Андрей Державин,
Антон Шурыгин

JIT-библиотеки для симуляции CPU: сложности выбора

C++ Russia 2026

Авария «Ариан-5»

- 4 июня, 1996 года произошла одна из самых дорогостоящих компьютерных ошибок в истории
- Сбой в ПО, унаследованном от ракеты предыдущего поколения

Спикеры

- аспиранты МФТИ
- авторы и преподаватели курса “Симуляция ЦП и ОС” в МФТИ и ИТМО

Андрей Державин

- Функциональная симуляция
- Микро-архитектурные оптимизации
- Двоичная инструментация

Антон Шурыгин

- Двоичная инструментация
- Дизайн архитектуры
- Симуляция

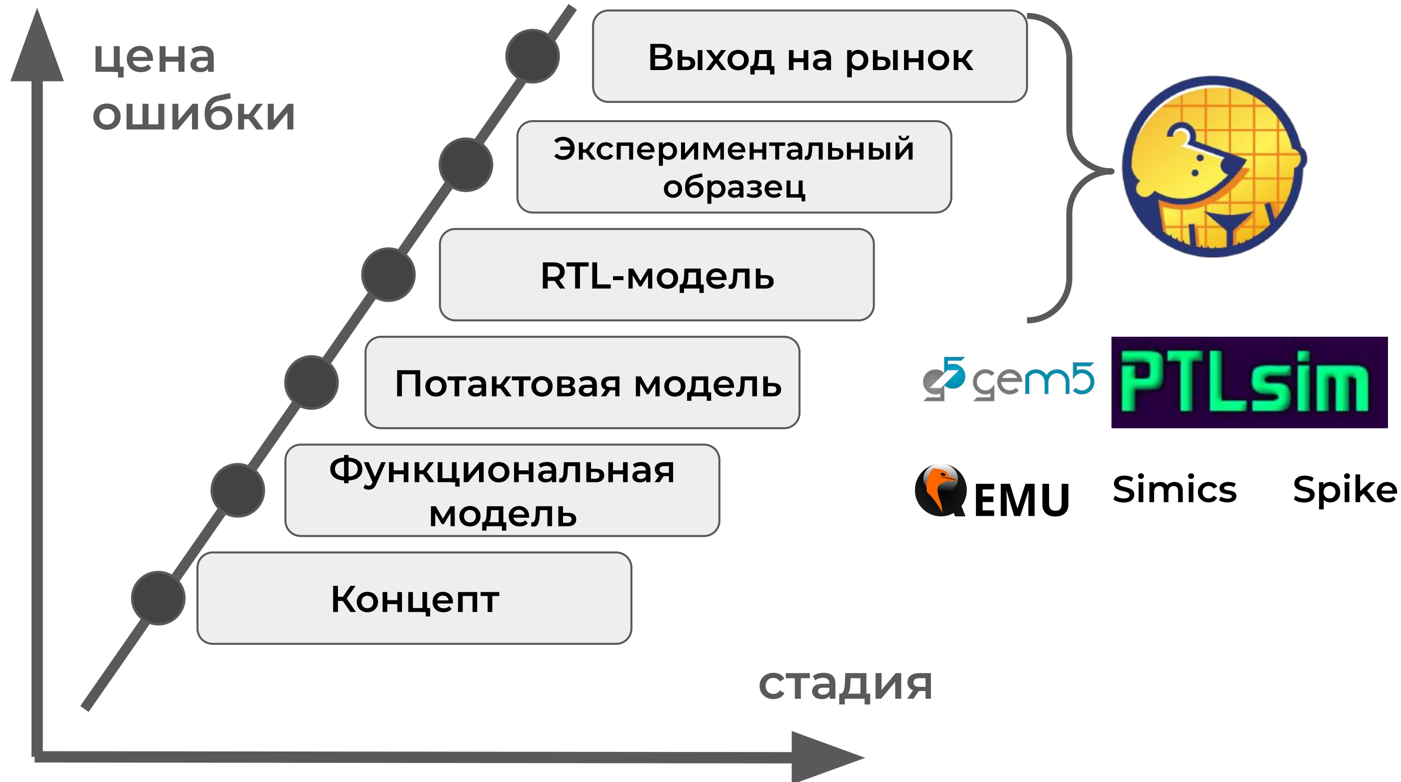
План

- Функциональная симуляция
- Ускорение симуляции с помощью JIT
- Существующие JIT-библиотеки
- Фреймворк для экспериментов
- Результаты бенчмарков
- Рекомендации

Моделирование систем

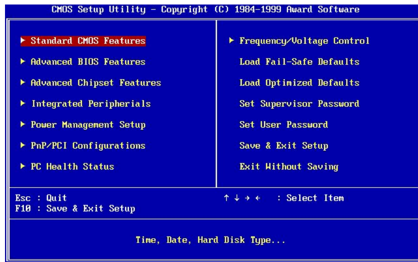
Разработка вычислительных систем – сложный и длительный процесс: Архитектура, дизайн, обратная совместимость, выявление ошибок на ранних этапах

Моделирование систем

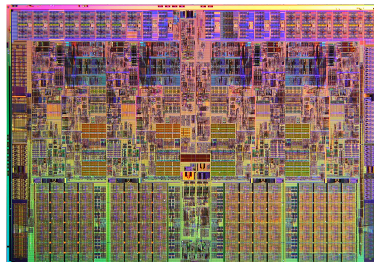


Применение моделирования

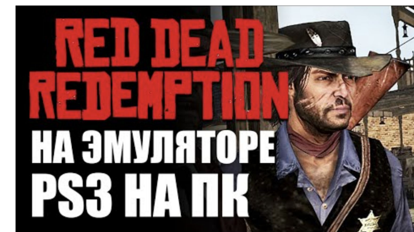
Обнаружение ошибок проектирования



Разработка ПО под недоступную аппаратуру



Исследование пространства проектирования

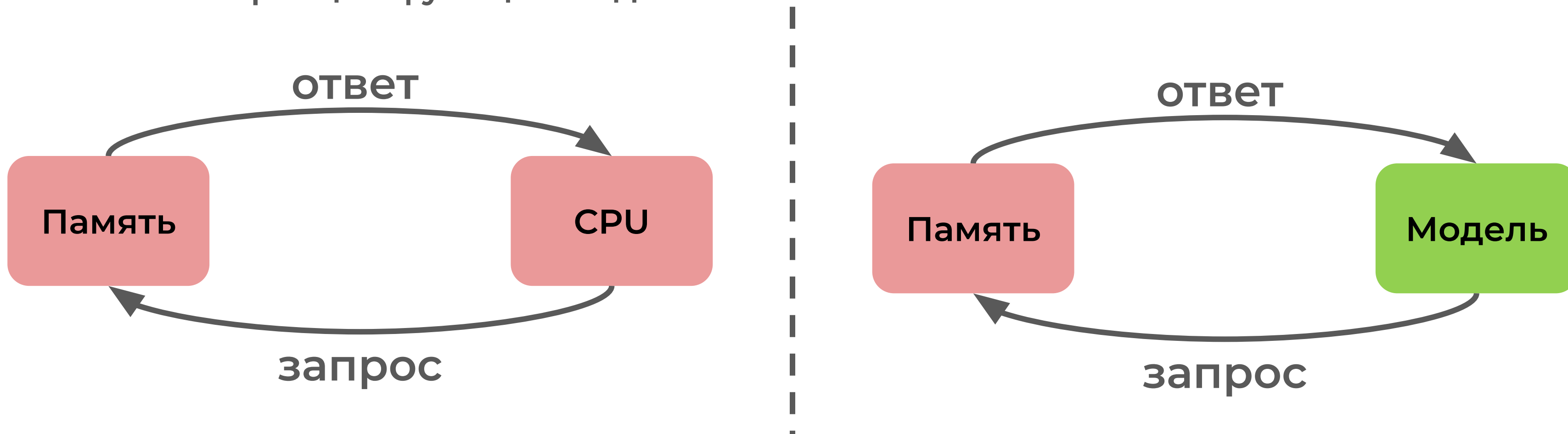


Выполнение программ на «неродной» архитектуре

Что такое моделирование?

- Метод исследования процессов на моделях

➢ *Почему моделирование возможно для ЦП?*
  - Модульность подсистем
  - Абстракция функций подсистем



grigory-rechistov/simbook: Simulation Foundations Book, in Russian · GitHub

Уровни абстракций

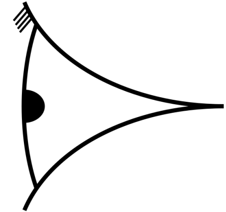
программный уровень

- Приложения
- Алгоритмы
- Язык программирования
- Операционная система

Набор команд

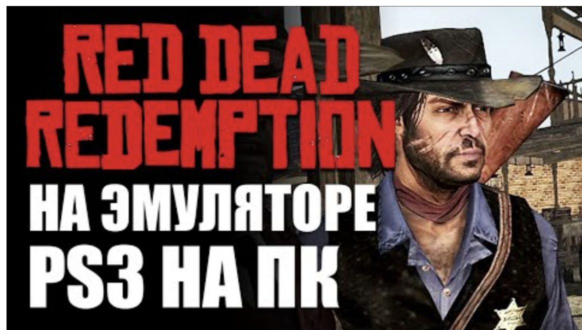
аппаратный уровень

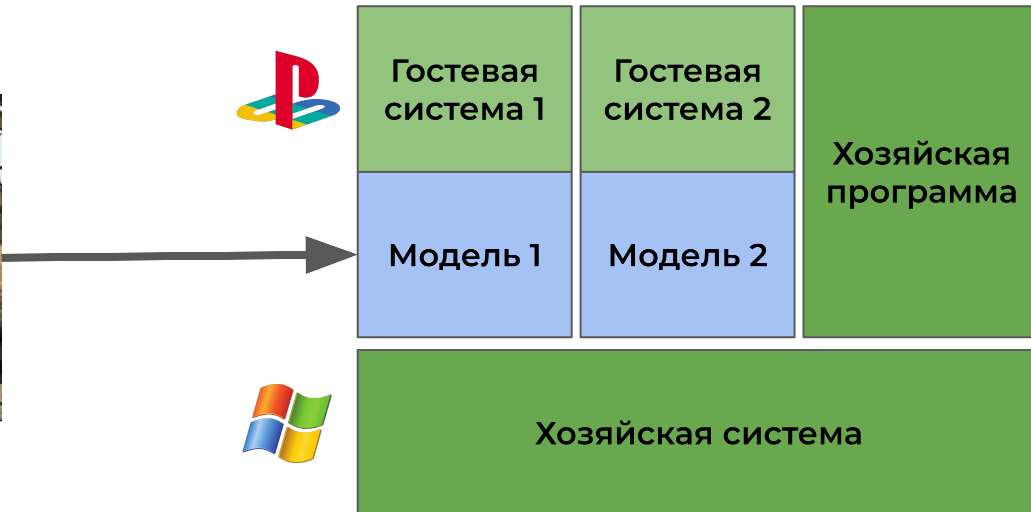
- Микроархитектура
- Интегральные схемы
- Цифровая логика
- Физический уровень



Гостевые и хозяйские системы

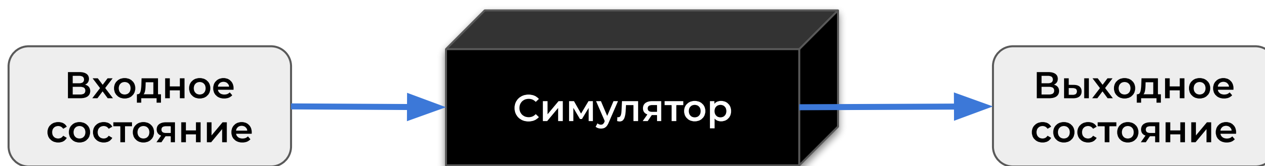
- Гостевая система (GUEST) - система, поведение которой моделируется
- Хозяйская система (HOST) - система, на которой моделируется поведение гостевой системы





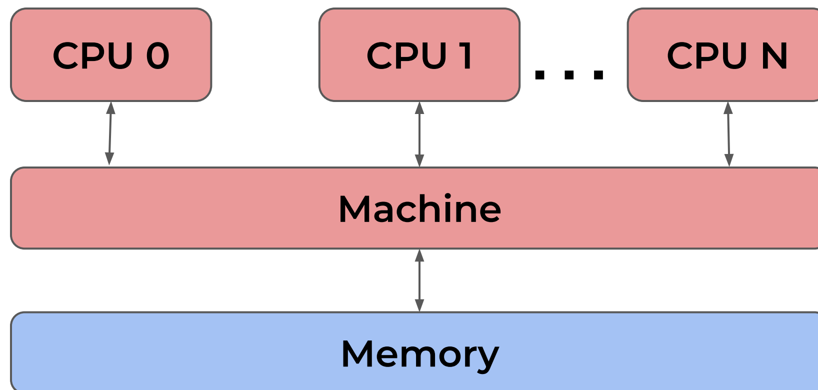
Функциональная симуляция

- Фокус на поведении СК
- Микроархитеткурные параметры не учитываются
- Подходит для разработки и отладки ПО



Программная модель CPU

- Ядро исполняет инструкции
- Память - хранит код и данные
- Пишем свой симулятор...



Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

Интерпретатор

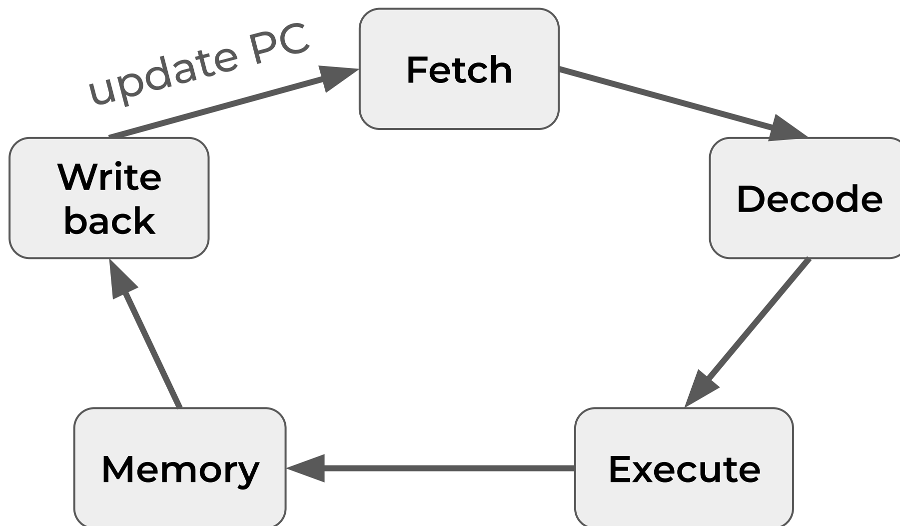
- Простейший подход симуляции
- Аналогия с интерпретаторами ЯП

```
a = 20
b = 11
b *= 2
c = a + b
```

```
addi x1, x0, 20
addi x2, x0, 11
slli x2, x2, 1
add x3, x1, x2
```

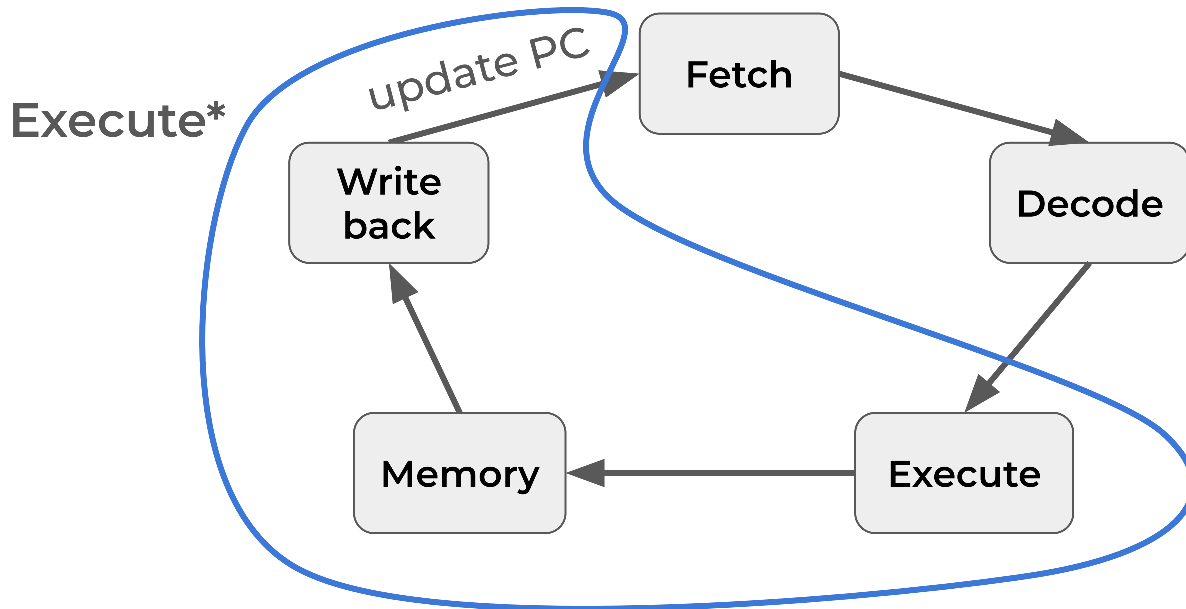
Интерпретатор

- CPU исполняет инструкции по стадиям
- Модели не нужно учитывать все детали



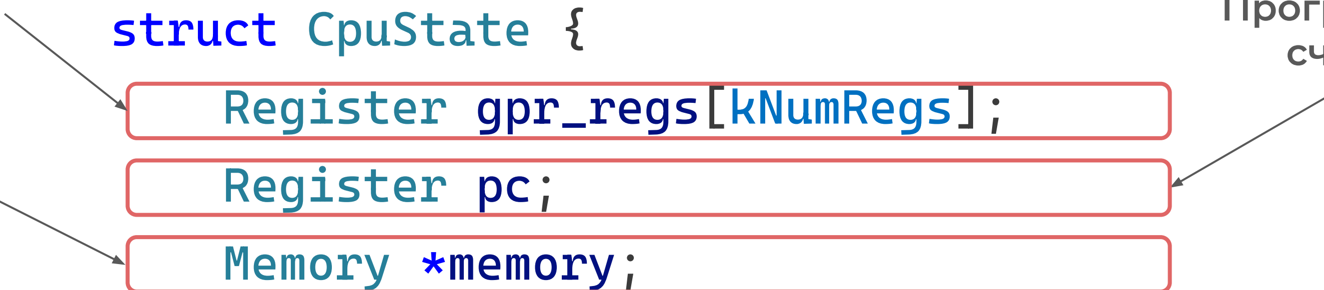
Интерпретатор

- CPU исполняет инструкции по стадиям
- Модели не нужно учитывать все детали



Простейшая модель на C++

```cpp
using Register = uint32_t;
constexpr std::size_t kNumRegs = 32;
struct CpuState {
    Register gpr_regs[kNumRegs];
    Register pc;
    Memory *memory;
};
struct Memory {
    std::vector<uint8_t> data;
};
```

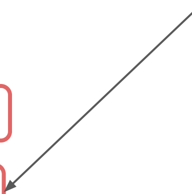


Простейшая модель на C++

```cpp
using Word = std::uint32_t;
enum class Opcode : std::uint8_t {
    kUnknown = 0,
    kAdd,
    // …
};
struct Instruction {
    Opcode opc{};
    Word src1{}, src2{}, dst{};
};
```

Код операции

Операнды инструкции



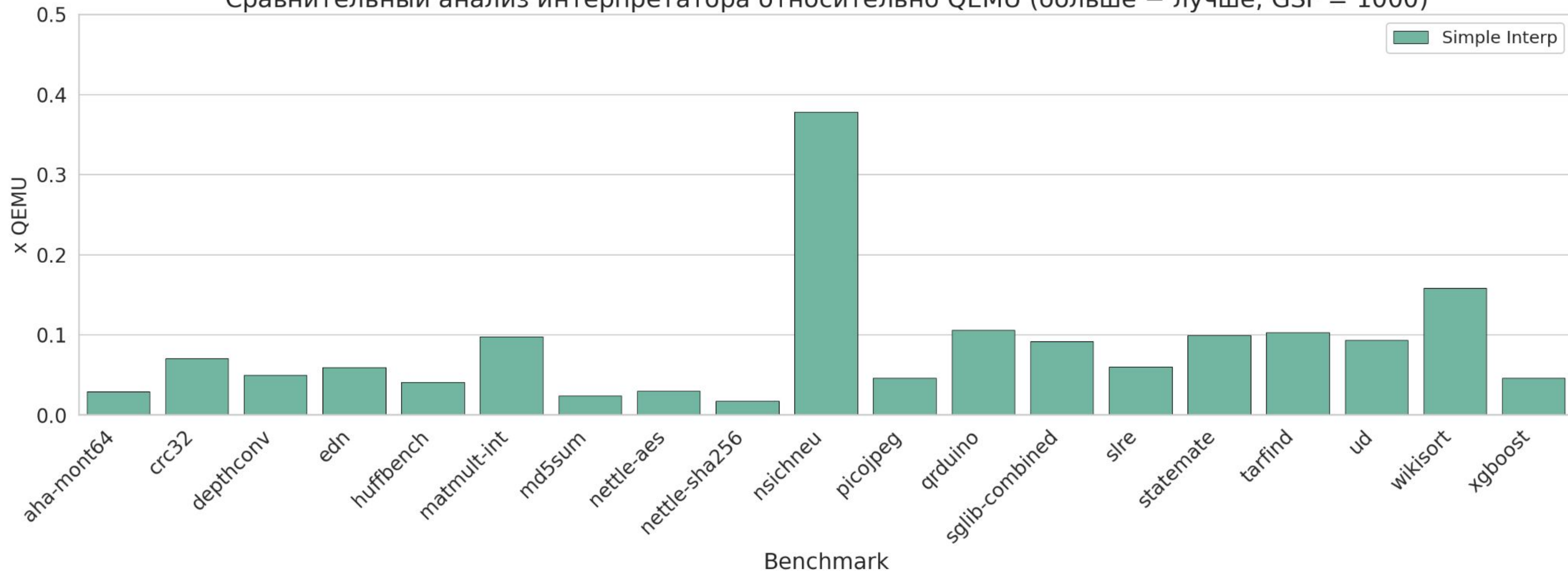
Простейшая модель на C++

```
Instruction decode(Word encoding) {
    Instruction insn{.opcode = get_opcode(encoding)};
    switch (insn.opcode) {
        case Opcode::kAdd:
            insn.src1 = get_src1(encoding);
            insn.src2 = get_src2(encoding);
            insn.dst = get_dst(encoding);
            break;
    }
    return insn;
}
```

Простейшая модель на C++

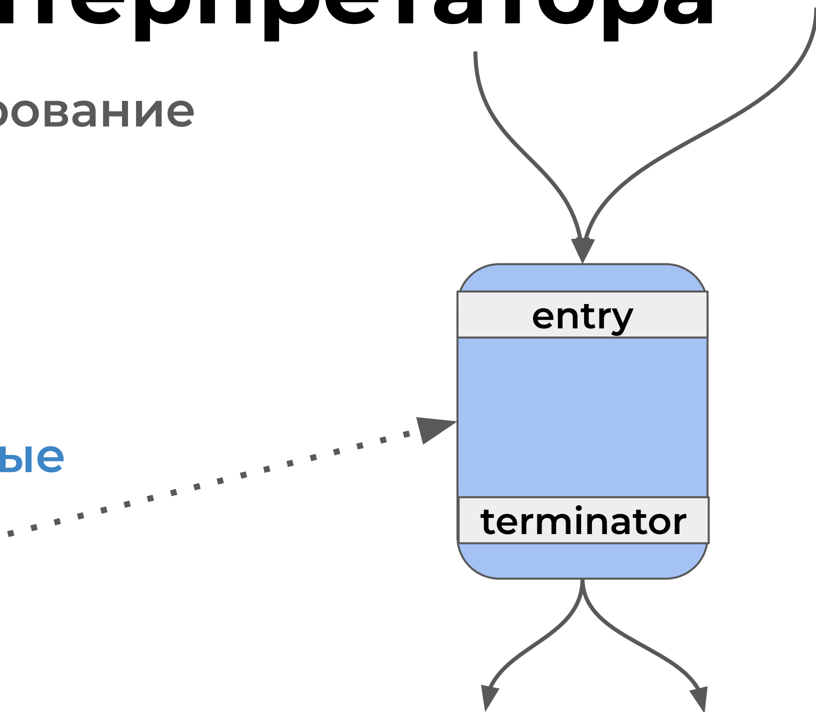
```
void execute(CpuState *cpu, Instruction insn) {
    switch (insn.opcode) {
        case Opcode::kAdd:
            auto res = cpu->getReg(insn.src1) +
                       cpu->getReg(insn.src2);
            cpu->setReg(insn.dst, res);
            break;
    }
}
```

Померимся?

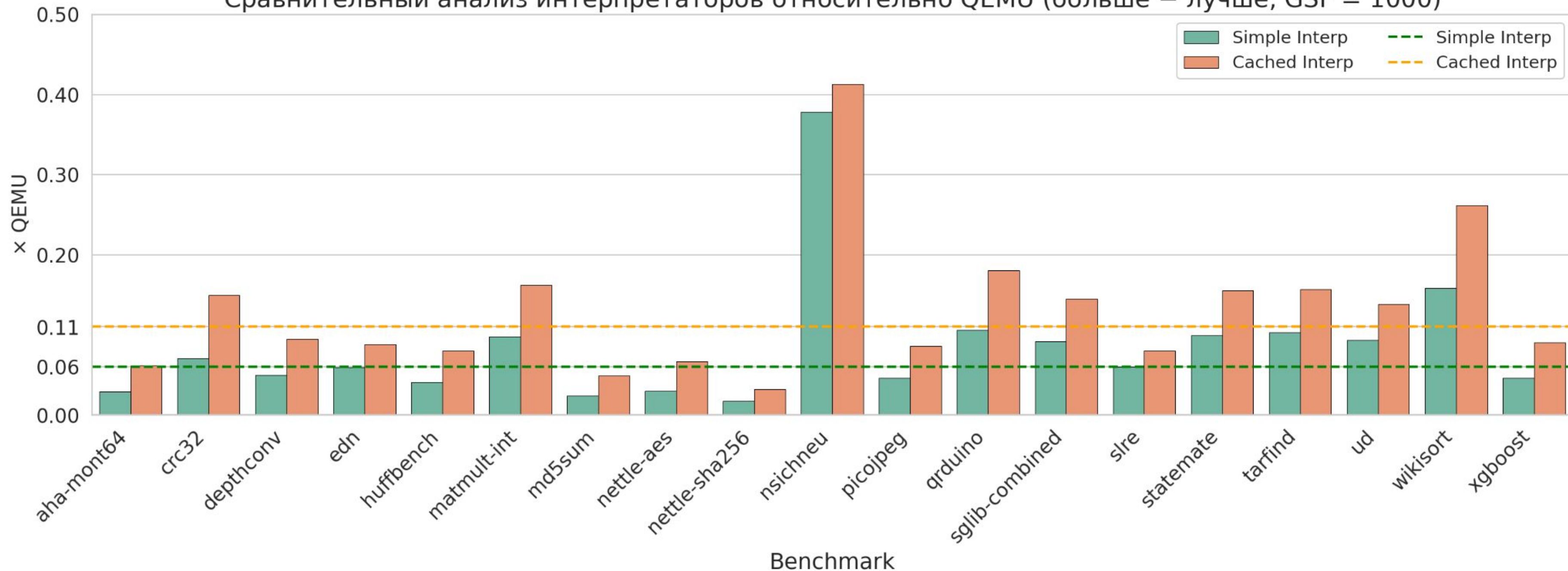


Ускорение интерпретатора

- Повторное декодирование инструкций:
  - Добавим кэш декодирования
  - Храним линейные участки кода

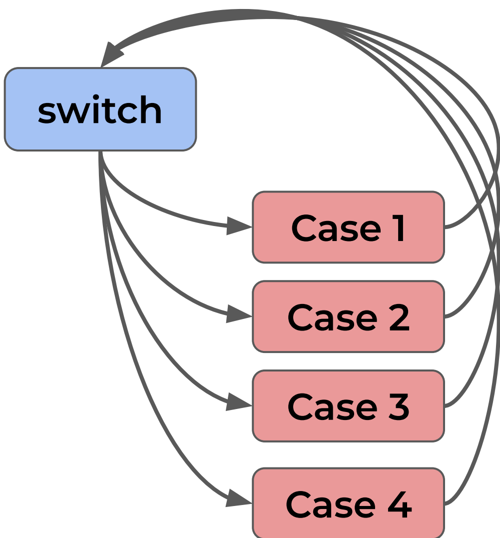


Померимся 2



Ускорение интерпретатора

- Чем плох один большой switch?

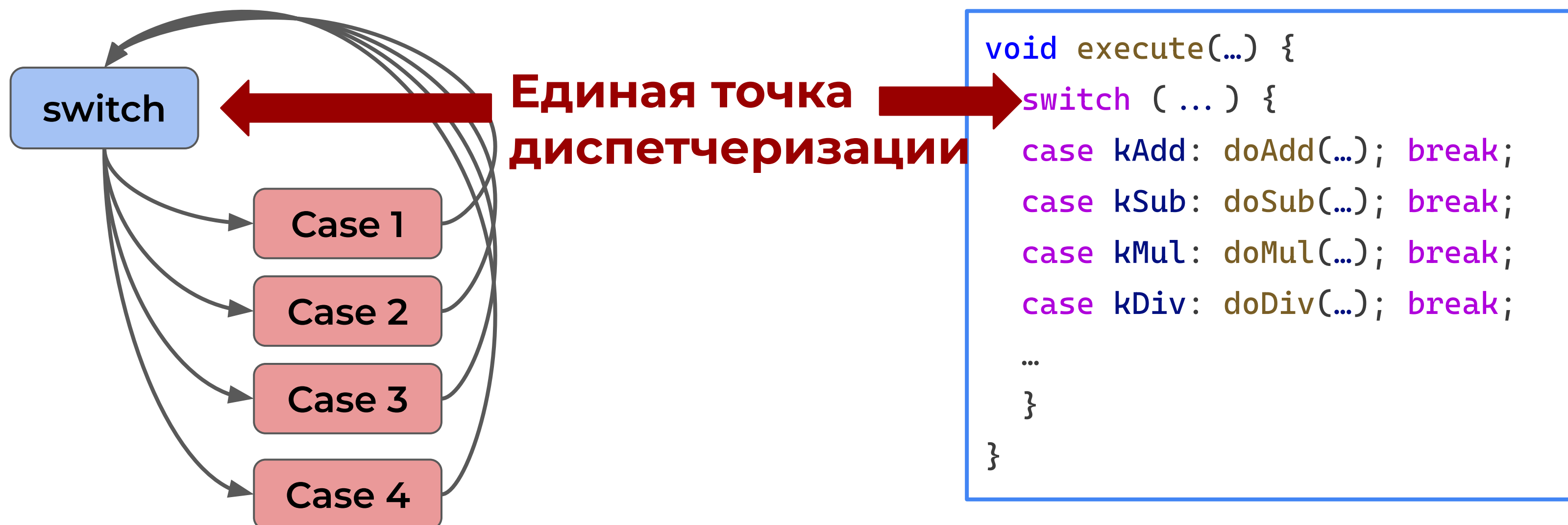



```
void execute(…) {
  switch ( ... ) {
  case kAdd: doAdd(…); break;
  case kSub: doSub(…); break;
  case kMul: doMul(…); break;
  case kDiv: doDiv(…); break;
  …
  }
}
```

Branch prediction, или Откуда процессор берет производительность (Часть 1) | Доклад на C++ Russia 2025

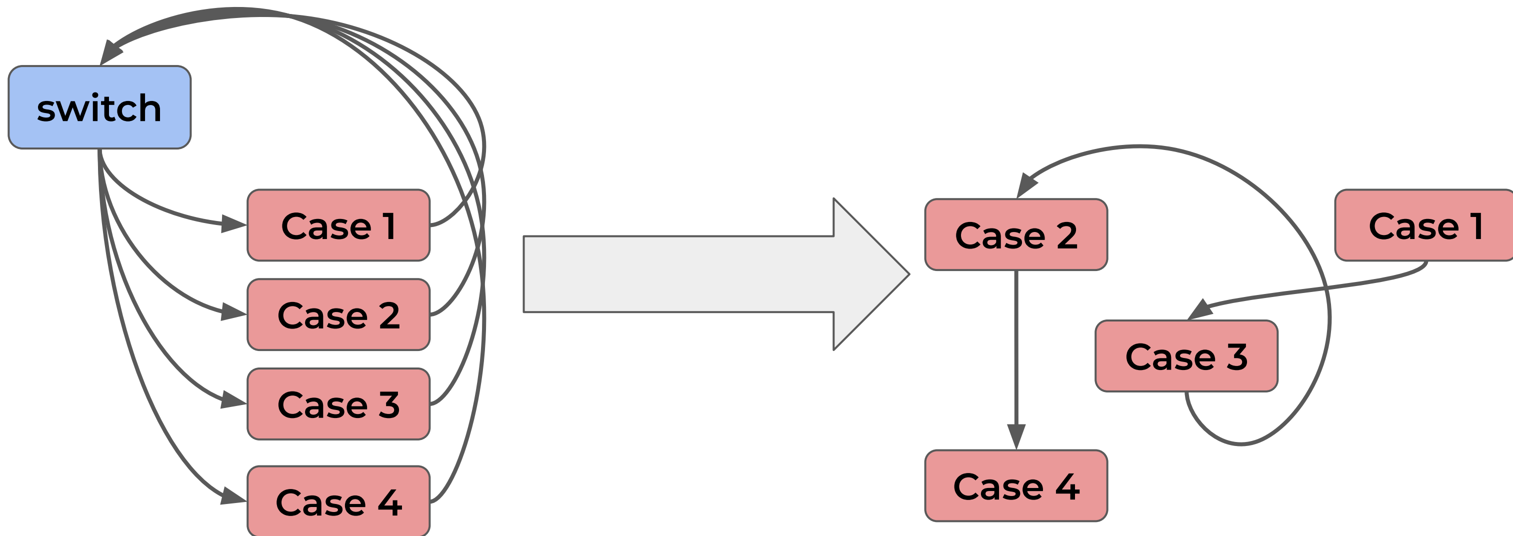
Ускорение интерпретатора

- Чем плох один большой switch?

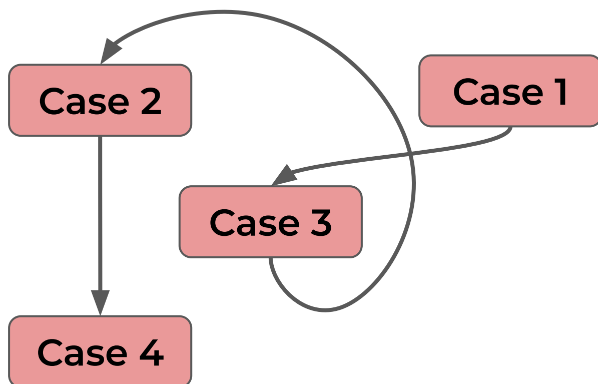


Ускорение интерпретатора

- Шитый код (threaded code)
- Введём несколько точек диспетчеризации:



Ускорение интерпретатора



GNU Extension

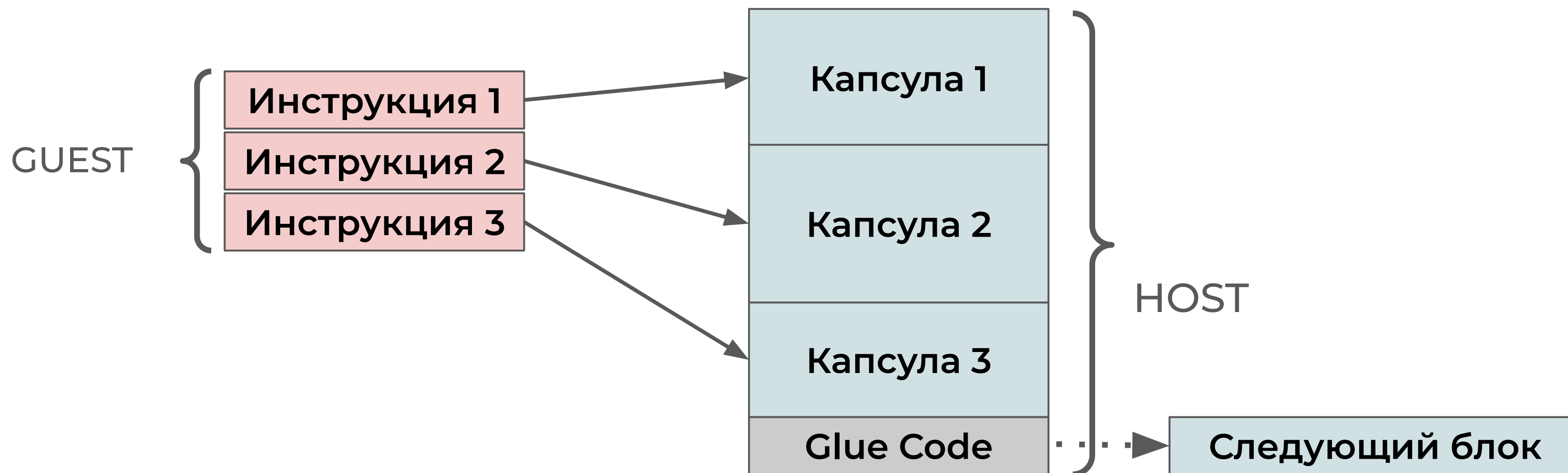
```
DO_ADD:
  doAdd(cpu, insn);
  goto *label[++pc];
```

Указатели на функции

```
#define DISPATCH()    \
  [[clang::musttail]] \
  return              \
  (handlers[memory[pc]])()
void doAdd() {
  // ...
  DISPATCH();
}
```

Еще быстрее!

- Интерпретатор - простой, но медленный
- Идея: “компилировать” блоки кода гостевой архитектуры



Капсулы

- Рассмотрим капсулы для инструкции сложения – ADD
- Регистр rdi хранит адрес начала `CpuState` (регистровый файл)

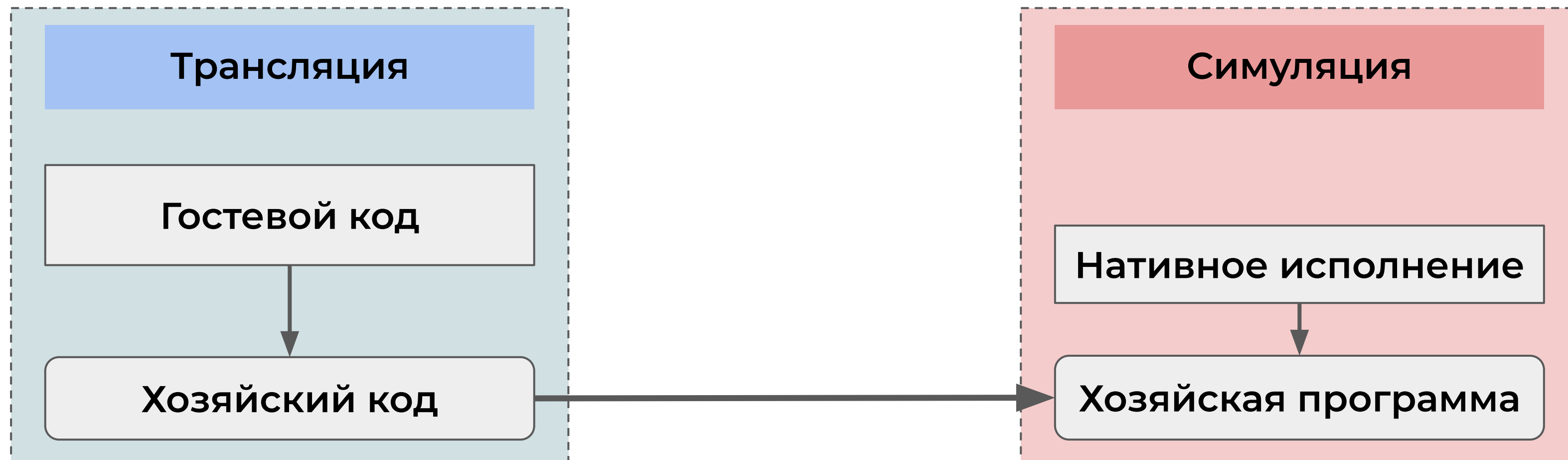
```
add x1, x2, x3
```

→

```
mov edx,DWORD PTR [rdi+0x8]
add edx,DWORD PTR [rdi+0xC]
mov DWORD PTR [rdi+0x4],edx
```

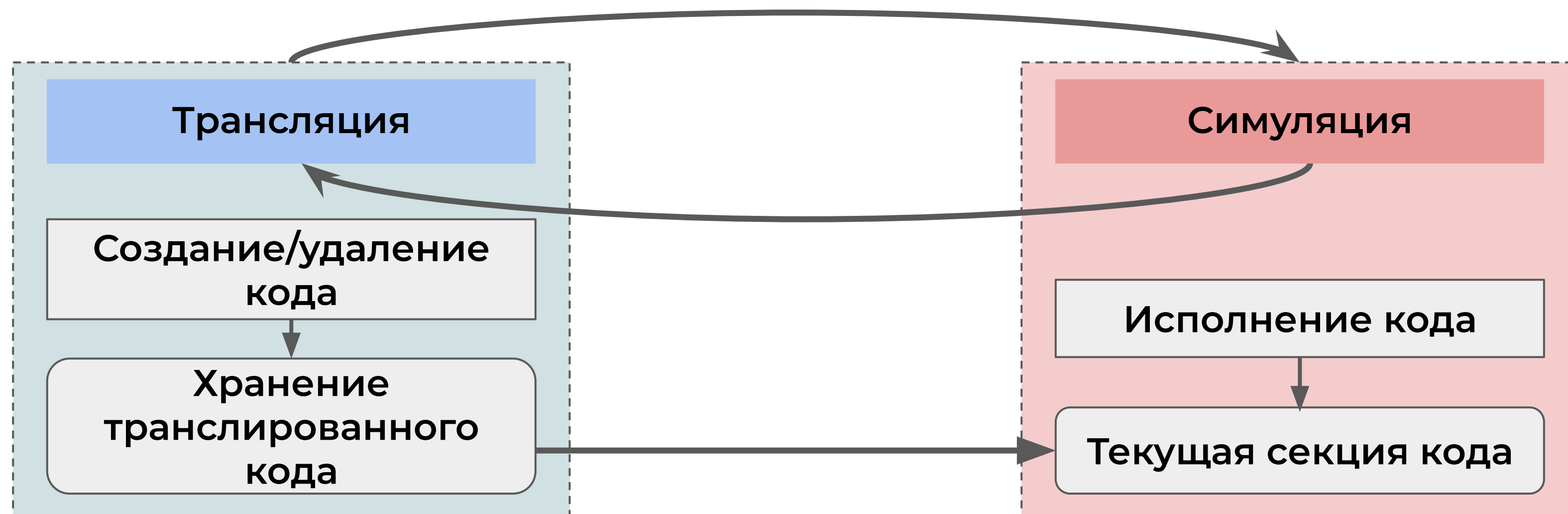
Типы трансляции

➢ Статическая двоичная трансляция – трансляция всей программы из гостевого кода в хозяйский до ее запуска (AOT-компиляция)



Типы трансляции

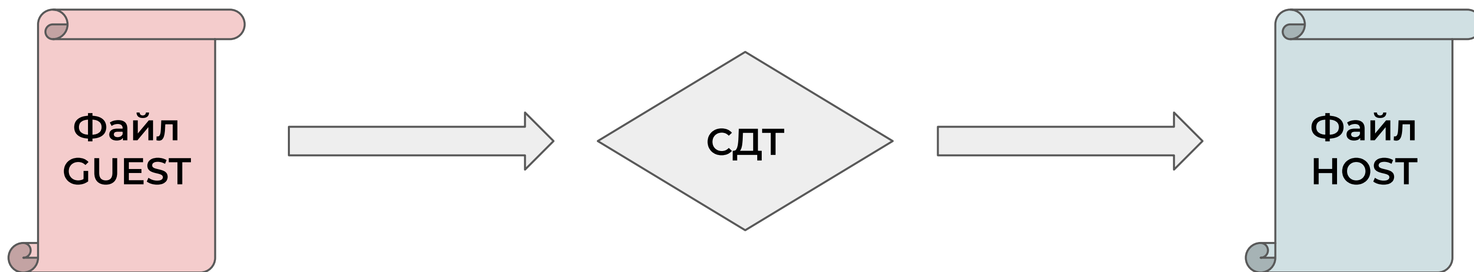
- Динамическая двоичная трансляция – трансляция во время исполнения программы по мере необходимости (JIT-компиляция)



Двоичная трансляция

- Единица двоичной трансляции для симуляции ЦП

➢ Предложение №1: файл/модуль



➕ Многократный запуск
➕ Нативное исполнение
➖ Неэффективное использование ресурсов
➖ Нет доступа к runtime информации
➖ Сломали самомодифицирующийся код

Двоичная трансляция

- Единица двоичной трансляции для симуляции ЦП

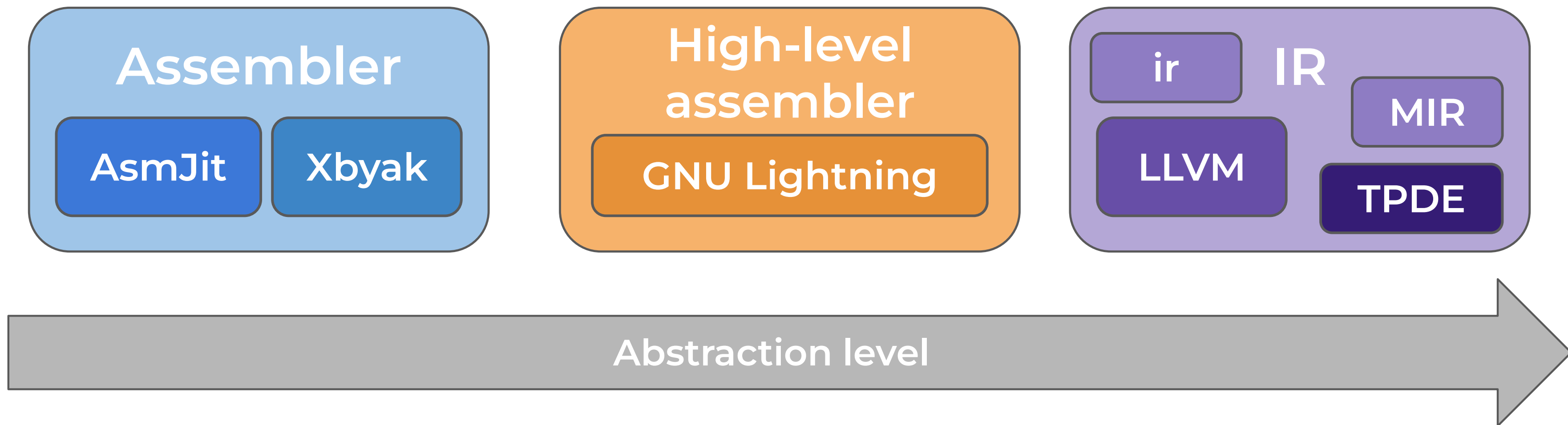
➢ Предложение №2: линейный участок кода

➕ Быстрый старт, трансляция по требованию
➕ Доступ к runtime информации об исполнении
➖ Накладные расходы на частый interop
➖ Дополнительная память под кэш трансляций

Современное состояние

- Быстрые симуляторы с JIT трансляцией:
  - QEMU – популярный эмулятор с собственным JIT компилятором
  - Pydgin – исследовательский фреймворк с автоматической генерацией симулятора с JIT
- Запрос на создание собственных симуляторов с JIT:
  - Прототипирование для архитектурных исследований
  - Образовательная ценность
- Сторонние JIT библиотеки:
  - Фокус на трансляции
  - Отсутствие сравнительного анализа производительности

Сторонние JIT библиотеки



Сторонние JIT библиотеки

Библиотека	Кодогенерация	Архитектуры	Интеграция	Статус
AsmJit	regalloc + min. opt	x86_64, ARM64	Долгая история...	Активен
Xbyak	pure asm	x86_64, ARM64*, RISC-V*		Активен
GNU Lightning	regalloc + min. opt	x86_64, ARM, MIPS, PowerPC,...		Активен? (2024)
LLVM	regalloc + opt	x86_64, ARM, RISC-V,...		Активен
MIR	regalloc + base opts	x86_64, ARM64, RISC-V, ppc64le,...		Заморожен (2024)
TPDE-llvm	regalloc + base opts	x86_64, ARM64		Активен
ir	regalloc + base opts*			Активен

IR JIT Framework: база для следующего поколения JIT в PHP | Доклад на Joker 2023

Архитектура проекта

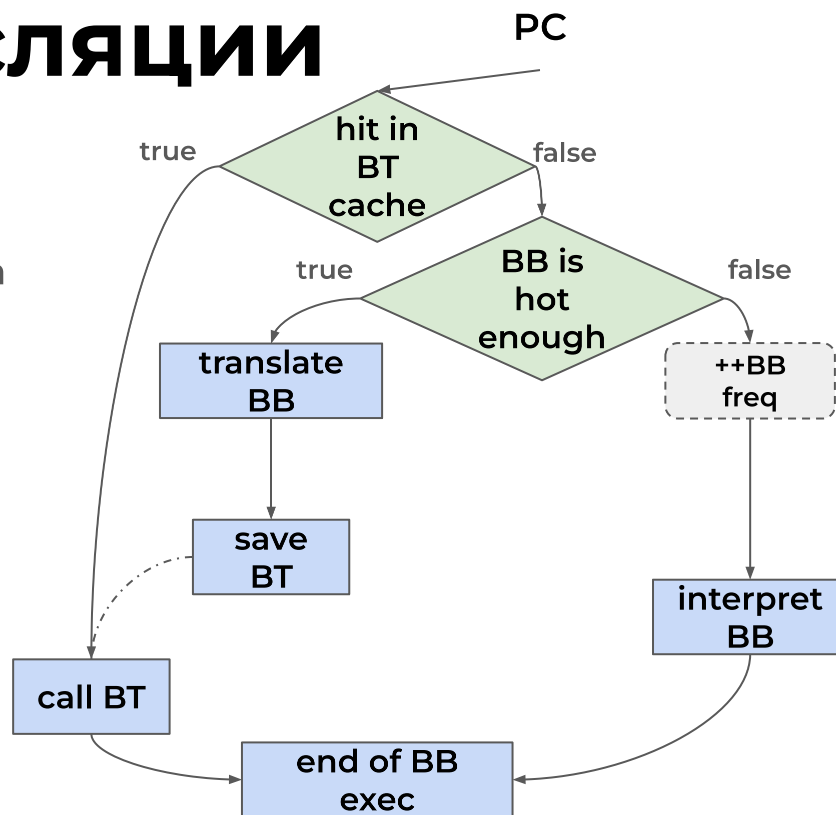
- Модульный симулятор для сравнения JIT-библиотек.
- Поддержка ДДТ с использованием популярных JIT-библиотек.
- Сравнение использования, производительности и накладных расходов.

Naive Interpreter
DBT Engine
Execution engine(s)
CPU state
Memory model(s)
Core Model
Translator
MIR
LLVM
GNU
TPDE
ASMjit
xbyak
ir
Empty
DBT Backend(s) Factory
Plain Memory
Page Memory

Алгоритм трансляции

- BTCache – кэш трансляций
- BBCache – кэш базовых блоков
- THRESHOLD - параметр транслятора
  - Минимальная “горячесть” для начала трансляции

```
if PC in BTCache:
  BTCache[PC].call()
else if BBCache[PC].exec_count ≥ THRESHOLD:
  code = translate(PC)
  BTCache[PC] = code
  code.call()
else:
  BBCache[PC].exec_count++
  interpret(PC)
```

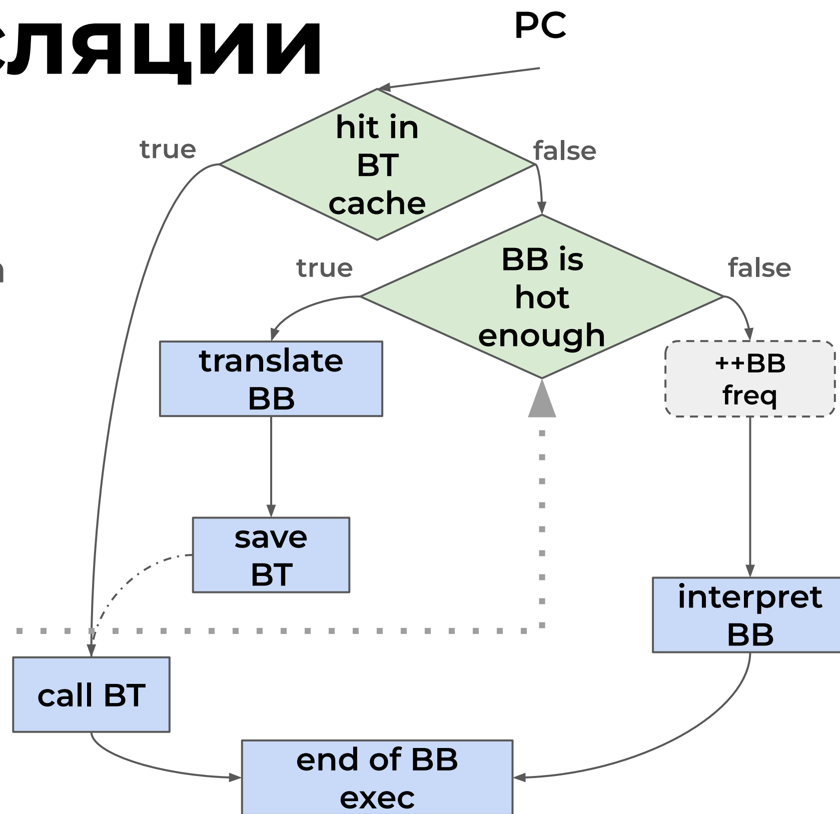


Topham & Jones, 2007

Алгоритм трансляции

- BTCache – кэш трансляций
- BBCache – кэш базовых блоков
- THRESHOLD - параметр транслятора
  - Минимальная “горячесть” для начала трансляции

```
if PC in BTCache:
  BTCache[PC].call()
else if BBCache[PC].exec_count ≥ THRESHOLD:
  code = translate(PC)
  BTCache[PC] = code
  code.call()
else:
  BBCache[PC].exec_count++
  interpret(PC)
```



Topham & Jones, 2007

Бенчмарки

- embench-iot – набор тестов для встраиваемых систем на базе ARM Cortex-M

Тип вычислений	Тесты (Embench IOT 2.0)	Описание
Compute-bound	aha-mont64, crc32, nettle-sha256, md5sum, nettle-aes	**Хеши, шифрование, CRC, целочисленное умножение**
Memory-bound	matmult-int, statemate	**Перемножение матриц, конечный автомат**
Memory & Branching Intensive	sglib-combined, wikisort, slre, nsichneu	**Структуры данных, сортировка, регулярные выражения, сетевой протокол**
Balanced	edn, picojpeg, qrduino, ud, depthconv, xgboost	**Свёртки, JPEG, QR, сжатие, поиск в архиве**

Державин Андрей, Шурыгин Антон

Померимся 3

Сравнительный анализ симуляторов с ДДТ относительно QEMU (больше − лучше, GSF = 1000)
THRESHOLD=0

× QEMU

1.2
1.0
0.8
0.6
0.4
0.2
0.0

aha-mont64 crc32 depthconv edn huffbench matmult-int md5sum nettle-aes nettle-sha256 nsichneu picojpeg qrduino sglib-combined slre statemate tarfind ud wikisort xgboost

Benchmark

Державин Андрей, Шурыгин Антон

Померимся 3

Сравнительный анализ симуляторов с ДДТ относительно QEMU (больше – лучше, GSF = 1000)
THRESHOLD=10

× QEMU

1.2
1.0
0.8
0.6
0.4
0.2
0.0

aha-mont64
crc32
depthconv
edn
huffbench
matmult-int
md5sum
nettle-aes
nettle-sha256
nsichneu
picojpeg
qrduino
sglib-combined
slre
statemate
tarfind
ud
wikisort
xgboost

Benchmark

Померимся 3

Сравнительный анализ симуляторов с ДДТ относительно QEMU (больше – лучше, GSF = 1000)
THRESHOLD=1000

× QEMU

1.2
1.0
0.8
0.6
0.4
0.2
0.0

aha-mont64
crc32
depthconv
edn
huffbench
matmult-int
md5sum
nettle-aes
nettle-sha256
nsichneu
picojpeg
qrduino
sglib-combined
slre
statemate
tarfind
ud
wikisort
xgboost

Benchmark

Померимся 3

Сравнительный анализ симуляторов с ДДТ относительно QEMU (больше – лучше, GSF = 1000)
THRESHOLD=10000

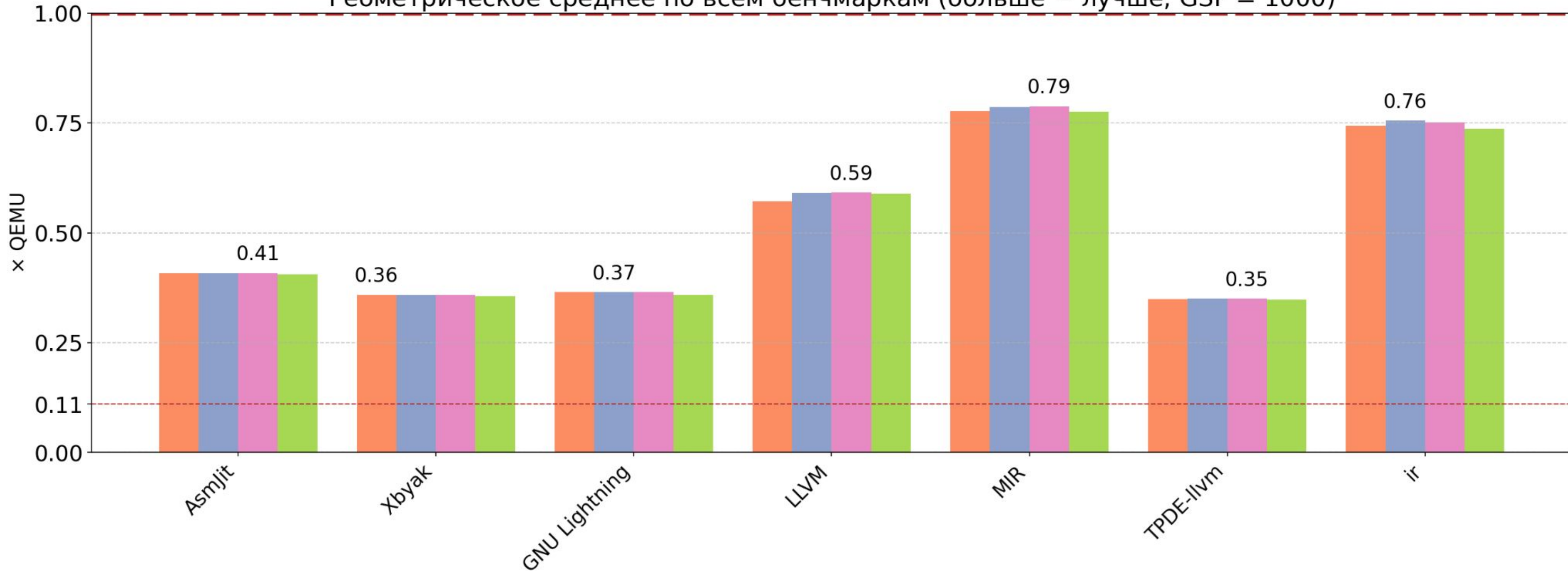
× QEMU

1.2
1.0
0.8
0.6
0.4
0.2
0.0

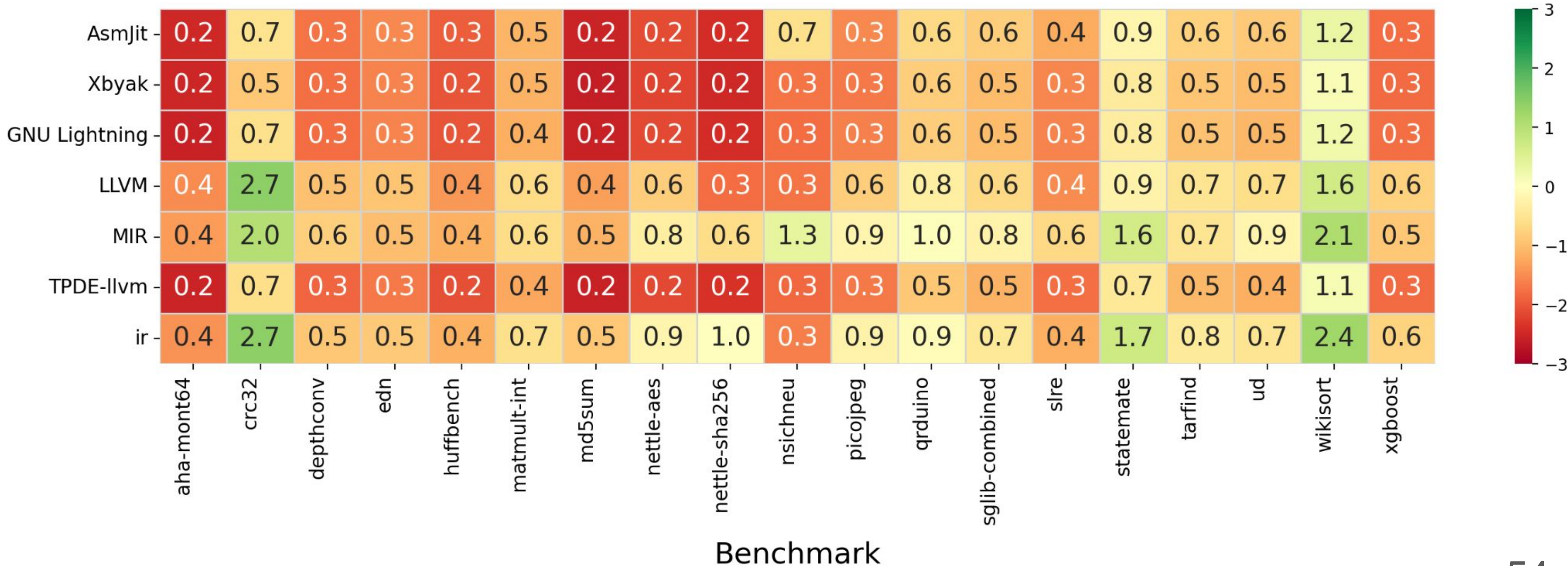
aha-mont64
crc32
depthconv
edn
huffbench
matmult-int
md5sum
nettle-aes
nettle-sha256
nsichneu
picojpeg
qrduino
sglib-combined
slre
statemate
tarfind
ud
wikisort
xgboost

Benchmark

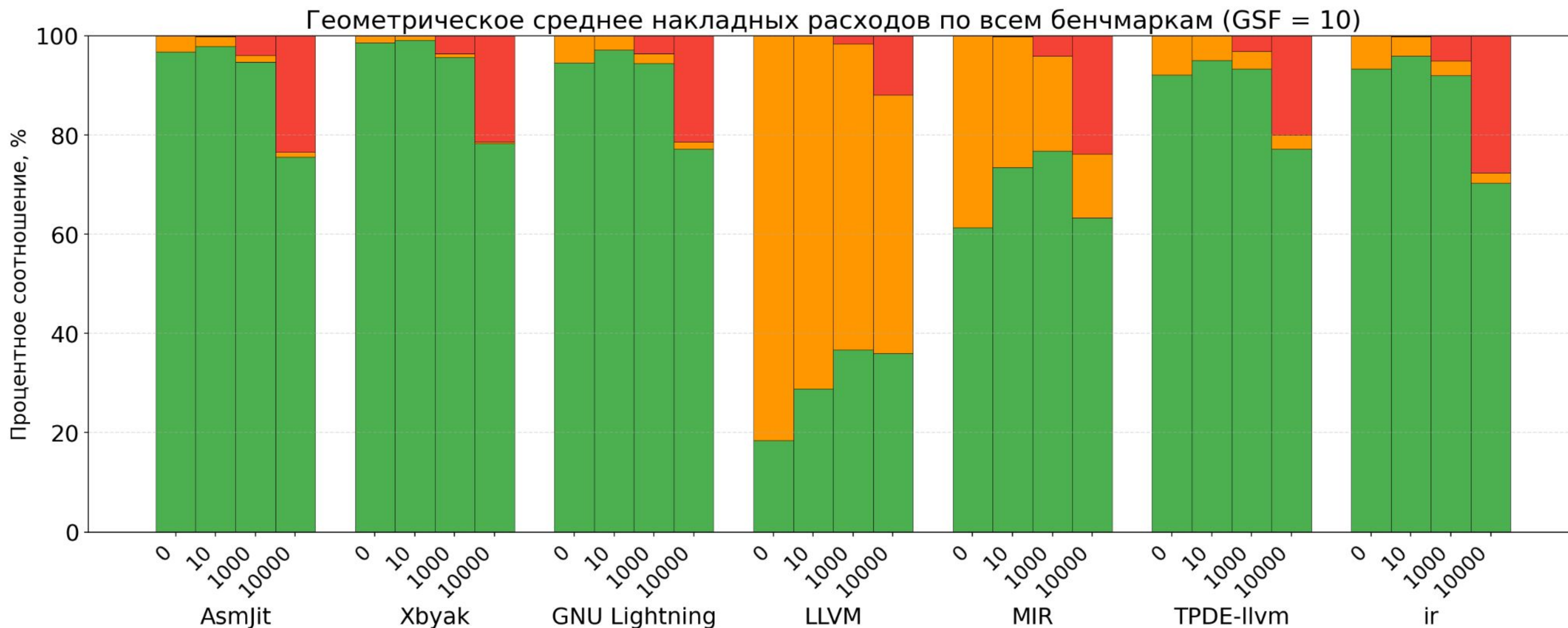
А в среднем?



Тепловая карта



Накладные расходы



Сложно?

Библиотека	Число API вызовов
AsmJit	113
Xbyak	72
GNU Lightning	57
LLVM	657
MIR	187
TPDE-llvm	630 (llvm) + 6 (TPDE)
ir	99

Рекомендации

- MIR / ir — максимальная скорость
  - C API
- Xbyak / AsmJit — (почти) полный контроль генерации
- LLVM JIT — неплох для ДДТ:
  - Тяжелые оптимизации со временем приносят свои плоды при многократном исполнении
- GNU Lightning — сложно использовать как в современных C++ проектах, так и для разработки ДДТ
- TPDE – tpde-llvm не раскрывает весь потенциал
  - tpde-ir(?)

Итоги

1. Рекомендации по использованию JIT библиотек
2. Модульная песочница для прототипирования симуляторов с ДДТ
3. Научная статья, посвященная сравнительному анализу

Q&A

JIT-библиотеки для симуляции CPU: сложности выбора

Державин Андрей, Шурыгин Антон

Ссылка на проект