

Обработка и исполнение запросов в СУБД (Лекция 8)

Колоночные СУБД в оперативной памяти

v2

Георгий Чернышев

Академический Университет

chernishev@gmail.com

16 ноября 2016 г.

Долг: данные на диске

В двух словах (очень приблизительно):

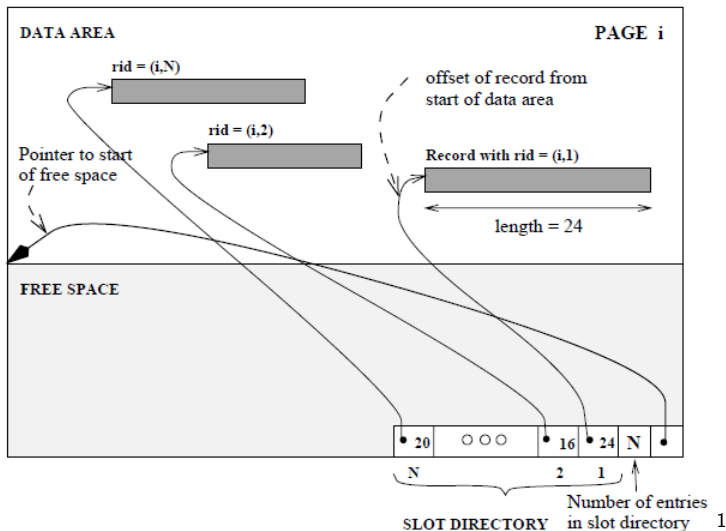
- Данных хранятся на диске;
- Считываются и временно хранятся в памяти;
- Обычно какой-то memory mapping;
- Менджер буферов (buffer-manager): считывание, удержание в памяти, модификация данных, сброс на диск;
 - pin/unpin;
 - алгоритм замещения: LRU, Clock algorithm, MRU, ...
- “Нарезка” на страницы: единицы оперирования менеджера буферов, 4-8 KB;

Кому интересно дальше:

https://web.stanford.edu/class/cs346/2015/notes/Lecture_One.pdf

Как представлять страницу? Записи ведь бывают переменной длины!

Долг: слотированная страница



¹Изображение взято из [Ramakrishnan and Gehrke, 2000]

Про “железо” “на пальцах” I

Elements:

1 Storage

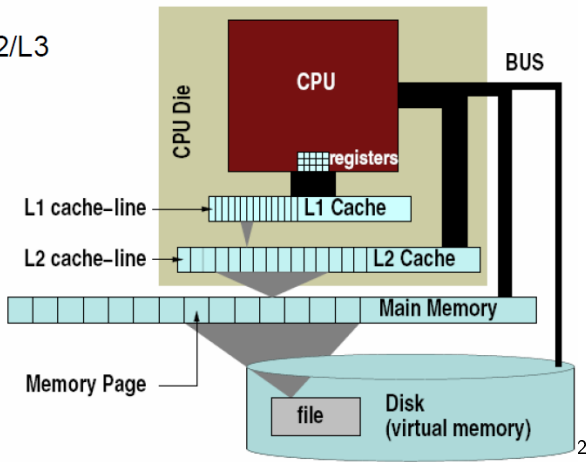
1 CPU caches L1/L2/L3

1 Registers

1 Execution Unit(s)

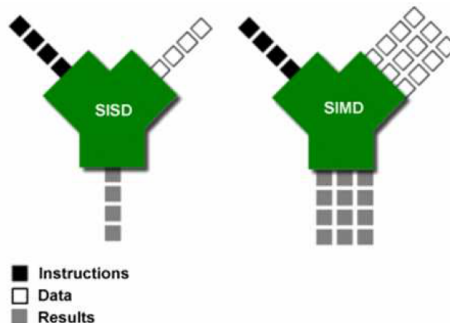
1 Pipelined

1 SIMD



²Изображение взято из [Harizopoulos et al., 2009]

SIMD



1 Single Instruction Multiple Data

- 1 Same operation applied on a vector of values
- 1 MMX: 64 bits, SSE: 128bits, AVX: 256bits
- 1 SSE, e.g. multiply 8 short integers

3

³ Изображение взято из [Harizopoulos et al., 2009]

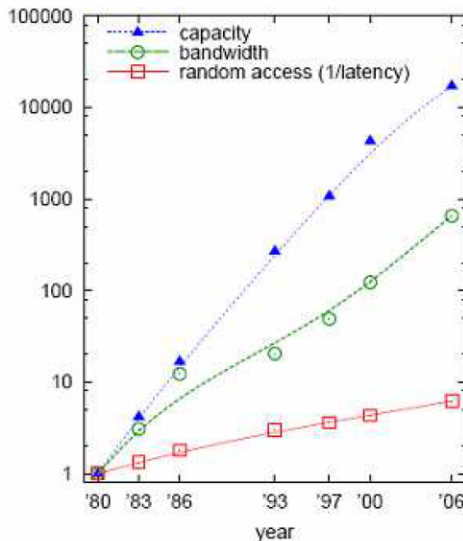
Про “железо” “на пальцах” III (эволюция процессоров)

| Microprocessor | 16-bit address/ bus, microcoded | 32-bit address/ bus, microcoded | 5-stage pipeline, on-chip I & D caches, FPU | 2-way superscalar, 64-bit bus | Out-of-order 3-way superscalar | Out-of-order superpipelined, on-chip L2 cache | Multicore OOO 4-way on chip L3 cache, Turbo |
|-----------------------------|---------------------------------------|---------------------------------------|--|----------------------------------|--------------------------------|--|--|
| Product | Intel 80286 | Intel 80386 | Intel 80486 | Intel Pentium | Intel Pentium Pro | Intel Pentium 4 | Intel Core i7 |
| Year | 1982 | 1985 | 1989 | 1993 | 1997 | 2001 | 2010 |
| Die size (mm ²) | 47 | 43 | 81 | 90 | 308 | 217 | 240 |
| Transistors | 134,000 | 275,000 | 1,200,000 | 3,100,000 | 5,500,000 | 42,000,000 | 1,170,000,000 |
| Processors/chip | 1 | 1 | 1 | 1 | 1 | 1 | 4 |
| Pins | 68 | 132 | 168 | 273 | 387 | 423 | 1366 |
| Latency (clocks) | 6 | 5 | 5 | 5 | 10 | 22 | 14 |
| Bus width (bits) | 16 | 32 | 32 | 64 | 64 | 64 | 196 |
| Clock rate (MHz) | 12.5 | 16 | 25 | 66 | 200 | 1500 | 3333 |
| Bandwidth (MIPS) | 2 | 6 | 25 | 132 | 600 | 4500 | 50,000 |
| Latency (ns) | 320 | 313 | 200 | 76 | 50 | 15 | 4 |

4

⁴ Изображение взято из [Hennessy and Patterson, 2011]

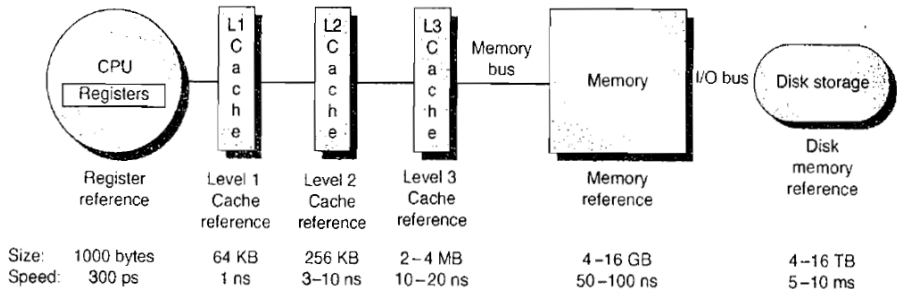
Про “железо” “на пальцах” IV (25 лет эволюции RAM)



5

⁵ Изображение взято из [Harizopoulos et al., 2009]

Про “железо” “на пальцах” V (иерархия носителей)



(a) Memory hierarchy for server

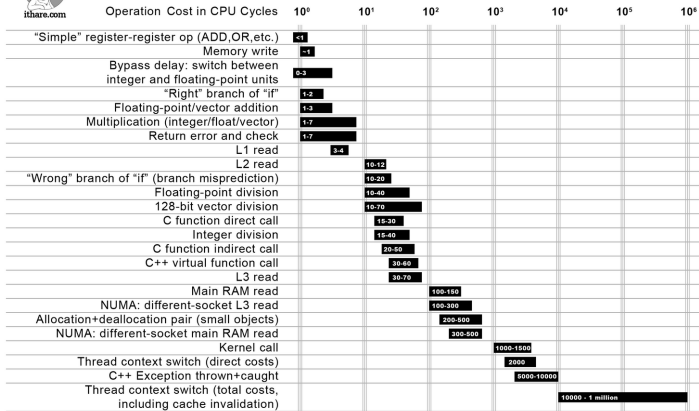
6

⁶ Изображение взято из [Hennessy and Patterson, 2011]

Про “железо” “на пальцах” VI (стоимость операций)



Not all CPU operations are created equal



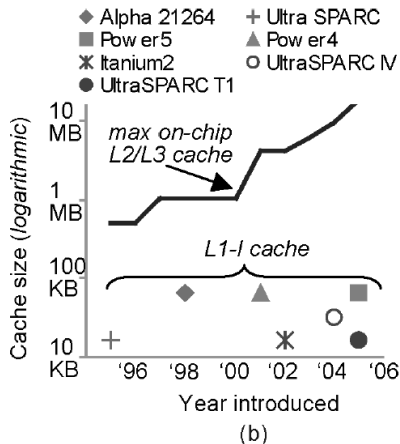
Distance which light travels while the operation is performed



7 Изображение взято из http://ithare.com/wp-content/uploads/part101_infographics_v07.png

Бывает:

- Кеш данных — хранит данные, которыми оперирует программа,
- Кеш инструкций — хранит саму программу.



8

Как выглядит код в кеш-памяти

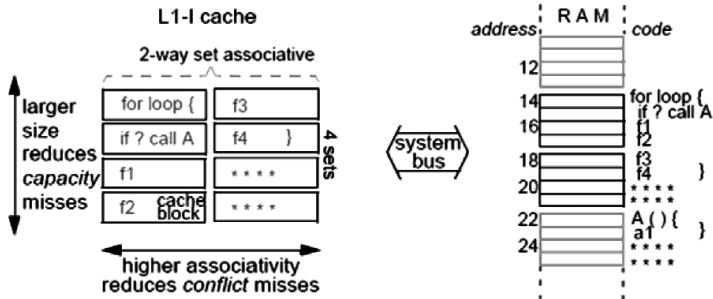
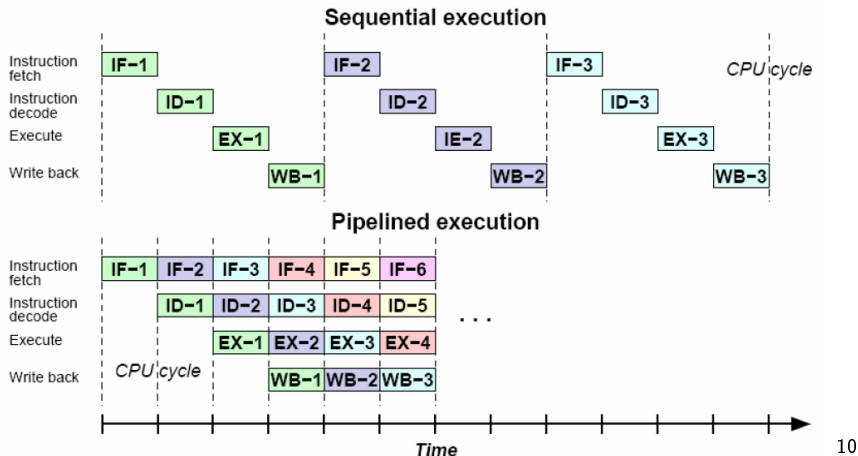


Fig. 1. Example of a two-way set associative, four-set (eight cache blocks) L1-I cache. Code stored in RAM maps to one set of cache blocks and is stored to any of the two blocks in that set. For simplicity, we omit L2/L3 caches. In this example, the *for-loop* code fits in the cache only if procedure A is never called. In that case, repeated executions of the code will always hit in the L1-I cache. Larger code (more than eight blocks) would result in *capacity* misses. On the other hand, frequent calls to A would result to *conflict* misses because A's code would replace code lines f3 and f4 needed in the next iteration.

- L1 кеш 8-64KB (и данные, и инструкции), больше не будет:
 - придется снижать частоту, температура
- В OLTP код занимает 556 KB, смотри ссылку из [Harizopoulos and Ailamaki, 2006].

То есть, для достижения хорошей производительности надо писать “правильный” код.

Про “железо” “на пальцах” VII: конвейер



Кому непонятно, читайте https://en.wikipedia.org/wiki/Instruction_pipelining.

¹⁰ Изображение взято из [Harizopoulos et al., 2009]

Про “железо” “на пальцах” VIII: проблемы выполнения

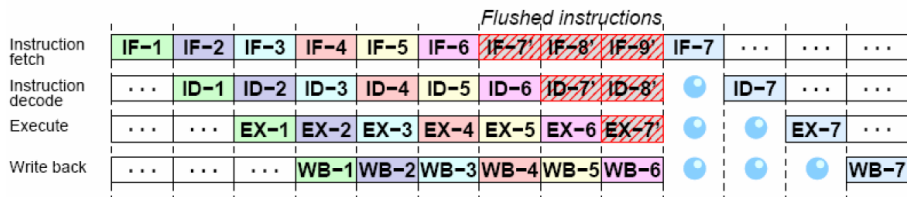
1 Data hazards

- 1 Dependencies between instructions
- 1 L1 data cache misses

1 Control Hazards

- 1 Branch mispredictions
- 1 Computed branches (late binding)
- 1 L1 instruction cache misses

Result: bubbles in the pipeline



Out-of-order execution addresses data hazards

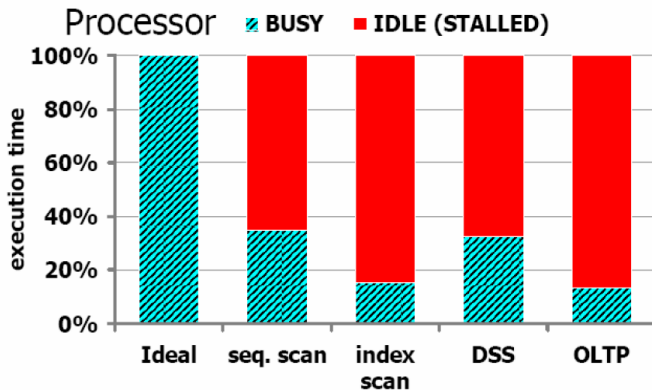
- 1 control hazards typically more expensive

11

¹¹Изображение взято из [Harizopoulos et al., 2009]

Как себя ведет процессор под нагрузкой (итераторами) I

DB workload execution on a modern computer



“DBMSs On A Modern Processor: Where Does Time Go? ”
Ailamaki, DeWitt, Hill, Wood, VLDB'99

12

Как себя ведет процессор под нагрузкой (итераторами) II

Вкратце из статьи [Boncz et al., 2005]:

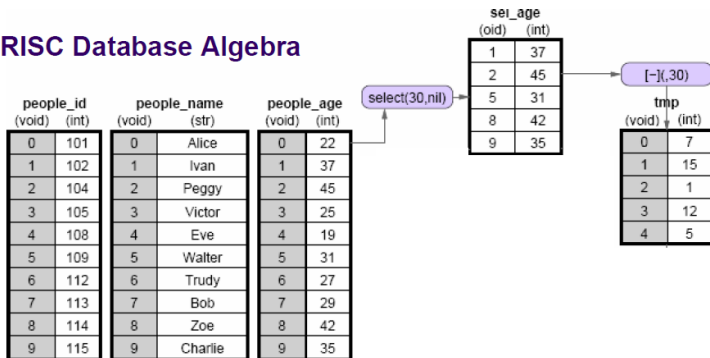
- TPC-H 1GB, Q1 (выборки, агрегация)
- Итог:
 - C program: 0.2s
 - MySQL: 26.2s
 - DBMS “X”: 28.1s

Вкратце из статей, например [Boncz et al., 2008]:

- Все данные в памяти, mmap-ом отображаются на диск, если виртуальной памяти не хватит, используем за-mmap-ленную дисковую, есть подсистема;
- Данные представлены в виде BAT структур (Binary Association Table): (*key*, *value*);
- Над BAT есть своя алгебра, каждый оператор очень простой;
- Фактически, операции над массивом;

Этот подход позволяет очень эффективно использовать процессор и оперативную память.

RISC Database Algebra



```

SELECT  id, name, (age-30)*50 as bonus
FROM    people
WHERE   age > 30

```

13

Как такой подход помогает?

Вот так [Boncz et al., 2008, Harizopoulos et al., 2009]:

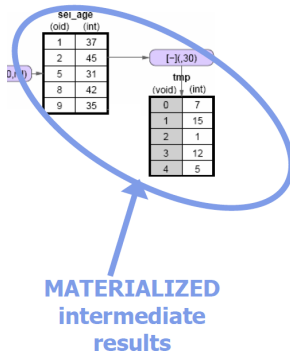
- “Хороший” с точки зрения CPU код:
 - Меньше зависимостей по данным;
 - Меньше зависимостей по управлению;
 - Можно автоматически генерировать SIMD-код.
- Один цикл для целой колонки:
 - Отказ от работы на уровне записи;
 - Массивы позволяют переходить по смещениям;
 - Лучшая ситуация с кешем инструкций.

```
1 {  
2   for(i=0; i<n; i++)  
3     res[i] = col[i] - val;  
4 }
```

Итог: Monet около 4s (см 16 слайд).

Материализация результата

[Boncz et al., 2008, Harizopoulos et al., 2009]:

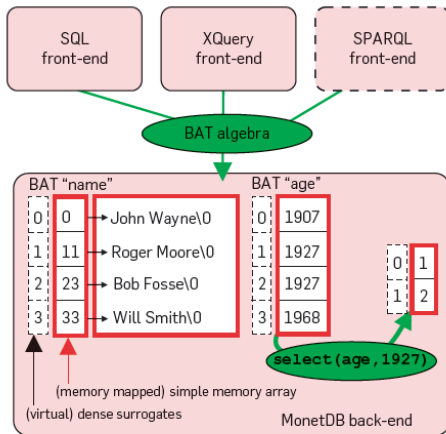


14

Памяти просто может не хватить, не будет масштабируемости.

¹⁴ Изображение взято из [Harizopoulos et al., 2009]

Figure 2: MonetDB: a BAT algebra machine.



15

Ниша: исследовательская платформа II

1 Cache-Conscious Joins

- 1 Cost Models, Radix-cluster
Radix-decluster

- “Database Architecture Optimized for the New Bottleneck: Memory Access” VLDB'99
- “Generic Database Cost Models for Hierarchical Memory Systems”, VLDB'02 (all Manegold, Boncz, Kersten)
- “Cache-Conscious Radix-Declude Projections”, Manegold, Boncz, Nes, VLDB'04

1 MonetDB/XQuery:

- 1 structural joins exploiting
positional column access

“MonetDB/XQuery: a fast XQuery processor powered by a relational engine” Boncz, Grust, vanKeulen, Rittinger, Teubner, SIGMOD'06

1 Cracking:

- 1 on-the-fly automatic indexing
without workload knowledge

“Database Cracking”, CIDR'07
“Updating a cracked database “, SIGMOD'07
“Self-organizing tuple reconstruction in column-stores”, SIGMOD'09 (all Idreos, Manegold, Kersten)

1 Recycling:

- 1 using materialized intermediates

“An architecture for recycling intermediates in a column-store”, Ivanova, Kersten, Nes, Goncalves, SIGMOD'09

1 Run-time Query Optimization:

- 1 correlation-aware run-time
optimization without cost model

“ROX: run-time optimization of XQueries”, Abdelkader, Boncz, Manegold, vanKeulen, SIGMOD'09

16

Как работает [Shatdal et al., 1994], существующий Grace Hash-Join вариант:

- Оба отношения фрагментируем (по хеш-коду атрибута соединения) на N кластеров;
- Каждый кластер помещается в L2 кеш;
- Для каждого кластера “прикладываем” элементы соответствующего фрагмента из второго отношения.

Кластеризация это проблема:

- Алгоритм имеет один скан и кластеризацию, пишет в N случайных регионов (кластеров), это портит шаблон доступа к памяти;
- Если N слишком большой, то записей в TLB может не хватить, будет TLB miss;
- Если N слишком большой то линейек в кеше может не хватить, будет засорение кеша и L1, L2 cache miss;

→ придумали radix-cluster алгоритм

Про “железо” “на пальцах” IX: TLB

Translation Lookaside Buffer (TLB):

- Тоже “кеш-память”, встроена в CPU, обеспечивает работу виртуальной памяти;
- “Помнит” последние трансляции логических в физические адреса (обычно 64-96 штук);
- Адресует страницы в оперативной памяти:
 - на каждый load/store нужна одна трансляция
 - нашлась — получили адрес сразу же
 - не нашлась: TLB miss, идем в RAM смотрим в TLB таблицу (тут тоже может быть цепочка из cache miss)
- При 4KB странице, 64 записи, можно встретить TLB miss при случайном доступе к структурам (хеш-таблицам, например) размером больше чем 256KB.

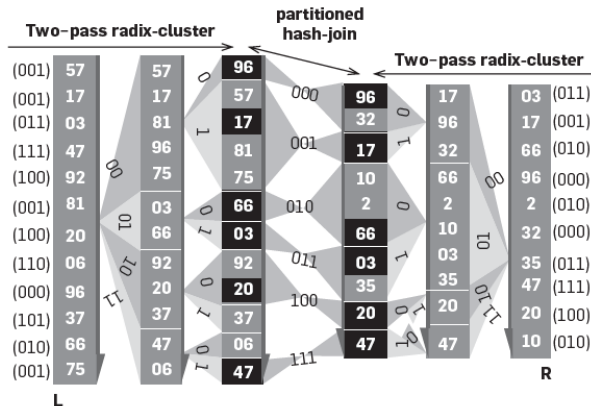
Сейчас всё сложнее, например на Sandy Bridge отдельные TLB для разных размеров¹⁷.

¹⁷[https://stackoverflow.com/questions/40649655/](https://stackoverflow.com/questions/40649655/how-is-the-size-of-tlb-in-intels-sandy-bridge-cpu-determined)

[how-is-the-size-of-tlb-in-intels-sandy-bridge-cpu-determined](https://stackoverflow.com/questions/40649655/how-is-the-size-of-tlb-in-intels-sandy-bridge-cpu-determined)

Предложенное решение: radix cluster I

Figure 3: Partitioned hash-join ($H = 8 \leftrightarrow B = 3$).

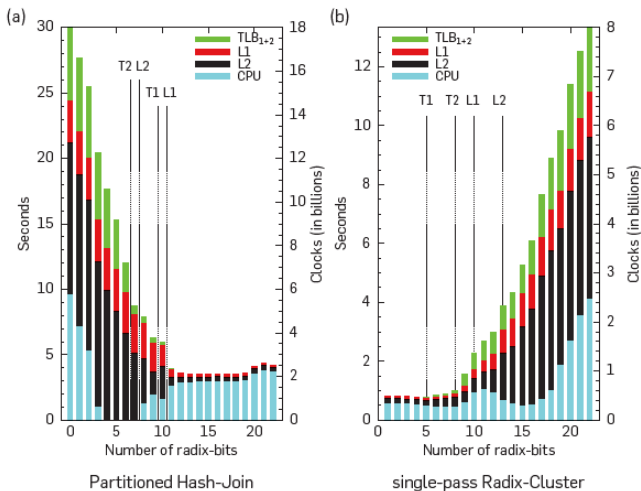


Black tuples hit (lowest 3-bits of values in parenthesis)

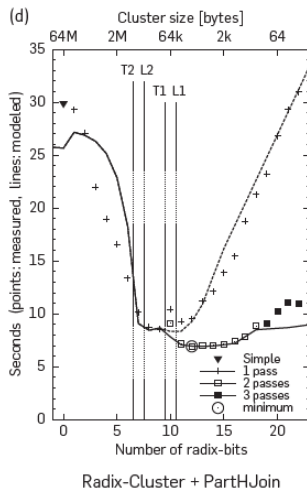
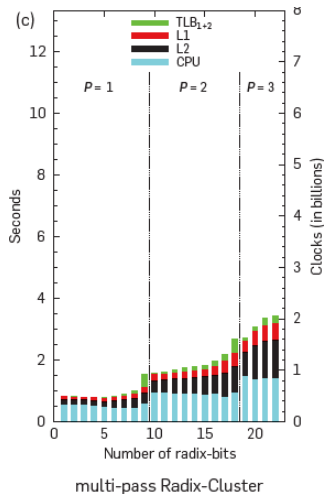
18

Предложенное решение: radix cluster II

- Перестановка происходит только в рамках блока, лучше паттерн доступа, меньше вероятность промахов;
- Обычно хешируют, и работают с хеш-кодами, хотя на рисунке это не показано;
- Если колонка начинается с 0 и плотна, можно не хешировать, будет radix-sort.



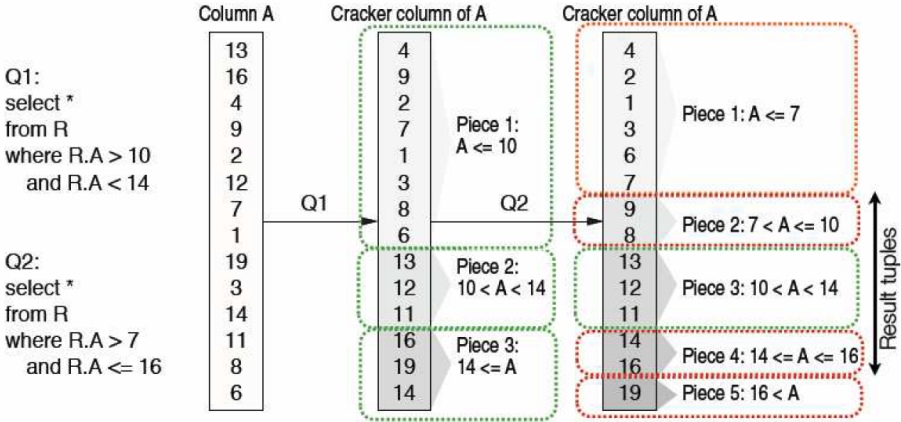
19



20

Database Cracking: идея

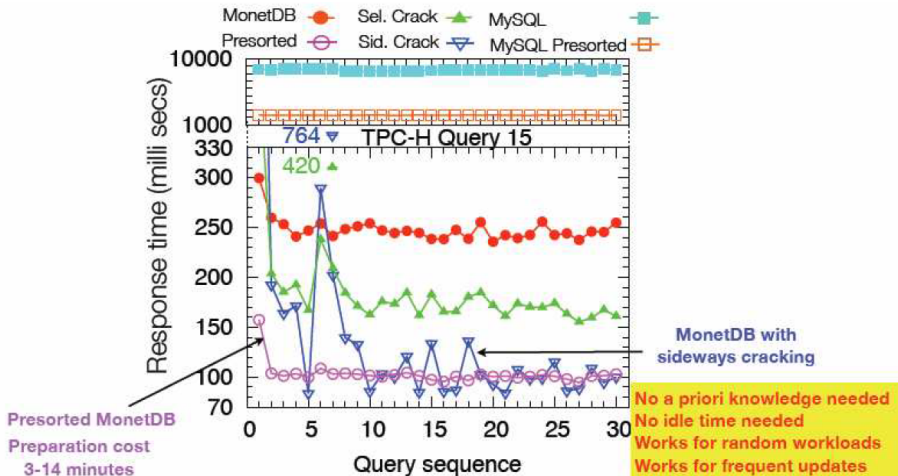
Physically reorganize based on the selection predicate



²¹Изображение взято из [Harizopoulos et al., 2009]

- Выше было описано как применять к колонкам при запросах на выборку (selection cracking);
- Кроме того, можно заменить восстановление записи на cracking (например, вспомните прошлую лекцию, место про соединение и нарушение порядка) — sideways cracking.

Database Cracking: эксперименты



22

22 Изображение взято из [Harizopoulos et al., 2009]

Recycler I: дерево запроса

```
function user.s1_2(A0:date,A1:date,A2:int,A3:str):void;  
  X5 := sql.bind("sys","lineitem","l_returnflag",0);  
  X11 := algebra.useselect(X5,A3);  
  X14 := algebra.markT(X11,0@0);  
  X15 := bat.reverse(X14);  
  X16 := sql.bindldxbat("sys","lineitem","li_fkey");  
  X18 := algebra.join(X15,X16);  
  X19 := sql.bind("sys","orders","o_orderdate",0);  
  X25 := mtime.addmonths(A1,A2);  
  X26 := algebra.select(X19,A0,X25,true,false);  
  X30 := algebra.markT(X26,0@0);  
  X31 := bat.reverse(X30);  
  X32 := sql.bind("sys","orders","o_orderkey",0);  
  X34 := bat.mirror(X32);  
  X35 := algebra.join(X31,X34);  
  X36 := bat.reverse(X35);  
  X37 := algebra.join(X18,X36);  
  X38 := bat.reverse(X37);  
  X40 := algebra.markT(X38,0@0);  
  X41 := bat.reverse(X40);  
  X45 := algebra.join(X31,X41);  
  X46 := algebra.join(X41,X45);  
  X49 := algebra.selectNotNil(X46);  
  X50 := bat.reverse(X49);  
  X51 := algebra.kunique(X50);  
  X52 := bat.reverse(X51);  
  X53 := aggr.count(X52);  
  sql.exportValue(1,"sys.orders","L1","wrd",32,0,6,X53);  
end s1_2;
```

Fig. 1. MAL plan of the example query

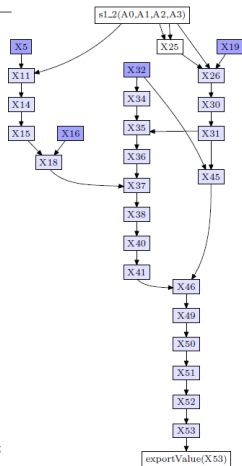


Fig. 2. Execution plan marked by the recycler optimiser

23

- Полная материализация промежуточных результатов — можно переиспользовать некоторые из них:
 - Локально: в рамках одного плана + в рамках шаблона;
 - Глобально: в рамках нескольких параллельно идущих запросов;
- Кеш с политикой допуска:
 - KEEPALL — храним всё;
 - CREDIT — экономические принципы;
- ...и с политикой вытеснения:
 - LRU;
 - Benefit policy;
 - History policy.

Recycler III: сколько можно сэкономить?

| Query | Instructions | | | Time (s) | | | |
|-------|--------------|------------|------------|----------|---------|-------|-------|
| | # | Intra % | Inter % | Total | Savings | | |
| | | | | | Pot. | Local | Glob. |
| Q1 | 36 | 2.8 | 0 | 5.72 | 3.54 | 0.30 | 0 |
| Q2 | 106 | 0.9 | 2.8 | 0.22 | 0.22 | 0 | 0.07 |
| Q3 | 39 | 0 | 5.1 | 2.61 | 2.40 | 0 | 0 |
| Q4 | 36 | 0 | 41.7 | 1.72 | 1.65 | 0 | 1.44 |
| Q5 | 74 | 0 | 2.7 | 1.16 | 1.15 | 0 | 0 |
| Q6 | 11 | 0 | 0 | 0.53 | 0.52 | 0 | 0 |
| Q7 | 106 | 3.8 | 3.8 | 1.61 | 1.11 | 0.36 | 0.56 |
| Q8 | 61 | 0 | 6.6 | 0.60 | 0.56 | 0 | 0.16 |
| Q9 | 59 | 0 | 3.4 | 1.38 | 1.25 | 0 | 0 |
| Q10 | 54 | 0 | 3.7 | 1.37 | 1.34 | 0 | 0.20 |
| Q11 | 36 | 33.3 | 2.8 | 0.16 | 0.16 | 0.03 | 0 |
| Q12 | 6 | 0 | 33.3 | 1.17 | 0.55 | 0 | 0 |
| Q13 | 17 | 0 | 11.8 | 2.88 | 1.27 | 0 | 0 |
| Q14 | 18 | 0 | 0 | 0.21 | 0.21 | 0 | 0 |
| Q15 | 12 | 0 | 0 | 0.23 | 0.19 | 0 | 0 |
| Q16 | 14 | 0 | 42.9 | 0.88 | 0.27 | 0 | 0.01 |
| Q17 | 29 | 0 | 3.4 | 0.96 | 0.95 | 0 | 0 |
| Q18 | 12 | 0 | 75.0 | 1.83 | 1.70 | 0 | 1.68 |
| Q19 | 39 | 15.4 | 7.7 | 3.72 | 1.69 | 0.99 | 0.49 |
| Q20 | 25 | 0 | 12.0 | 0.95 | 0.82 | 0 | 0.01 |
| Q21 | 154 | 9.1 | 12.3 | 5.80 | 5.38 | 0.72 | 2.94 |
| Q22 | 4 | 0 | 75.0 | 0.65 | 0.15 | 0 | 0.15 |

Table II. Characteristics of TCP-H queries

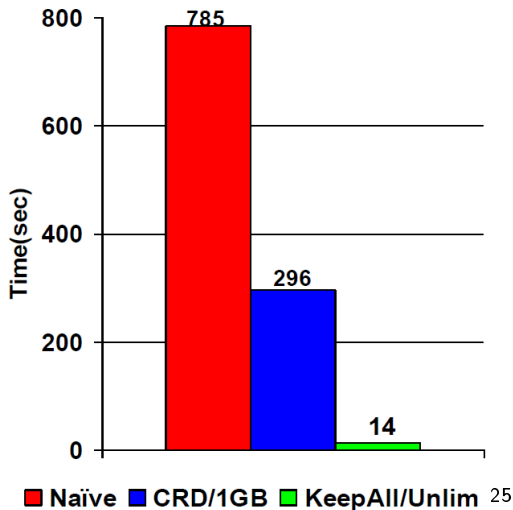
24

Recycler VI: тест на реальной базе

Sloan Digital Sky Survey/
SkyServer

<http://cas.sdss.org>

- 1 100 GB subset of DR4
- 1 100-query batch from January 2008 log
- 1 1.5GB intermediates, 99% reuse
- 1 Join intermediates major consumer of memory and major contributor to savings

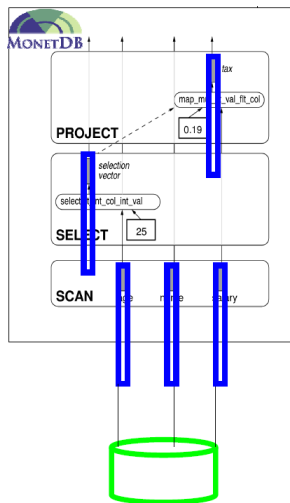
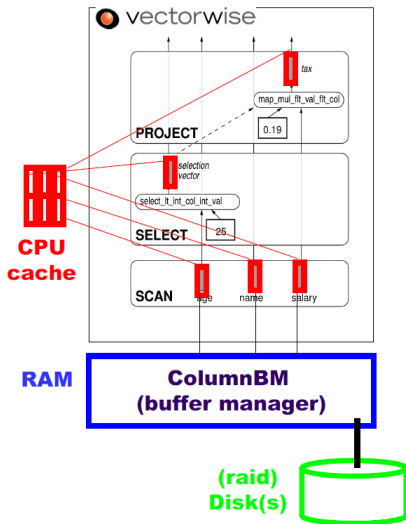


²⁵ Изображение взято из [Ivanova et al., 2010]

- Избавились от полной материализации и добавили pipelining;
- Сделали “векторизацию”: в кеше массив из ста элементов;
 - Все вектора должны помещаться в кеш!
- Такие же простые операторы цикла для обработки;
- Поздняя материализация + selection vectors: `col[sel[i]]`;

Итог: MonetDB/X100: 0.6s (слайд 16)

MonetDB/X100: архитектура



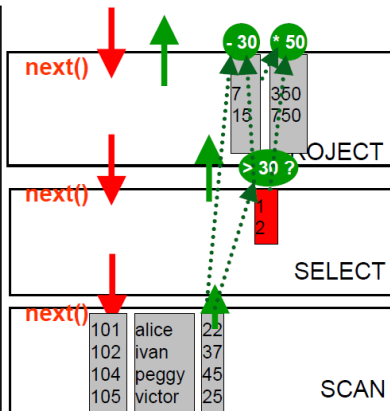
26

Пояснение про selection vectors

```
map_mulflt_valflt_col(  
    float *res,  
    int* sel,  
    float val,  
    float *col, int n)  
{  
    for(int i=0; i<n; i++)  
        res[i] = val * col[sel[i]];  
}
```

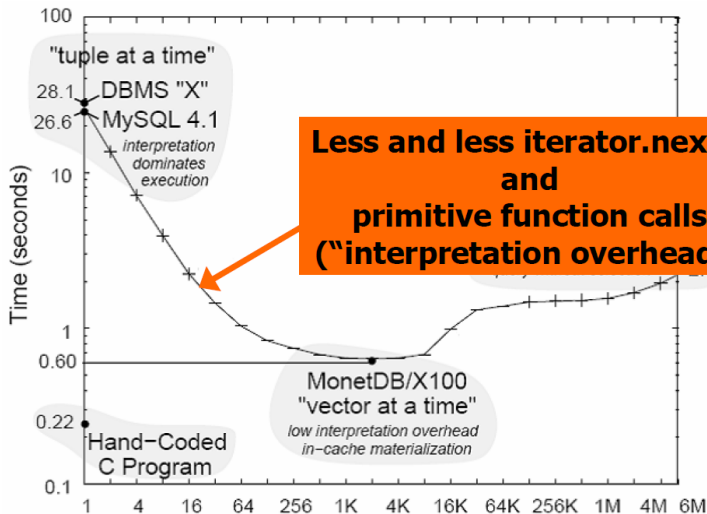
selection vectors used to reduce
vector copying

contain selected positions



“MonetDB/X100: Hyper-Pipelining Query Execution” Boncz, Zukowski, Nes, CIDR’05

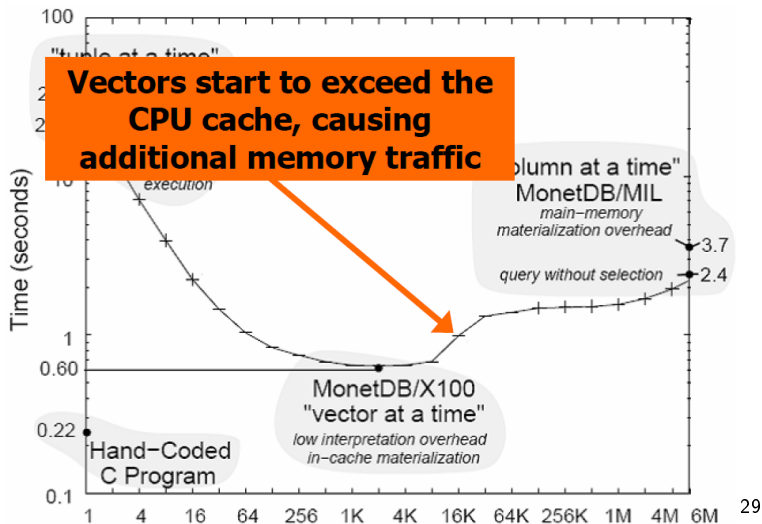
MonetDB/X100: размер вектора |



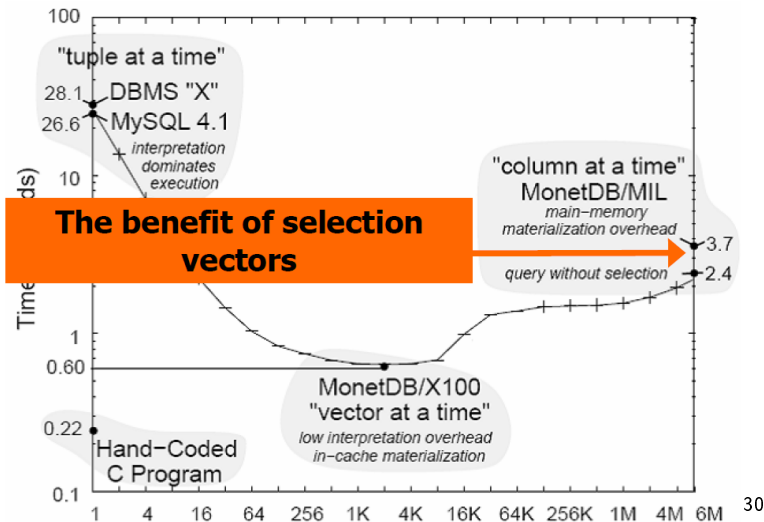
28

28 Изображение взято из [Harizopoulos et al., 2009]

MonetDB/X100: размер вектора II



MonetDB/X100: размер вектора III



30 Изображение взято из [Harizopoulos et al., 2009]



Milena G. Ivanova, Martin L. Kersten, Niels J. Nes, and Romulo A.P. Gonçalves. 2010. An architecture for recycling intermediates in a column-store. ACM Trans. Database Syst. 35, 4, Article 24 (October 2010), 43 pages.
DOI=10.1145/1862919.1862921 <http://doi.acm.org/10.1145/1862919.1862921>



Ambuj Shatdal, Chander Kant, and Jeffrey F. Naughton. 1994. Cache Conscious Algorithms for Relational Query Processing. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 510–521.



Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. 2008. Breaking the memory wall in MonetDB. Commun. ACM 51, 12 (December 2008), 77–85.
DOI=<http://dx.doi.org/10.1145/1409360.1409380>



Peter Boncz, Marcin Zukowski, Niels Nes. MonetDB/X100: Hyper-Pipelining Query Execution. CIDR'05.



John L. Hennessy and David A. Patterson. 2011. Computer Architecture, Fifth Edition: A Quantitative Approach (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.



Stavros Harizopoulos and Anastassia Ailamaki. 2006. Improving instruction cache performance in OLTP. ACM Trans. Database Syst. 31, 3 (September 2006), 887-920. DOI=<http://dx.doi.org/10.1145/1166074.1166079>



Daniel Abadi, Peter Boncz, Stavros Harizopoulos. The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends(R) in Databases Vol. 5, No. 3 (2012) 197–280



Stavros Harizopoulos, Daniel Abadi, Peter Boncz. Column-Oriented Database Systems. VLDB 2009 Tutorial (slides).



Raghu Ramakrishnan and Johannes Gehrke. 2000. Database Management Systems (2nd ed.). Osborne/McGraw-Hill, Berkeley, CA, USA.