Ускорение исполнения запросов в PostgreSQL с использованием JIT-компилятора LLVM

Дмитрий Мельник
dm@ispras.ru

Институт системного программирования
Российской академии наук
(ИСП РАН)

5 февраля 2016

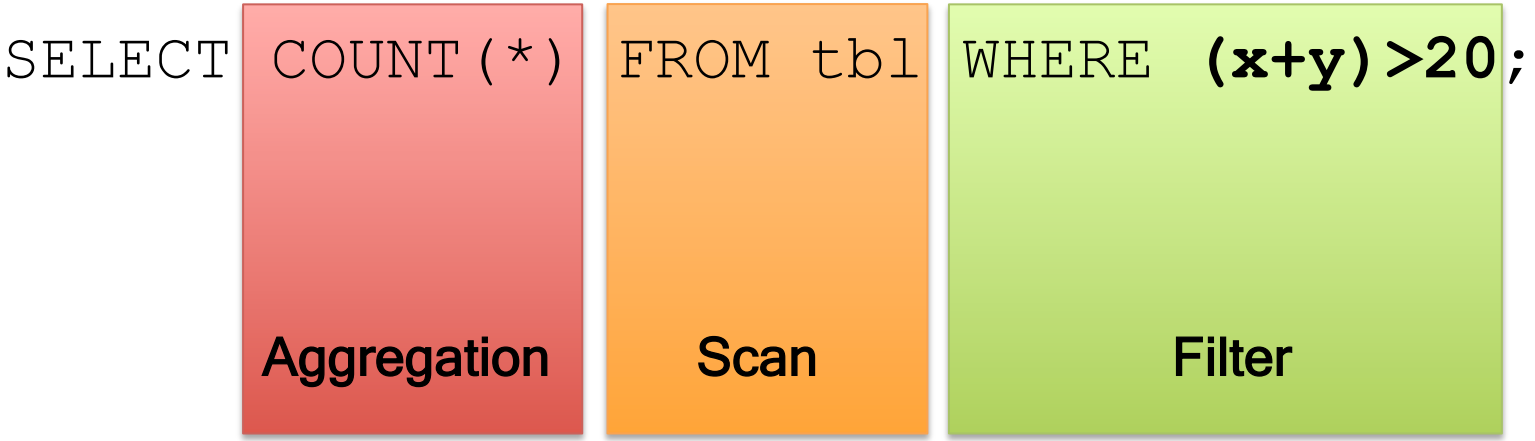
Ускорение PostgreSQL

ИСП РАН

- Что именно мы хотим ускорить?
  - Сложные запросы, где «узким местом» в производительности является процессор, а не дисковые операции
    - OLAP, поддержка принятия решений, и т.д.
  - Цель: оптимизация производительности на наборе тестов TPC-H
- Как ускорить?
  - Использовать LLVM JIT, на первом этапе – для компиляции выражений в операторе WHERE

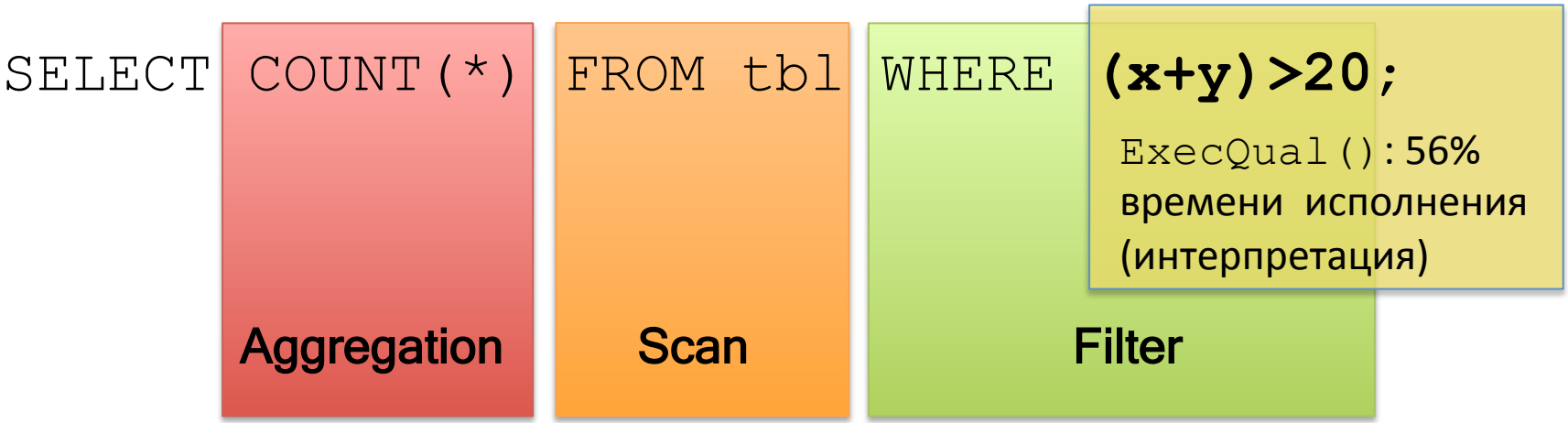
Пример оптимизации запроса

ИСП РАН



Пример оптимизации запроса

ИСП РАН



LLVM

ИСП РАН

- LLVM (Low Level Virtual Machine) – компиляторная инфраструктура для компиляции и оптимизации программ
  - Платформенно-независимое внутреннее представление (LLVM bitcode)
  - Широкий набор оптимизаций
  - Кодогенерация под популярные платформы (x86, x86_64, ARM, MIPS, …)
  - Подходит для построения JIT-компиляторов: динамическая библиотека с API для генерации биткода, оптимизации и кодогенерации
  - Лицензия: UIUC (permissive BSD-like)
  - Сравнительно простой код, легко разобраться

Использование LLVM JIT – популярный тренд

ИСП РАН

- Pyston (Python, Dropbox)
- HHVM (PHP & Hack, Facebook)
- LLILC (MSIL, .NET Foundation)
- Julia (Julia, community)

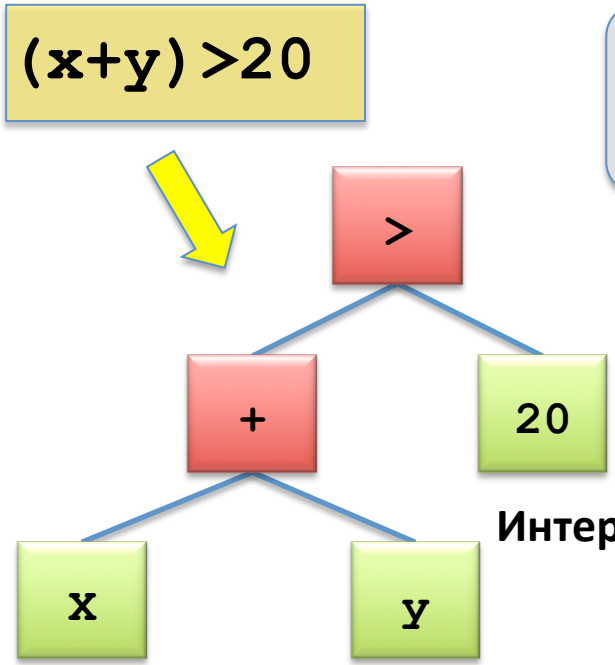
- JavaScript:
  - JavaScriptCore в WebKit (JavaScript, Apple)
  - LLV8 – добавление LLVM в качестве дополнительного уровня JIT в Google V8 (JavaScript, ISP RAS)

Что получится, если добавить к Postgres LLVM JIT?

ИСП РАН

Вычисление выражений

ИСП РАН



Генератор LLVM-биткода

Биткод LLVM

```
define i32 @where_expr(i32 %x, i32 %y) #0 {
entry:
  %add = add nsw i32 %y, %x
  %cmp = icmp sgt i32 %add, 20
  %conv = zext i1 %cmp to i32
  ret i32 %conv
}
```

LLVM MCJIT

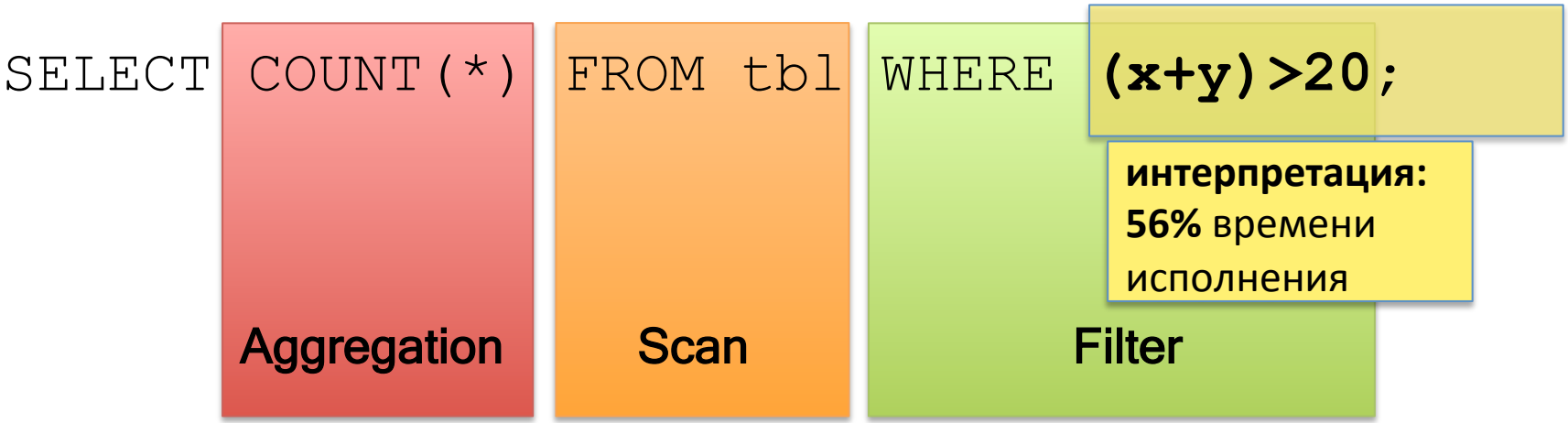
Интерпретация vs **Оптимизированный двоичный код**

```
foo:
        addl    %esi, %edi
        cmpl    $20, %edi
        setg    %al
        movzbl  %al, %eax
        retq
```

в ~10 раз быстрее

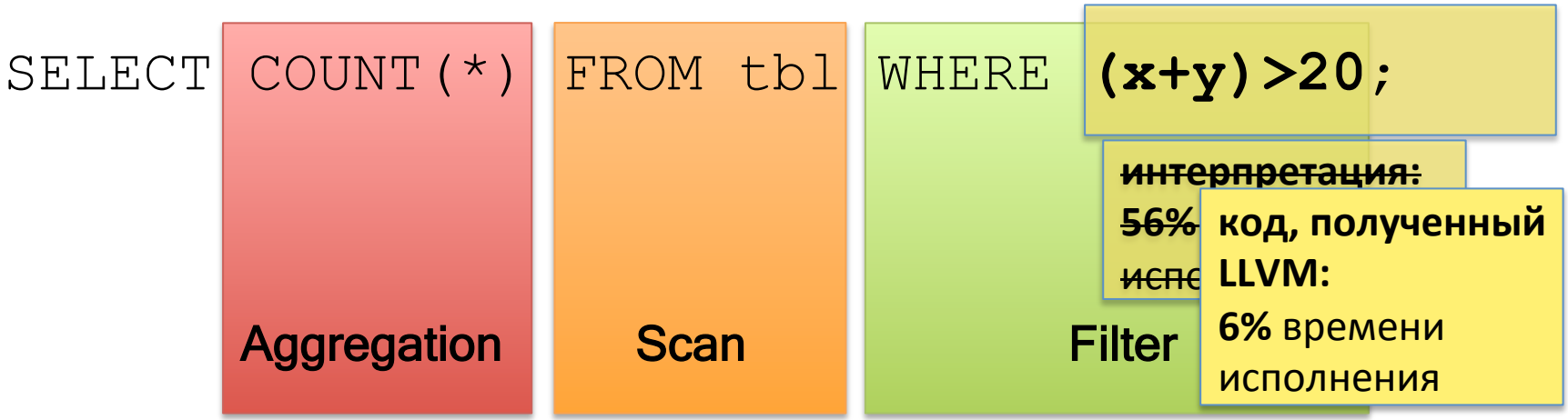
Пример оптимизации запроса

ИСП РАН



Пример оптимизации запроса

ИСП РАН



=> Ускорение выполнения запроса в 2 раза

Related work (1)

ИСП РАН

1. T. Neuman. Efficiently Compiling Efficient Query Plans for Modern Hardware. Proceedings of the VLDB Endowment, Vol. 4, No. 9, 2011.
2. PGStrom
   - Расширение для PostgreSQL (Custom scan)
   - Выполнение запросов на GPU (Cuda)
3. Axle Project
   - OpenCL

Related work (2)

ИСП РАН

4. Vitesse DB
   - Коммерческое, проприетарное ПО на основе PostgreSQL
   - Реализована часть идей из [1]:
     - Aggregation, Scan и Filter выполнены на LLVM в виде одной функции
     - Компиляция выражений в операторе WHERE с помощью LLVM
   - Ускорение в 2-8 раз на Q1, в ~3 раза в среднем на TPC-H

1-й шаг: компиляция выражений

ИСП РАН

- Расширение PostgreSQL (Custom Scan)
  - Для SeqScan изменена фильтрация:
    - Добавлена компиляция дерева выражений в LLVM
    - Исполнение оптимизированного кода для фильтрации каждого tuple
    - Поддерживаются базовые операции для int, float, Date (около 100 из ~2000)
- Результат: ускорение в 2 раза для простых синтетических тестов
  - удаление неявных вызовов для каждой операции
  - константы в запросе => значения в инструкции
  - оптимизация выражений средствами LLVM

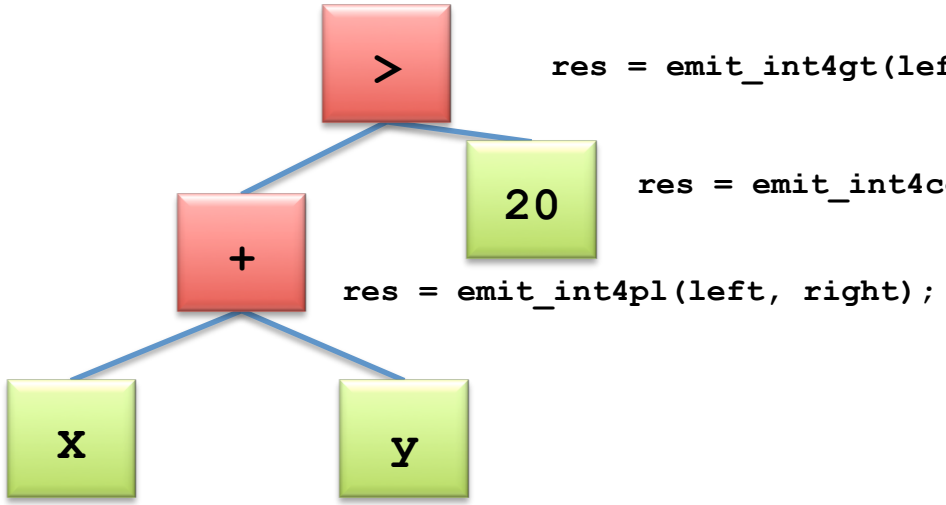
Поддержка операций в LLVM

ИСП РАН

- При обходе дерева выражений операция в узле не выполняется, а создает LLVM-биткод, который ей соответствует
- Необходимо реализовать кодогенерацию для всех операций и всех типов, поддерживаемых в выражениях (около 2000)
  1. Реализация всех операций вручную
     - однообразная работа, легко допустить ошибку, сложно поддерживать
  2. Предварительная компиляция кода операций в LLVM-биткод: *src/backend/*.c* => *.bc => all_ops.bc
     - Долгая компоновка
  3. Автоматическая компиляция операций *src/backend/*.c* в Си-код, использующий LLVM API для генерации кода из п.1
     - Соответствующий инструмент из LLVM устарел и не поддерживается

Кодогенерация для выражений

ИСП РАН




```
res = emit_int4gt(left, right);
```



```
res = emit_int4const(20);
```



```
res = emit_int4pl(left, right);
```



```
res = emit_load_int4_var('X');
```

```
res = emit_load_int4_var('Y');
```

1. *emit_int4gt() { LLVMBuildICmp(…); }*
2. *LLVMBuildCall("int4gt");*
   код int4gt уже скомпилирован при сборке PostgreSQL и подгружен из .bc-файла. При компиляции вызов заменится на тело функции.
3. *emit_int4gt()* как в п.1, но был создан автоматическим генератором на этапе сборки, как в п.2

Профилирование TPC-H

ИСП РАН

TPC-H Q1:

```
select
      l_returnflag,
      l_linestatus,
      sum(l_quantity) as sum_qty,
      sum(l_extendedprice) as sum_base_price,
      sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
      sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
      avg(l_quantity) as avg_qty,
      avg(l_extendedprice) as avg_price,
      avg(l_discount) as avg_disc,
      count(*) as count_order
from
      lineitem
where
      l_shipdate <=
      date '1998-12-01' -
      interval '60 days'
group by
      l_returnflag,
      l_linestatus
order by
      l_returnflag,
      l_linestatus;
```

Function	TPC-H Q1	TPC-H Q2	TPC-H Q3	TPC-H Q6	TPC-H Q22	Average on TPC-H
ExecQual	6%	14%	32%	3%	72%	25%
ExecAgg	75%	-	1%	1%	2%	16%
SeqNext	6%	1%	33%	-	13%	17%
IndexNext	-	57%	-	-	19%	38%
BitmapHeapNext	-	-	-	85%	-	85%

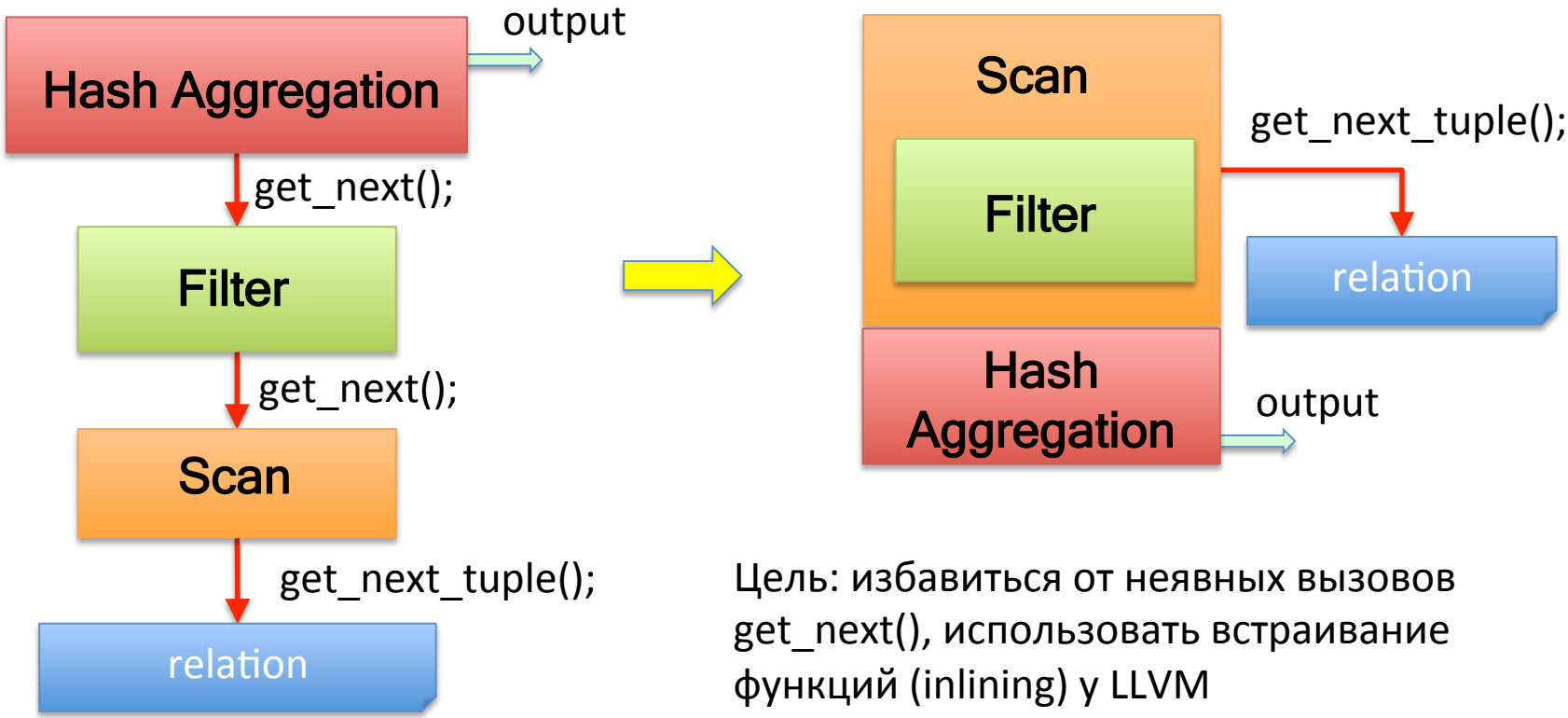
2-й шаг: реализация Scan и Aggregation на LLVM

ИСП РАН

- Расширение PostgreSQL (Executor hook)
  - Реализация Scan, Aggregation на LLVM
    - Поддерживаются SeqScan, IndexScan, IndexOnlyScan, DirectAggregation, HashAggregation
  - Отказ от “Volcano-Style” итерационной модели
    - Реализация HashAggregation, SeqScan и Filter на LLVM в виде одной функции, внешний цикл выполняет Scan
- Результат: на данный момент ускорение ~25% на TPC-H Q1

Избавление от итерационной “Volcano-style” модели

ИСП РАН



Цель: избавиться от неявных вызовов get_next(), использовать встраивание функций (inlining) у LLVM

Заключение

ИСП РАН

- Разработано расширение PostgreSQL для динамической компиляции SQL-запросов с помощью LLVM JIT. Реализованы фазы:
  - Фильтрация (компилируются выражения из оператора WHERE, поддерживаются типы int, float, Date, Numeric)
  - Сканирование (SeqScan, IndexScan, IndexOnlyScan)
  - Агрегация (DirectAggregation, HashAggregation; sum, avg, count)
- Результаты:
  - Ускорение в ~2 раза на простых синтетических тестах
  - Ускорение на ~25% на TPC-H Q1

Будущая работа

ИСП РАН

- Компиляция выражений: реализовать все операции и типы данных на LLVM, или скомпилировать автоматически из кода Postgres
- Реализовать на LLVM все виды сканирования и агрегации, сортировки (в т.ч. в одном цикле)
- Реализовать поддержку JOIN и подзапросов
- Реализовать на LLVM *slot_deform_tuple()* под конкретный запрос
- Тестирование на всех тестах TPC-H и других бенчмарках, профилирование, поиск мест для оптимизации

Будущая работа

ИСП РАН

- Использование параллелизма:
  - Реализация параллельного сканирования и агрегации на LLVM
  - Параллельная компиляция
- Подготовка к релизу в Open Source, взаимодействие с PostgreSQL Community

ИСП РАН

Спасибо!

Вопросы, пожелания, обратная связь:
dm@ispras.ru

ИСП РАН