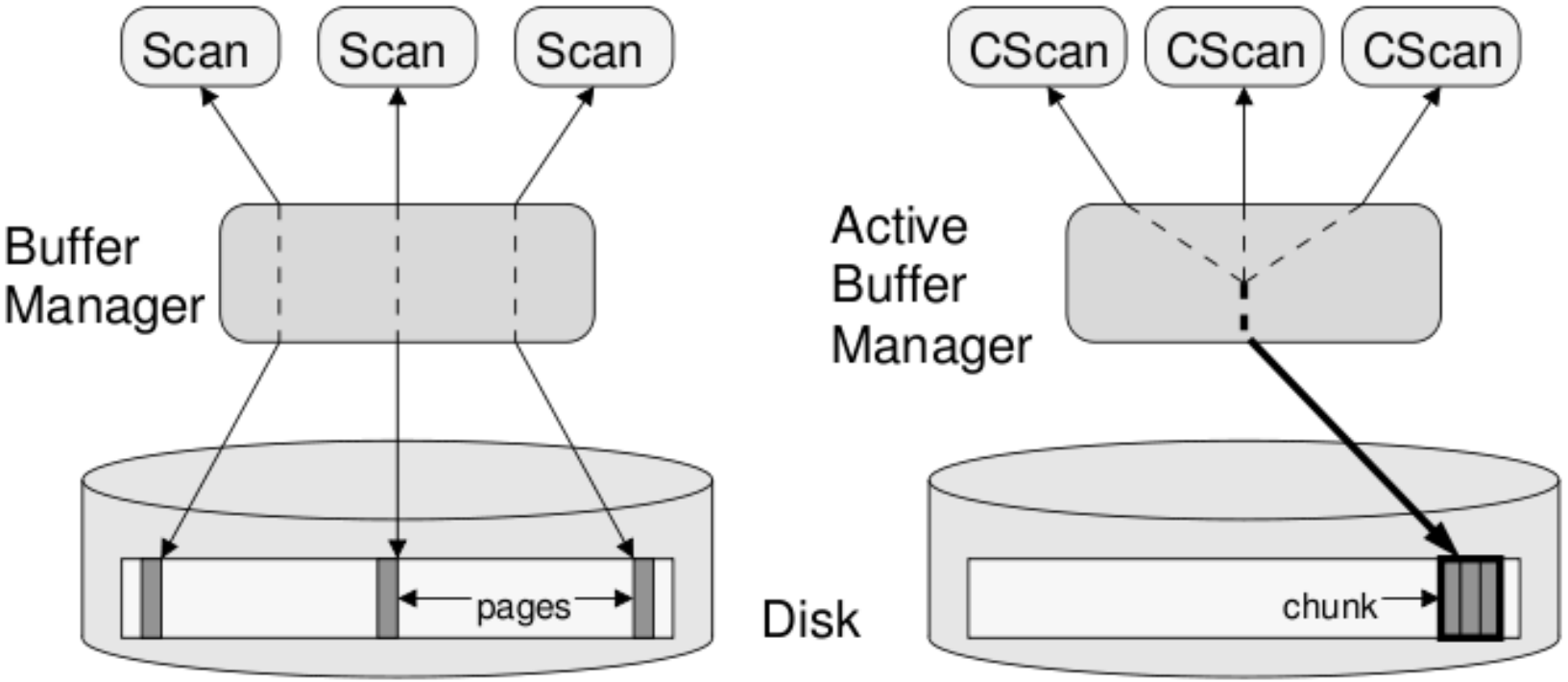
vectorwise

Cooperative Scans – motywacja

- Większość zapytań czyta duże ilości danych
- Dane są czytane sekwencyjnie w kolejności w jakiej są zapisane na dysku (operator „Scan”)
- W wielu zastosowaniach uruchamia się kilka zapytań równolegle
- Proste polityki zarządzania buforem są nieefektywne (np. LRU)

vectorwise

Cooperative Scans vs. Normal Scans



vectorwise

Cooperative Scans – idea

Zamiast izolować od siebie zapytania pozwólmy im współpracować na poziomie bufora. Aby to osiągnąć:

- Rejestrujemy wszystkie operatory Scan w ABM (Active Buffer Manager)
- Dzielimy tabelę na duże kawałki (chunks)
- Pozwalamy przetwarzać dane poza kolejnością
- Do zarządzania buforem używamy funkcji *Relevance (queryRelevance, loadRelevance, useRelevance, keepRelevance)

vectorwise

Cooperative Scans – funkcje *Relevance

Zamiast izolować od siebie zapytania pozwólmy im współpracować na poziomie bufora. Aby to osiągnąć:

- queryRelevance – wybieraj najpierw zagłodzone zapytania, spośród nich faworyzuj te krótsze
- loadRelevance – wybieraj kawałki potrzebne dla wielu zapytań
- useRelevance – wybieraj kawałki potrzebne dla jak najmniejszej ilości zapytań (przyśpiesz zwalnianie pamięci)
- keepRelevance – zwalniaj kawałki, które są potrzebne jak najmniejszej ilości zapytań

vectorwise

Join indeksy w VectorWise

- Aby przyśpieszyć łączenie dużych tabel VectorWise używa algorytmu MergeJoin.
- Tabele, które uczestniczą w operacji MergeJoin są posortowane na dysku
- Problem: Cooperative Scans zakładają przetwarzanie poza kolejnością

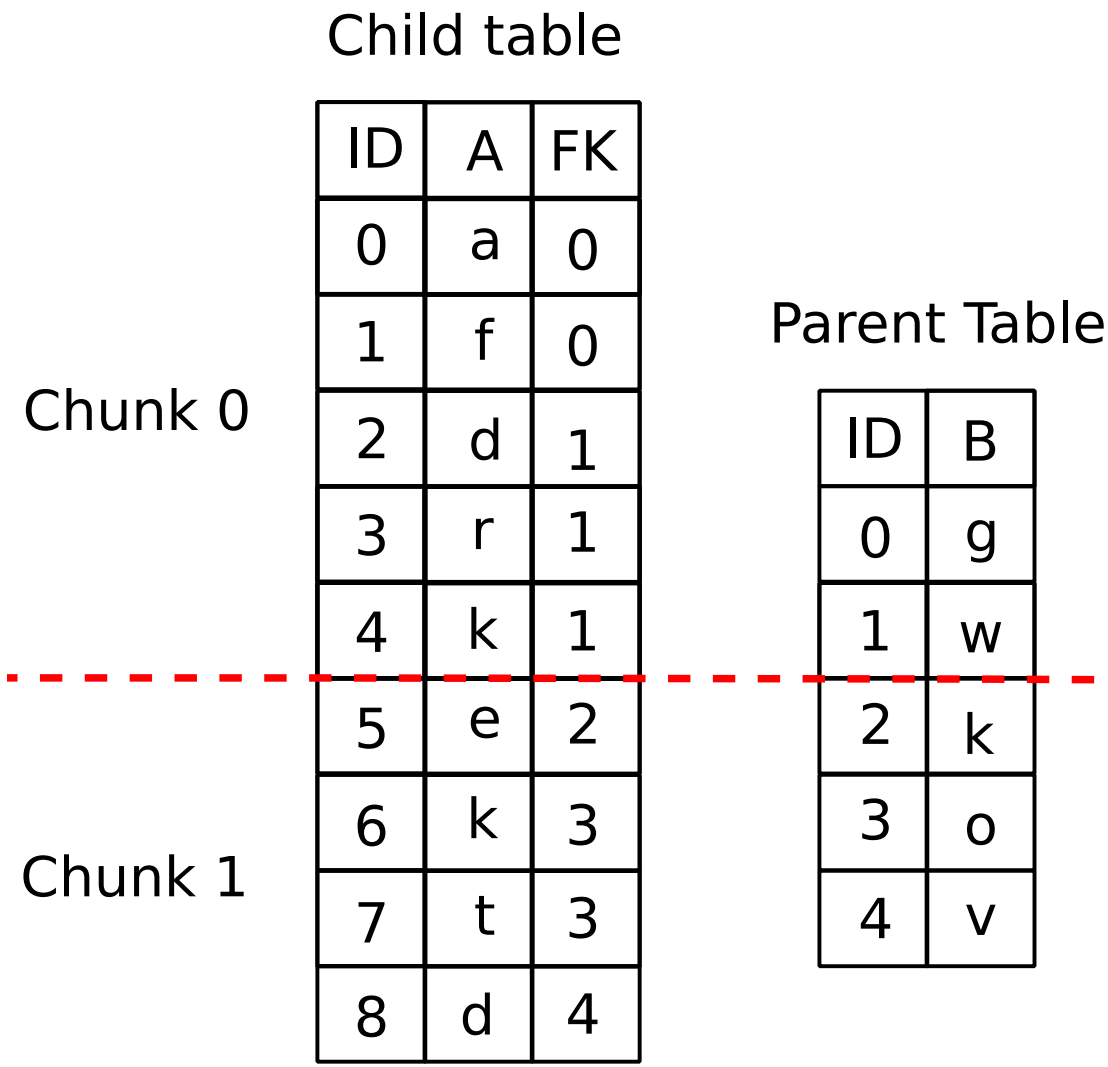
vectorwise

Cooperative Merge Join

- Obserwacja: wystarczy, że dane są posortowane wewnątrz kawałków
- Należy wyrównać granice kawałków do granic kluczy po obu stronach
- Podczas dostarczania danych ABM musi synchronizować dwa operatory CScan

vectorwise

Cooperative Merge Join



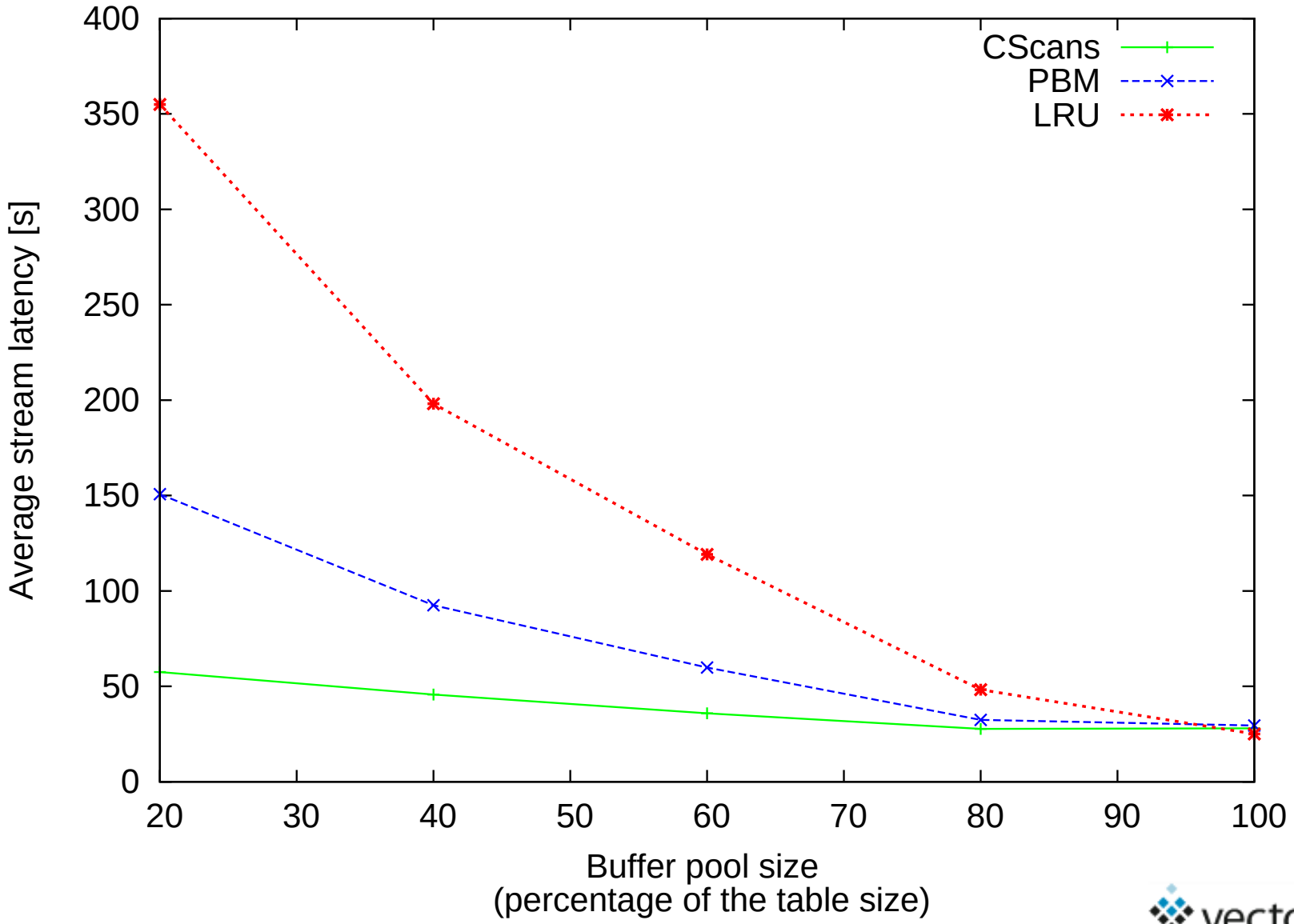
vectorwise

Benchmark

- TPC-H scale factor 40 (10 GB danych, używane 2 GB)
- Zapytania Q1 i Q6 skanujące losowo 1%, 10%, 50% lub 100% tabeli
- 16 równoległych strumieni po 4 zapytania

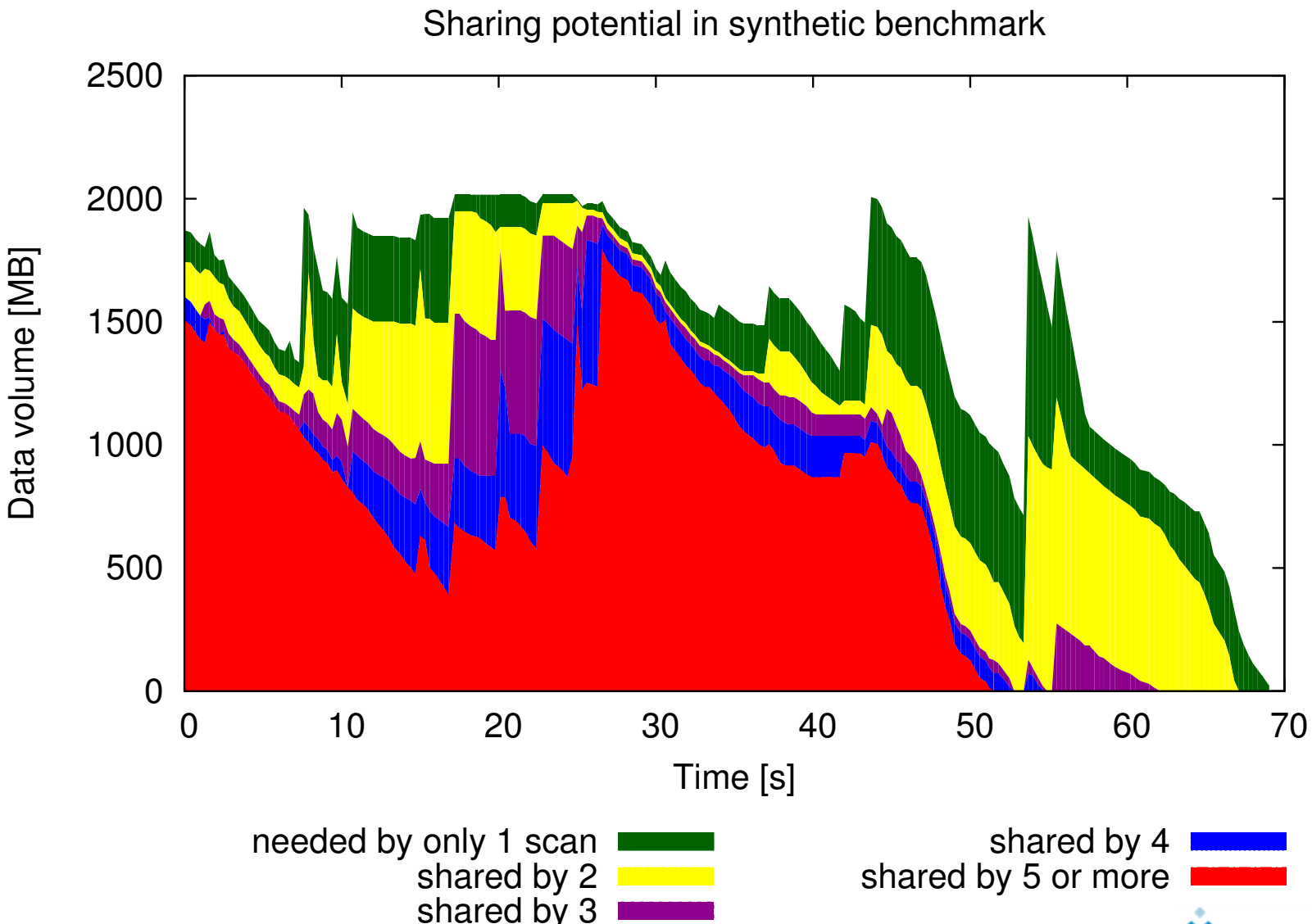
vectorwise

Benchmark – wyniki



vectorwise

Benchmark – sharing potential

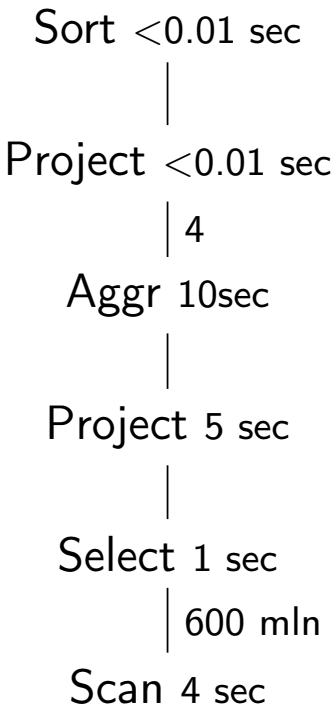


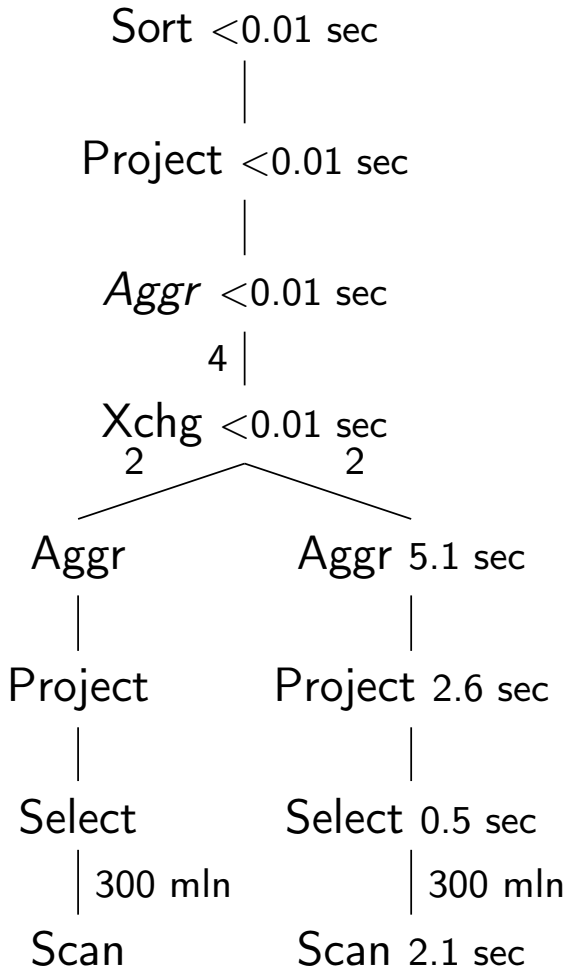
vectorwise

Multi-core Parallelization

vectorwise

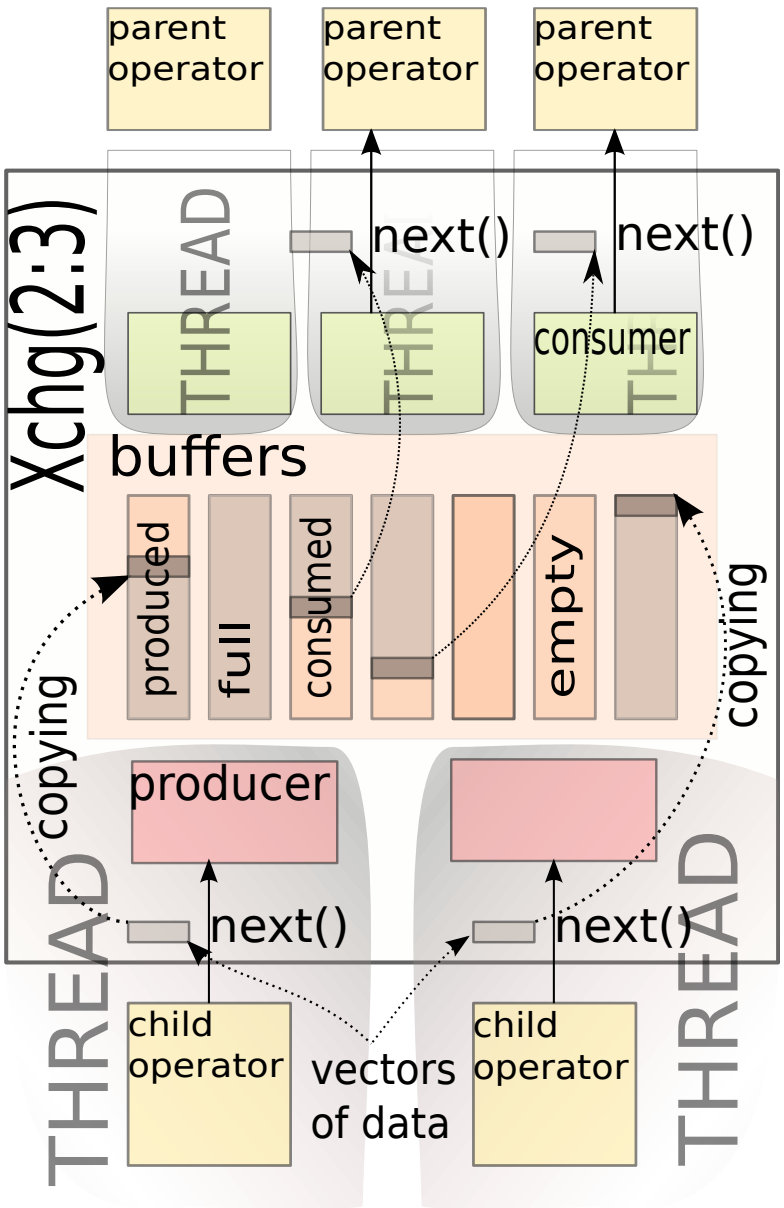
An example - TPC-H Query 1





vectorwise

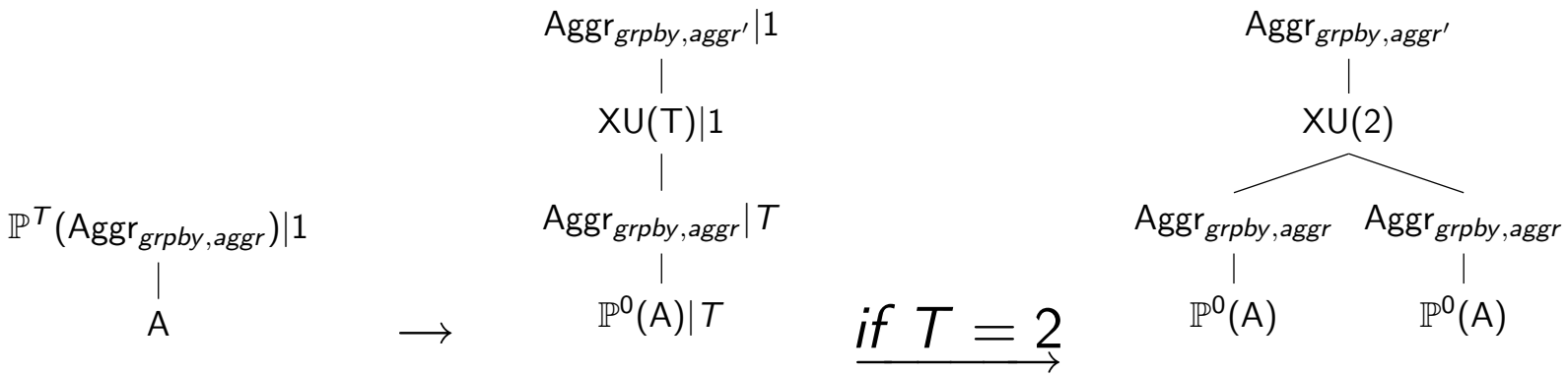
Xchg operator

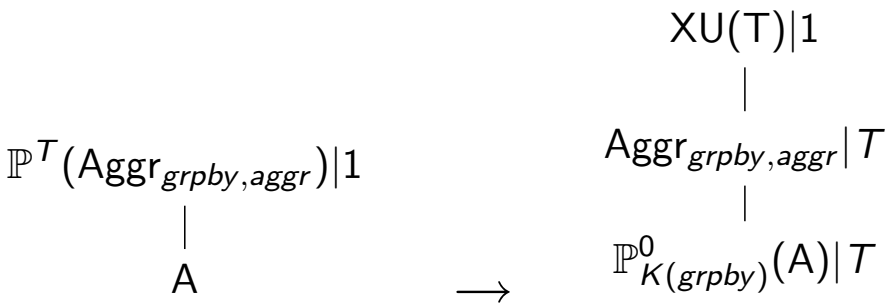


- producer-consumer schema
- producer copies the data from its child into buffers
- consumer passes the data from buffers to its parent
- a buffer changes its state from EMPTY to PRODUCED, FULL, CONSUMED and back to EMPTY.
- acquiring a buffer by producer or consumer is synchronized

vectorwise

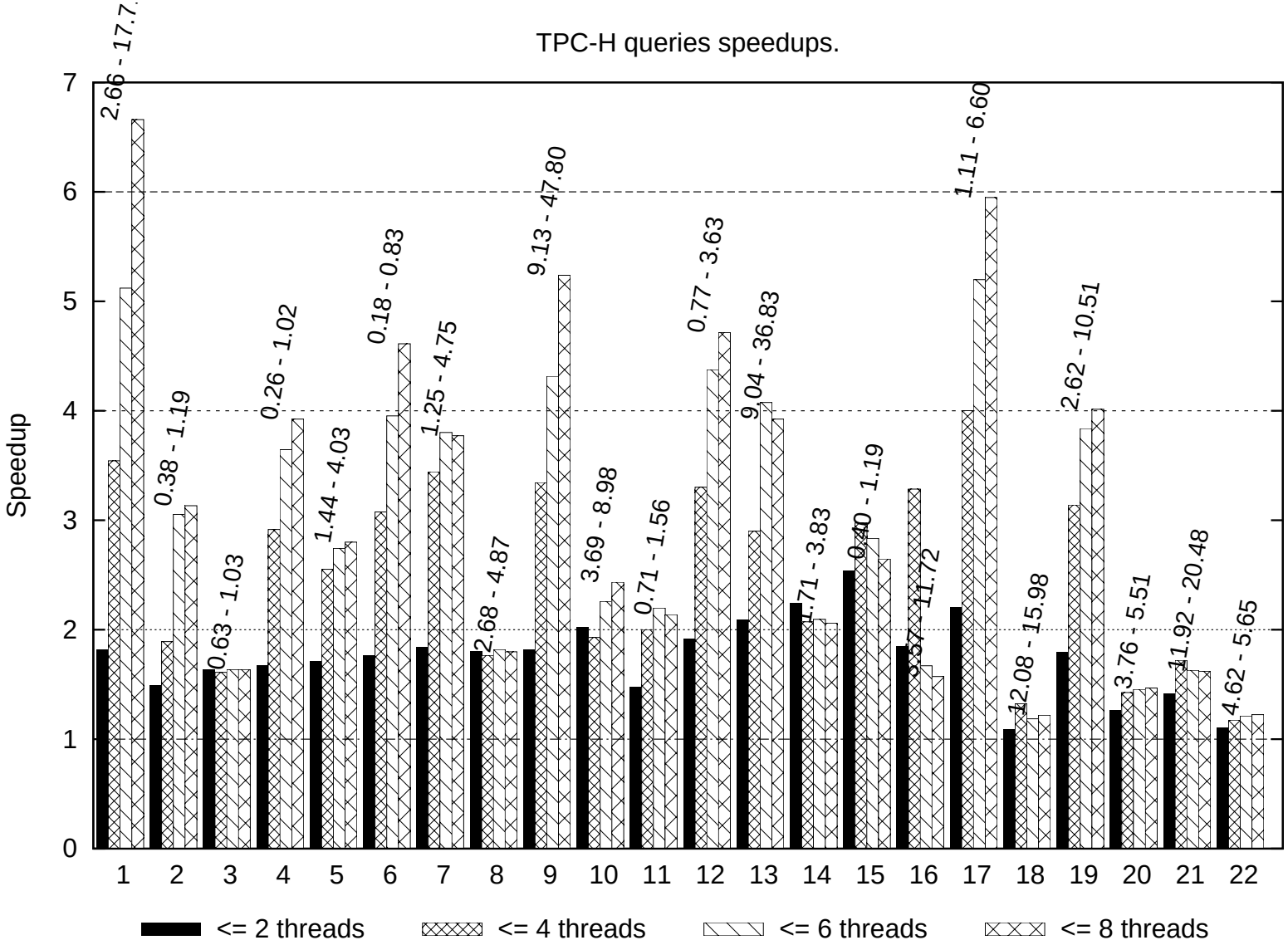
An example of transformations





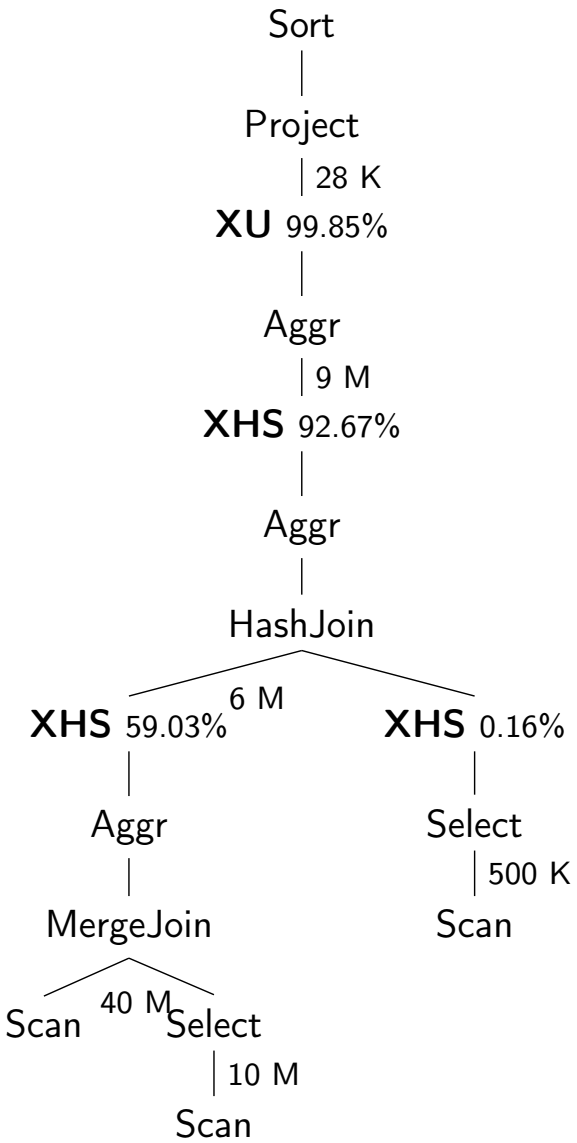
vectorwise

Operators



vectorwise

Operators



vectorwise

Propozycje projektów

vectorwise

Micro-adaptivity

Pewne funkcjonalności mogą być zaimplementowane na wiele różnych sposobów. Podobnie na wiele sposobów możemy dobierać stałe. Potrzebny jest moduł, który dynamicznie potrafi podejmować dobre decyzje.

- data vs. control dependency
- loop unrolling
- loop fussion/fission
- bloom filters

vectorwise

Szyfrowanie o przepustowości GB/s

- Niektórzy klienci mogą potrzebować szyfrowania by zabezpieczyć poufne dane.
- Architektura Nehalem wprowadza instrukcje wspierające szyfrowanie.
- Wyzwanie: zapewnić przepustowość rzędu GB/s
- Pytanie: Kiedy dokonywać deszyfrowania?

vectorwise

VectorWise na klastrze

VectorWise w tej chwili nie posiada wersji działającej na klastrach. Uniemożliwia to konkurencję z najszybszymi rozproszonymi bazami danych. Co jest potrzebne:

- Stworzenie rozproszonego systemu przechowywania danych
- Stworzenie modułu rozdzielającego zapytania na kilka węzłów
- Stworzenie operatorów łączących wyniki z kilku węzłów
- Stworzenie narzędzia do równoległego ładowania danych

vectorwise

Podsumowanie

- VectorWise udowadnia, że innowacyjne podejście do starego problemu może dać ciekawe wyniki
- Wciąż wiele pomysłów i możliwości rozwoju
- Zapraszamy stażystów

vectorwise

Dziękujemy za uwagę

Q & A

vectorwise

Bibliografia

- "Balancing Vectorized Query Execution with Bandwidth-Optimized Storage", Marcin Zukowski
- "Multi-core parallelization of vectorized query execution", Kamil Anikiej
- "Simple Solutions for Compressed Execution in Vectorized Database System", Alicja Łuszczak
- "Just-in-time Compilation in Vectorized Query Execution", Juliusz Sompolski
- "Integrating Cooperative Scans in a column-oriented DBMS", Michał Świtakowski

vectorwise