LLVM шиворот навыворот

от дизассемблера к моделированию производительности

Роман Беленов, Захар Матвеев

О чём будем рассказывать

- Что такое LLVM?
- Как использовать для анализа бинарного кода
  - Смотрим на инструкции по отдельности
  - Анализируем поток инструкций
- Как использовать для моделирования производительности
  - Как профилировать без профилировки
  - Как в этом помогает LLVM
  - Как можно улучшить LLVM

Что такое LLVM?

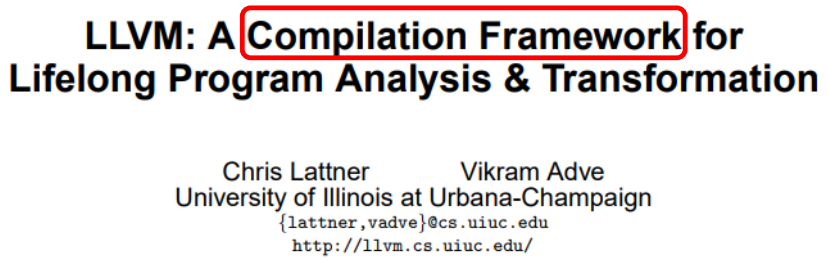
LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
{lattner,vadve}@cs.uiuc.edu
http://llvm.cs.uiuc.edu/

https://llvm.org/pubs/2004-01-30-CGO-LLVM.html

Что-то такое, из чего можно собрать свой компилятор

Что такое LLVM?



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

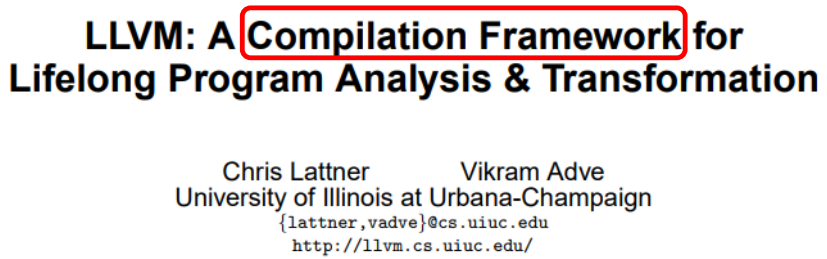
Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
{lattner,vadve}@cs.uiuc.edu
http://llvm.cs.uiuc.edu/

https://llvm.org/pubs/2004-01-30-CGO-LLVM.html

Что-то такое, из чего можно собрать свой компилятор

Правда?

Что такое LLVM?



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
{lattner,vadve}@cs.uiuc.edu
http://llvm.cs.uiuc.edu/

https://llvm.org/pubs/2004-01-30-CGO-LLVM.html

Что-то такое, из чего можно собрать свой компилятор

Правда?

Да, но не совсем

Что такое фреймворк?

Фре́ймворк — программная платформа, определяющая структуру программной системы.

https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA

Что такое фреймворк?

Фре́ймворк — программная платформа, определяющая структуру программной системы.

https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA

Типичный пример – gcc

Что такое фреймворк?

Фре́ймворк — программная платформа, определяющая структуру программной системы.

https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA

Типичный пример – gcc
Можно гибко модифицировать

Что такое фреймворк?

Фре́ймворк — программная платформа, определяющая структуру программной системы.

https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA

Типичный пример – gcc
Можно гибко модифицировать
Но на выходе всё равно получится компилятор

А на что похож LLVM?

А на что похож LLVM?

Photo courtesy Artem Kashkanov aka @radiolok

Простой детский конструктор

Набор сочетаемых друг с другом деталей

Можно собирать игрушки по инструкции

Можно собирать то, что хочется

Не устраивает деталька – можно и погнуть)

А на что похож LLVM?

https://llvm.org/doxygen/inherits.html

Это только маленький кусочек...

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies.

https://llvm.org/

Набор сочетаемых друг с другом классов

Можно собрать компилятор готовым CMakefile

Можно собирать то, что хочется

Не устраивает код – можно и поправить)

А на что похож LLVM?

https://llvm.org/doxygen/inherits.html

Это только маленький кусочек...

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies.

https://llvm.org/

Набор сочетаемых друг с другом классов

Можно собрать компилятор готовым CMakefile

Можно собирать то, что хочется

Не устраивает код – можно и поправить)

Можно поделиться с другими (у нас пока в планах)

И что LLVM может сделать с бинарным кодом?

Понять откуда он взялся...

“У вас проблема в инструкции по адресу 0xdeadbeef в модуле libfoo.so”

Понять откуда он взялся

“У вас проблема в инструкции по адресу 0xdeadbeef в модуле libfoo.so”

```
llvm::object::SectionedAddress address = {offset, llvm::object::SectionedAddress::UndefSection};
auto info = symbolizer.symbolizeCode(obj_name, address);
```

“У вас проблема в файле foo.cpp, функция bar, строка 42”

Понять откуда он взялся

“У вас проблема в инструкции по адресу 0xdeadbeef в модуле libfoo.so”

```
llvm::object::SectionedAddress address = {offset, llvm::object::SectionedAddress::UndefSection};
auto info = symbolizer.symbolizeCode(obj_name, address);
```

“У вас проблема в файле foo.cpp, функция bar, строка 42”

Деманглинг включается опцией

Z3fooIdEiRKT → int foo<double>(double const&)

Для поддержки inline есть symbolizeInlinedCode()

Показать по-человечески

“У вас проблема в инструкции 0xd5bf3b03

Показать по-человечески

“У вас проблема в инструкции 0xd5bf3b03”

```
std::unique_ptr<llvm::MCDisassembler> disasm(target->createMCDisassembler(*sti, ctx));
std::unique_ptr<llvm::MCInstPrinter> ip(target->createMCInstPrinter(
            llvm::Triple(triple_name), asm_dialect, *asm_info, *mii, *mri));
llvm::MCInst inst;    uint64_t size;
status = disasm->getInstruction(inst, size, code, address, log_stream);
ip->printInst(&inst, offset, "", *sti, out_stream);
```

“У вас проблема в инструкции dmb ish”

Да это же барьер!

https://developer.arm.com/documentation/100941/0101/Barriers

Показать по-человечески

“У вас проблема в инструкции 0xd5bf3b03”

```
std::unique_ptr<llvm::MCDisassembler> disasm(target->createMCDisassembler(*sti, ctx));
std::unique_ptr<llvm::MCInstPrinter> ip(target->createMCInstPrinter(
            llvm::Triple(triple_name), asm_dialect, *asm_info, *mii, *mri));
llvm::MCInst inst;   uint64_t size;
status = disasm->getInstruction(inst, size, code, address, log_stream);
ip->printInst(&inst, offset, "", *sti, out_stream);
```

“У вас проблема в инструкции dmb ish”

Да это же барьер!

https://developer.arm.com/documentation/100941/0101/Barriers

Очень важный класс

C++ объект, описывающий инструкцию
Полная информация – тип, регистры и прочее
Можно поанализировать:

```
unsigned opcode = inst.getOpcode();
auto first_op = inst.getOperand();

const MCInstrDesc &desc = mcii.get(opcode);
if (desc.isConditionalBranch())
..
```

llvm::MCInst inst;

llvm-mc – универсальный инструмент:

```
$ echo "0xCD 0x21" | llvm-mc –disassemble
int $33
$ echo "addl %eax, %ebx" | llvm-mc -show-encoding -show-inst
 .section __TEXT,__text,regular,pure_instructions
 addl %eax, %ebx            ## encoding: [0x01,0xc3]
                            ## <MCInst #66 ADD32rr
                            ##  <MCOperand Reg:29>
                            ##  <MCOperand Reg:29>
                            ##  <MCOperand Reg:27>>
```

И пример использования llvm::MCInst!

https://blog.llvm.org/2010/04/intro-to-llvm-mc-project.html
https://github.com/llvm/llvm-project/tree/main/llvm/tools/llvm-mc

Немножко посложнее

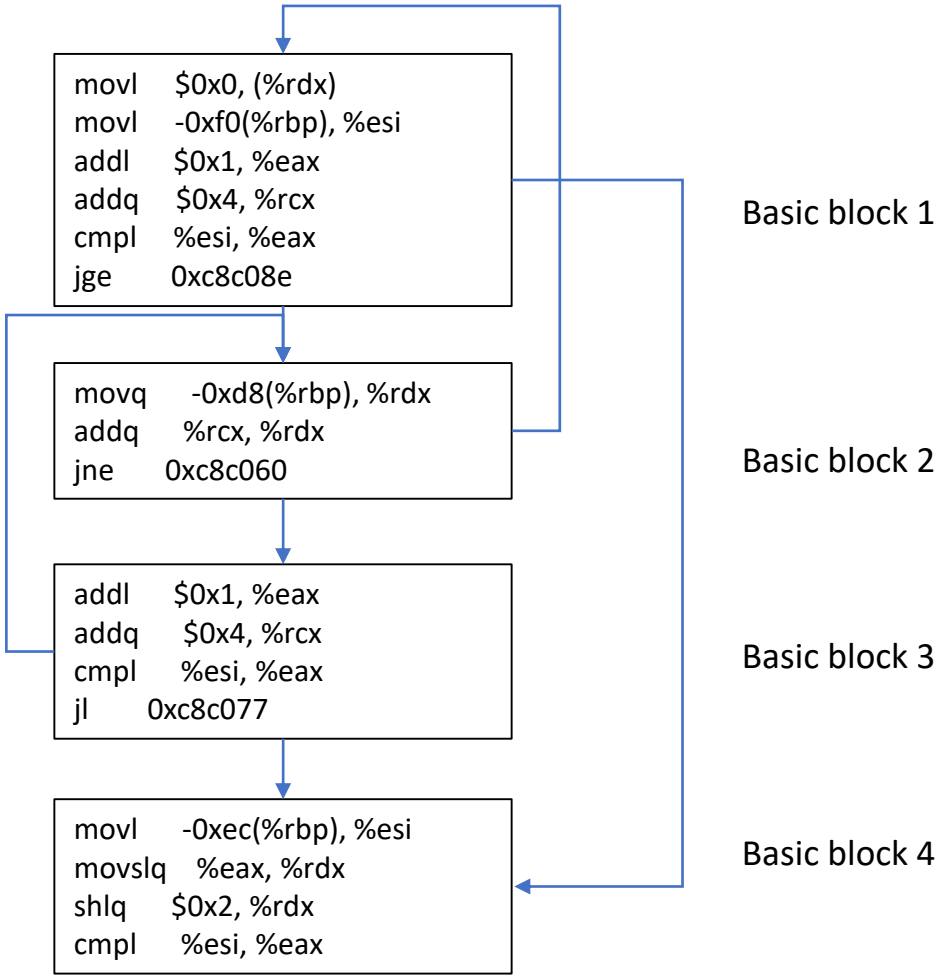
Найти структуру в потоке инструкций

```
0x000000c8c060      movl    $0x0, (%rdx)
0x000000c8c066      movl    -0xf0(%rbp), %esi
0x000000c8c06c      addl    $0x1, %eax
0x000000c8c06f      addq    $0x4, %rcx
0x000000c8c073      cmpl    %esi, %eax
0x000000c8c075      jge     0xc8c08e
0x000000c8c077      movq    -0xd8(%rbp), %rdx
0x000000c8c07e      addq    %rcx, %rdx
0x000000c8c081      jne     0xc8c060
0x000000c8c083      addl    $0x1, %eax
0x000000c8c086      addq    $0x4, %rcx
0x000000c8c08a      cmpl    %esi, %eax
0x000000c8c08c      jl      0xc8c077
0x000000c8c08e      movl    -0xec(%rbp), %esi
0x000000c8c094      movslq  %eax, %rdx
0x000000c8c097      shlq    $0x2, %rdx
0x000000c8c09b      cmpl    %esi, %eax
```

Сплошной поток – есть ли в нём какая нибудь структура?

Найти структуру в потоке инструкций

```
0x000000c8c060      movl    $0x0, (%rdx)
0x000000c8c066      movl    -0xf0(%rbp), %esi
0x000000c8c06c      addl    $0x1, %eax
0x000000c8c06f      addq    $0x4, %rcx
0x000000c8c073      cmpl    %esi, %eax
0x000000c8c075      jge     0xc8c08e
0x000000c8c077      movq    -0xd8(%rbp), %rdx
0x000000c8c07e      addq    %rcx, %rdx
0x000000c8c081      jne     0xc8c060
0x000000c8c083      addl    $0x1, %eax
0x000000c8c086      addq    $0x4, %rcx
0x000000c8c08a      cmpl    %esi, %eax
0x000000c8c08c      jl      0xc8c077
0x000000c8c08e      movl    -0xec(%rbp), %esi
0x000000c8c094      movslq  %eax, %rdx
0x000000c8c097      shlq    $0x2, %rdx
0x000000c8c09b      cmpl    %esi, %eax
```

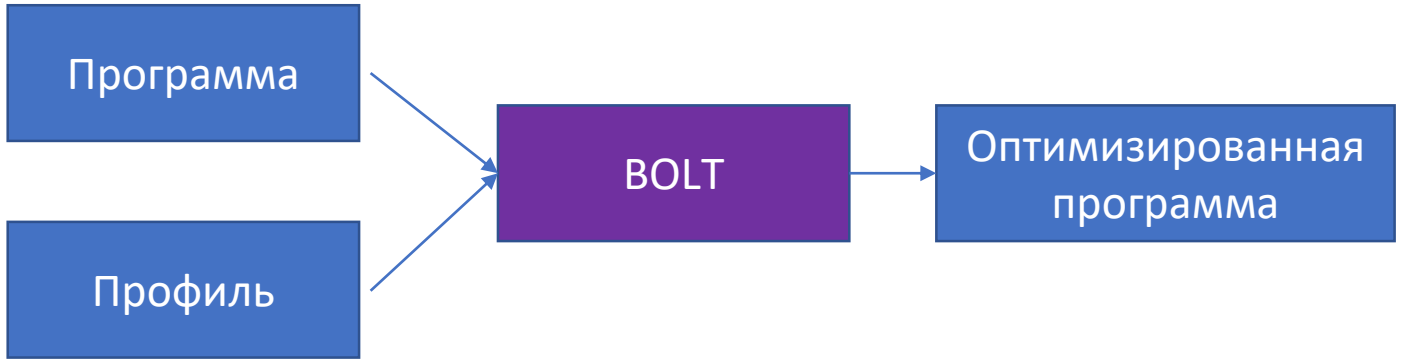


Надо построить Control Flow Graph – неужели LLVM это может?

LLVM BOLT – дополнительные детали

BOLT is a post-link optimizer developed to speed up large applications. It achieves the improvements by optimizing application's code layout based on execution profile gathered by sampling profiler, such as Linux perf tool.

https://github.com/llvm/llvm-project/blob/main/bolt/README.md



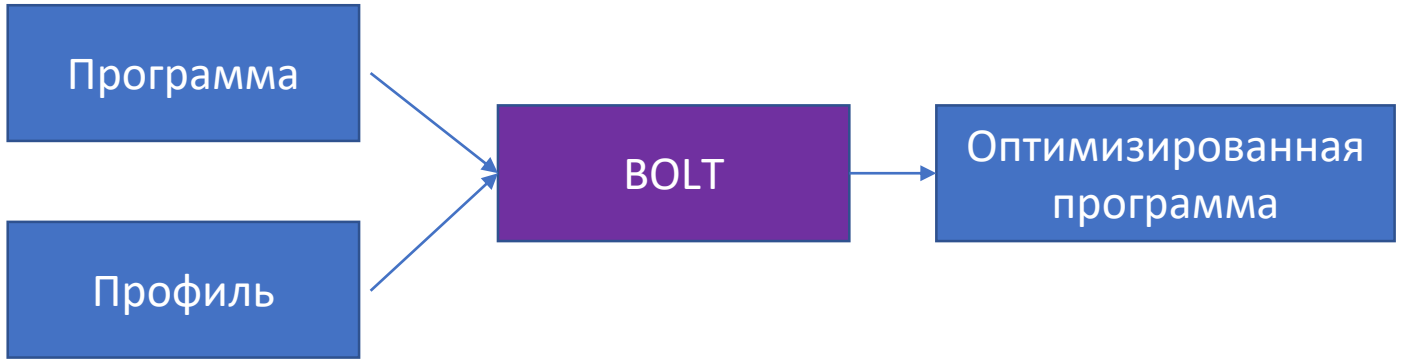
Разбирает бинарный код на куски

Собирает обратно с учётом профиля

LLVM BOLT – дополнительные детали

BOLT is a post-link optimizer developed to speed up large applications. It achieves the improvements by optimizing application's code layout based on execution profile gathered by sampling profiler, such as Linux perf tool.

https://github.com/llvm/llvm-project/blob/main/bolt/README.md



Тут есть нужные нам детальки!

Разбирает бинарный код на куски

Собирает обратно с учётом профиля

Переиспользуем и допиливаем

```
auto binary = llvm::object::createBinary(obj_name)->getBinary();
auto obj = llvm::dyn_cast<llvm::object::ObjectFile>(binary);
auto bc = llvm::bolt::BinaryContext::createBinaryContext(
      obj->makeTriple(), obj->getFileName(), nullptr, is_pic,
      llvm::DWARFContext::create(*obj), log_streams);
auto function = bc->getBinaryFunctionAtAddress(address);
for (auto basic_block : *function) {
   // Делаем что нужно
```

Дали имя бинарника и адрес, получили разбивку на базовые блоки

Этого достаточно?

Переиспользуем и допиливаем

```
auto binary = llvm::object::createBinary(obj_name)->getBinary();
auto obj = llvm::dyn_cast<llvm::object::ObjectFile>(binary);
auto bc = llvm::bolt::BinaryContext::createBinaryContext(
      obj->makeTriple(), obj->getFileName(), nullptr, is_pic,
      llvm::DWARFContext::create(*obj), log_streams);
auto function = bc->getBinaryFunctionAtAddress(address);
for (auto basic_block : *function) {
   // Делаем что нужно
```

Дали **имя бинарника** и адрес, получили разбивку на базовые блоки

Этого достаточно?

А если код не из бинарника?

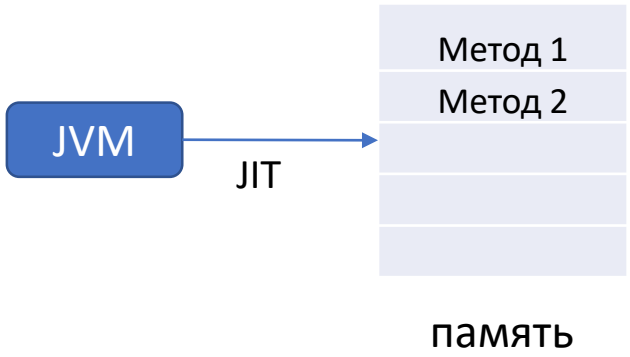
Переиспользуем и допиливаем

```
auto binary = llvm::object::createBinary(obj_name)->getBinary();
auto obj = llvm::dyn_cast<llvm::object::ObjectFile>(binary);
auto bc = llvm::bolt::BinaryContext::createBinaryContext(
      obj->makeTriple(), obj->getFileName(), nullptr, is_pic,
      llvm::DWARFContext::create(*obj), log_streams);
auto function = bc->getBinaryFunctionAtAddress(address);
for (auto basic_block : *function) {
   // Делаем что нужно
```

Дали имя бинарника и адрес, получили разбивку на базовые блоки

Этого достаточно?

А если код не из бинарника?



Можем сохранить в своём формате – но BOLT хочет ELF с заголовками

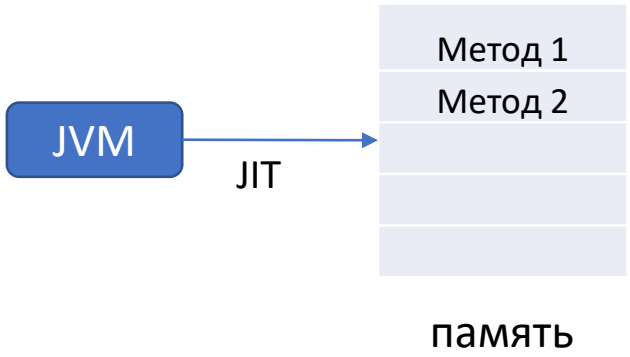
Переиспользуем и допиливаем

```
auto binary = llvm::object::createBinary(obj_name)->getBinary();
auto obj = llvm::dyn_cast<llvm::object::ObjectFile>(binary);
auto bc = llvm::bolt::BinaryContext::createBinaryContext(
      obj->makeTriple(), obj->getFileName(), nullptr, is_pic,
      llvm::DWARFContext::create(*obj), log_streams);
auto function = bc->getBinaryFunctionAtAddress(address);
for (auto basic_block : *function) {
   // Делаем что нужно
```

Дали имя бинарника и адрес, получили разбивку на базовые блоки

Этого достаточно?

А если код не из бинарника?



Можем сохранить в своём формате – но BOLT хочет ELF с заголовками

```
class JIT_object_file: public llvm::object::ObjectFile
{ … }
```

Создали свой класс для поддержки JIT
Отвязали BOLT от ELF

Код пока только в локальном бранче...

Во все тяжкие

Найти источник проблем

```
0x000000c8c060      movl    $0x0, (%rdx)
0x000000c8c066      movl    -0xf0(%rbp), %esi
0x000000c8c06c      addl    $0x1, %eax
0x000000c8c06f      addq    $0x4, %rcx
0x000000c8c073      cmpl    %esi, %eax
0x000000c8c075      jge     0xc8c08e
0x000000c8c077      movq    -0xd8(%rbp), %rdx
0x000000c8c07e      addq    %rcx, %rdx
0x000000c8c081      jne     0xc8c060
0x000000c8c083      addl    $0x1, %eax
0x000000c8c086      addq    $0x4, %rcx
0x000000c8c08a      cmpl    %esi, %eax
0x000000c8c08c      jl      0xc8c077
0x000000c8c08e      movl    -0xec(%rbp), %esi
```

Как и зачем решать проблемы с производительностью без профилировки?

Как может помочь LLVM?

Почему не хватает профилировщиков?

Address	Symbol: **org.apache.spark.unsafe.Platform::copyMemory**		Instructions
	Basic Block 1:		
0x000000001000	movl	%eax, -0x14000(%rsp)	648,000,972
0x000000001007	pushq	%rbp	216,000,324
0x000000001008	**movq**	%rsp, %rbp	288,000,432
0x00000000100b	subq	$0x70, %rsp	612,000,918
0x00000000100f	**movq**	%rdx, (%rsp)	396,000,594
0x000000001013	cmpq	%rdx, %r8	72,000,108
0x000000001016	**jl**	0x00000000119e	

Смотрим на perf профиль для одного базового блока
На самом деле все инструкции выполнялись вместе
Почему числа разные?

Почему не хватает профилировщиков?

Address	Symbol: **org.apache.spark.unsafe.Platform::copyMemory**		Instructions
	Basic Block 1:		
0x000000001000	movl	%eax, -0x14000(%rsp)	648,000,972
0x000000001007	pushq	%rbp	216,000,324
0x000000001008	**movq**	%rsp, %rbp	288,000,432
0x00000000100b	subq	$0x70, %rsp	612,000,918
0x00000000100f	**movq**	%rdx, (%rsp)	396,000,594
0x000000001013	cmpq	%rdx, %r8	72,000,108
0x000000001016	**jl**	0x00000000119e	

Смотрим на perf профиль для одного базового блока
На самом деле все инструкции выполнялись вместе
Почему числа разные?

Ответ – профилировщик работает статистически
Можно верить суммарным цифрам на функцию или цикл
Нельзя верить цифрам на инструкцию!

Есть технологии поточнее (Intel PT, AMD IBS, ARM SPE), но и их недостаточно

Почему не хватает профилировщиков?

Address	Symbol: **org.apache.spark.unsafe.Platform::copyMemory**		Instructions
	Basic Block 1:		
0x000000001000	movl	%eax, -0x14000(%rsp)	648,000,972
0x000000001007	pushq	%rbp	216,000,324
0x000000001008	**movq**	%rsp, %rbp	288,000,432
0x00000000100b	subq	$0x70, %rsp	612,000,918
0x00000000100f	**movq**	%rdx, (%rsp)	396,000,594
0x000000001013	cmpq	%rdx, %r8	72,000,108
0x000000001016	**jl**	0x00000000119e	

Смотрим на perf профиль для одного базового блока
На самом деле все инструкции выполнялись вместе
Почему числа разные?

Ответ – профилировщик работает статистически
Можно верить суммарным цифрам на функцию или цикл
Нельзя верить цифрам на инструкцию!

Что делать?

Анализировать код

Неужели вручную?

Есть технологии поточнее (Intel PT, AMD IBS, ARM SPE), но и их недостаточно

Анализаторы кода - Intel

IACA 2009-2019

IACA (the Intel Architecture Code Analyzer) is a (**2019: end-of-life**) freeware, closed-source static analysis tool made by Intel to statically analyze the scheduling of instructions when executed by modern Intel processors.
https://stackoverflow.com/questions/26021337/what-is-iaca-and-how-do-i-use-it

Только для процессоров Intel
Сейчас не поддерживается

```
Throughput Analysis Report
--------------------------
Block Throughput: 5.00 Cycles       Throughput Bottleneck: Dependency chains
Loop Count:  23
Port Binding In Cycles Per Iteration:
---------------------------------------------------------------------------------------
|  Port  |  0  -  DV  |  1  |  2  -  D  |  3  -  D  |  4  |  5  |  6  |  7  |
---------------------------------------------------------------------------------------
| Cycles |  2.5   0.0  |  2.5  |  2.0   2.0  |  2.0   1.0  |  1.0  |  4.0  |  2.0  |  0.0  |
---------------------------------------------------------------------------------------


DV - Divider pipe (on port 0)
D - Data fetch pipe (on ports 2 and 3)
F - Macro Fusion with the previous instruction occurred
* - instruction micro-ops not bound to a port
^ - Micro Fusion occurred
# - ESP Tracking sync uop was issued
@ - SSE instruction followed an AVX256/AVX512 instruction, dozens of cycles penalty is expected
X - instruction not supported, was not accounted in Analysis

| Num Of   |                    Ports pressure in cycles                     |     |
|  Uops    |  0  - DV  |  1  |  2  -  D  |  3  -  D  |  4  |  5  |  6  |  7  |
---------------------------------------------------------------------------------
|   1      |           |     | 1.0   1.0 |           |     |     |     |     | mov r10, qword ptr [rbp+0x170]
|   1      |           | 1.0 |           |           |     |     |     |     | lea r9d, ptr [r8*4]
|   1      |           |     |           |           |     |     | 1.0 |     | movsxd r9, r9d
|   1      |           |     |           |           |     |     | 1.0 |     | inc r8d
|   1      |           |     |           | 1.0   1.0 |     |     |     |     | mov r11, qword ptr [rbp+0x178]
|   1      |           |     | 1.0   1.0 |           |     |     |     |     | vmovups xmm3, xmmword ptr [r10+r9*4]
|   1      |           |     |           |           |     | 1.0 |     |     | vpslldq xmm2, xmm3, 0x4
|   1      |           |     |           |           |     | 1.0 |     |     | vpslldq xmm4, xmm3, 0x8
|   1      | 1.0       |     |           |           |     |     |     |     | vaddps xmm6, xmm2, xmm3
|   1      |           |     |           |           |     | 1.0 |     |     | vpslldq xmm5, xmm3, 0xc
|   1      | 0.5       | 0.5 |           |           |     |     |     |     | vaddps xmm7, xmm4, xmm5
|   1      | 0.5       | 0.5 |           |           |     |     |     |     | vaddps xmm8, xmm6, xmm7
|   1      | 0.5       | 0.5 |           |           |     |     |     |     | vaddps xmm9, xmm8, xmm0
|   1      |           |     |           |           |     | 1.0 |     |     | vshufps xmm0, xmm9, xmm9, 0xff
|   2      |           |     |           | 1.0       | 1.0 |     |     |     | vmovups xmmword ptr [r11+r9*4], xmm9
|   1*     |           |     |           |           |     |     |     |     | cmp r8d, esi
|   0*F    |           |     |           |           |     |     |     |     | jl 0xffffffffffffffb0
Total Num Of Uops: 17
```

```
it|in|Dissasembly                                 :0123456789012345678901234567890123456
 0| 0|mov r10, qword ptr [rbp+0x170]              :          |         |         |
 0| 0|    TYPE_LOAD (1 uops)                      :s---deeeew----R-------p         |
 0| 1|lea r9d, ptr [r8*4]                         :          |         |         |
 0| 1|    TYPE_OP (1 uops)                        :sdw----------R-------p         |
 0| 2|movsxd r9, r9d                              :          |         |         |
 0| 2|    TYPE_OP (1 uops)                        :A-dw----------R-------p         |
 0| 3|inc r8d                                     :          |         |         |
 0| 3|    TYPE_OP (1 uops)                        :sdw----------R-------p         |
 0| 4|mov r11, qword ptr [rbp+0x178]              :          |         |         |
 0| 4|    TYPE_LOAD (1 uops)                      : s---deeeew----R-------p        |
 0| 5|vmovups xmm3, xmmword ptr [r10+r9*4]        :          |         |         |
 0| 5|    TYPE_LOAD (1 uops)                      : A-------deeeeew----R-------p   |
 0| 6|vpslldq xmm2, xmm3, 0x4                     :          |         |         |
 0| 6|    TYPE_OP (1 uops)                        : A-------------dw----R-------p  |
 0| 7|vpslldq xmm4, xmm3, 0x8                     :          |         |         |
 0| 7|    TYPE_OP (1 uops)                        : A-------------cdw----R-------p
 0| 8|vaddps xmm6, xmm2, xmm3                     :          |         |         |
 0| 8|    TYPE_OP (1 uops)                        :  A-------------deeew----R-------p
 0| 9|vpslldq xmm5, xmm3, 0xc                     :          |         |         |
 0| 9|    TYPE_OP (1 uops)                        :  A------------ccdw------R-------p
```

https://www.intel.com/content/dam/develop/external/us/en/documents/intel-architecture-code-analyzer-3-0-users-guide-157552.pdf

Анализаторы кода - community

IACA 2009-2019

IACA (the Intel Architecture Code Analyzer) is a (**2019: end-of-life**) freeware, closed-source static analysis tool made by Intel to statically analyze the scheduling of instructions when executed by modern Intel processors.
https://stackoverflow.com/questions/26021337/what-is-iaca-and-how-do-i-use-it

Только для процессоров Intel
Сейчас не поддерживается

OSACA
Open Source Architecture Code Analyzer
For an innermost loop kernel in assembly, this tool allows automatic instruction fetching of assembly code and automatic runtime prediction including throughput analysis and detection for critical path and loop-carried dependencies.
https://github.com/RRZE-HPC/OSACA

uiCA
uops.info Code Analyzer
uiCA is a simulator that can predict the throughput of basic blocks on recent Intel microarchitectures. In addition to that, it also provides insights into how the code is executed.
https://github.com/andreas-abel/uiCA

Ithemal
Instruction THroughput Estimator using MAchine Learning
Ithemal is a data driven model for predicting throughput of a basic block of x86-64 instructions.
https://github.com/ithemal/Ithemal

Мы точно на докладе про LLVM?

Анализаторы кода - community

IACA 2009-2019

IACA (the Intel Architecture Code Analyzer) is a (**2019: end-of-life**) freeware, closed-source static analysis tool made by Intel to statically analyze the scheduling of instructions when executed by modern Intel processors.
https://stackoverflow.com/questions/26021337/what-is-iaca-and-how-do-i-use-it

Только для процессоров Intel
Сейчас недоступен

OSACA
Open Source Architecture Code Analyzer
For an innermost loop kernel in assembly, this tool allows automatic instruction fetching of assembly code and automatic runtime prediction including throughput analysis and detection for critical path and loop-carried dependencies.
https://github.com/RRZE-HPC/OSACA

uiCA
uops.info Code Analyzer
uiCA is a simulator that can predict the throughput of basic blocks on recent Intel microarchitectures. In addition to that, it also provides insights into how the code is executed.
https://github.com/andreas-abel/uiCA

Ithemal
Instruction THroughput Estimator using MAchine Learning
Ithemal is a data driven model for predicting throughput of a basic block of x86-64 instructions.
https://github.com/ithemal/Ithemal

Мы точно на докладе про LLVM?

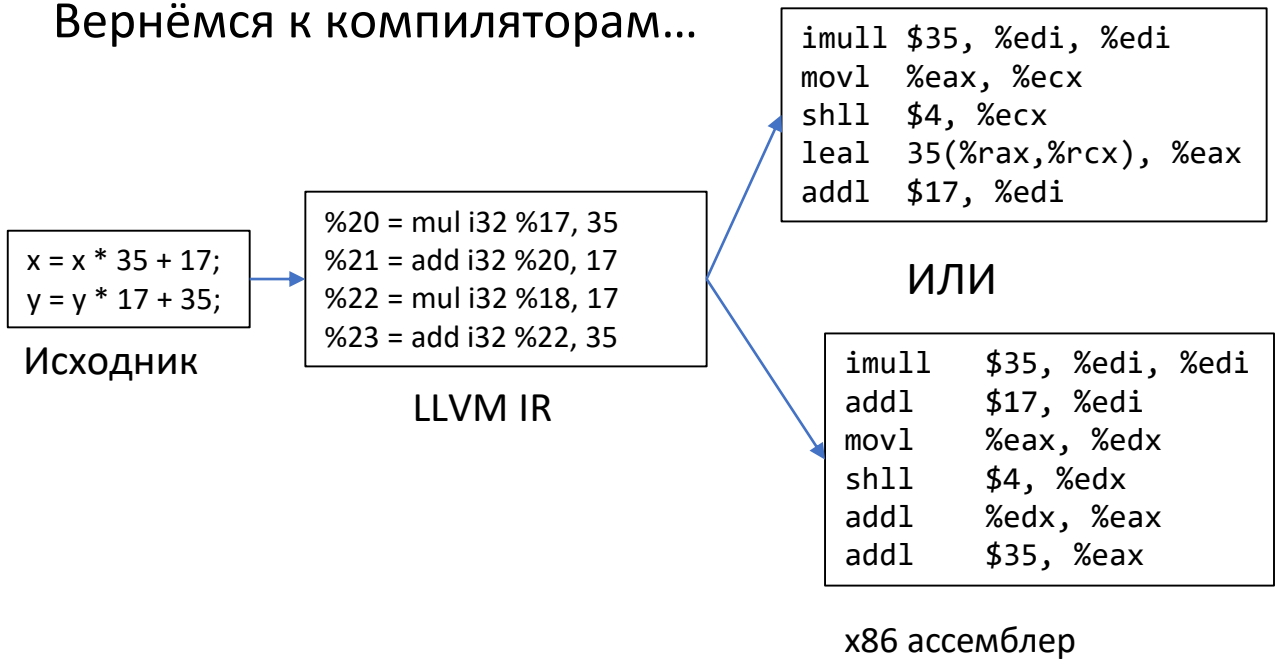
Да – в LLVM есть свой анализатор (LLVM Machine Code Analyzer)!

https://github.com/llvm/llvm-project/tree/main/llvm/lib/MCA

Откуда LLVM знает детали процессора?

Откуда LLVM знает детали процессора?

Вернёмся к компиляторам…



Как выбрать?

Откуда LLVM знает детали процессора?

Вернёмся к компиляторам…

```
x = x * 35 + 17;
y = y * 17 + 35;
```

Исходник

```
%20 = mul i32 %17, 35
%21 = add i32 %20, 17
%22 = mul i32 %18, 17
%23 = add i32 %22, 35
```

LLVM IR

```
imull $35, %edi, %edi
movl  %eax, %ecx
shll  $4, %ecx
leal  35(%rax,%rcx), %eax
addl  $17, %edi
```

ИЛИ

```
imull   $35, %edi, %edi
addl    $17, %edi
movl    %eax, %edx
shll    $4, %edx
addl    %edx, %eax
addl    $35, %eax
```

x86 ассемблер

Как выбрать?

Ответ – ***машинная модель***

```
def SapphireRapidsModel : SchedMachineModel {
  // SapphireRapids can allocate 6 uops per cycle.
  let IssueWidth = 6; // Based on allocator width.
  let MicroOpBufferSize = 512; // Based on the reorder buffer.
…
// SapphireRapids can issue micro-ops to 12 different ports in one cycle.
def SPRPort00 : ProcResource<1>;
def SPRPort01 : ProcResource<1>;
…
def SPRWriteResGroup547 : SchedWriteRes<[SPRPort00, SPRPort00_01_05, SPRPort02_03_11]> {
  let ReleaseAtCycles = [2, 1, 1];
  let Latency = 40;
  let NumMicroOps = 4;
}
def : InstRW<[SPRWriteResGroup547, ReadAfterVecYLd], (instregex "^VSQRTPHZ256m(b?)$")>;
…
```

https://github.com/llvm/llvm-project/blob/main/llvm/lib/Target/X86/X86SchedSapphireRapids.td

Общие параметры процессора
Информация по исполнительным устройствам
Информация о каждой инструкции

Добавляют и обновляют сами производители (чтобы clang генерировал код получше)

И что с этим знанием делать?

Уточняем описание инструкции

Информация о процессоре

llvm::MCInst Inst;

```
// Создаём билдер
mca::InstrBuilder ib(…); // используем машинную модель

// Преобразуем инструкцию
mca::Instruction mi = ib.createInstruction(inst, Instruments);
```

```
class Instruction {
   const InstrDesc &Desc;

   // текущее состояние
};
```

```
class InstrDesc {
   Reads; // что на входе
   Writes; // что на выходе
   Resources; // как используем процессор
   MaxLatency; // время от входа до выхода
   …
};
```

Может переиспользоваться

Уточняем описание инструкции

Информация о процессоре

llvm::MCInst Inst;

```
// Создаём билдер
mca::InstrBuilder ib(…); // используем машинную модель

// Преобразуем инструкцию
mca::Instruction mi = ib.createInstruction(inst, Instruments);
```

```
class Instruction {
    const InstrDesc &Desc;

    // текущее состояние
};
```

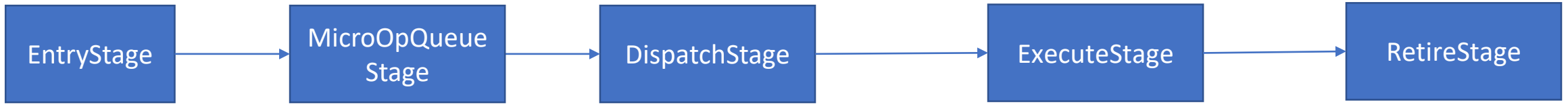
```
class InstrDesc {
    Reads; // что на входе
    Writes; // что на выходе
    Resources; // как используем процессор
    MaxLatency; // время от входа до выхода
    …
};
```

Может переиспользоваться

Можем моделировать – только чем?

Как устроена модель процессора

Стадии пайплайна (наследники llvm::mca::Stage)



Потребляем поток инструкций и скрамливаем дальше, пока можно

Копим микроинструкции

Проверяем готовность операндов и наличие свободных ресурсов, передаём дальше

Порядок передачи может отличаться от входного (out-of-order)!

Держим инструкции на ресурсах процессора столько, сколько надо, отдаём дальше

Порядок – как получится

Заканчиваем обработку инструкций в исходном порядке (ждём пока исполнятся все предыдущие)

Все параметры – из машинной модели для конкретного процессора

Собираем вместе

```
auto P = MCA.createDefaultPipeline(…);

Под капотом:

StagePipeline->appendStage(std::move(Fetch));
 if (Opts.MicroOpQueueSize)
  StagePipeline->appendStage(std::make_unique<MicroOpQueueStage>(
     Opts.MicroOpQueueSize, Opts.DecodersThroughput));
 StagePipeline->appendStage(std::move(Dispatch));
 StagePipeline->appendStage(std::move(Execute));
 StagePipeline->appendStage(std::move(Retire));
```

Как узнать что происходит?

```
mca::PipelinePrinter Printer(*P, ...);
Printer.addView(std::make_unique<mca::DispatchStatistics>());
// Есть и другие view, можно создать своё

Под капотом:

class HWEventListener {
public:
 virtual void onCycleBegin() {}
 virtual void onCycleEnd() {}
 virtual void onEvent(…) {}
 …
};

Получаем нотификации о работе процессора на каждом
такте
```

Можно запускать – mca::runPipeline(*P)

Что на выходе?

Запускаем:
llvm-mca -mcpu=tsv110 --iterations=100 --timeline test0.s

Конкретная модель ядра процессора

Моделируем выполнение в цикле

```
Iterations:        100
Instructions:      1200
Total Cycles:      374
Total uOps:        1200
```

Общая информация

Детально по тактам

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeER   .    .    .   ldrb   w1, [x7, x0]
[0,1]     DeE---R   .    .    .   add    w3, w3, #6
[0,2]     DeE---R   .    .    .   mov    w6, w0
[0,3]     D====eER  .    .    .   cmp    w1, #191
[0,4]     .D====eER .    .    .   cset   w2, le
[0,5]     .DeE----R .    .    .   cmp    w0, #4
[0,6]     .D=eE---R .    .    .   cset   w5, eq
```

Исполняем инструкцию

Ждём предыдущие инструкции

Ждём готовности данных (w1)

Арифметика и условия

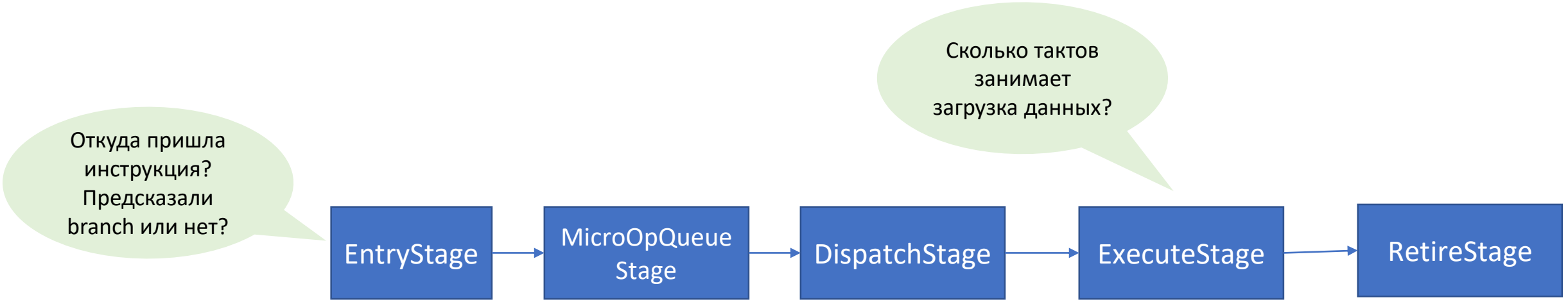
Чтение/запись

```
Resource pressure by instruction:
[0.0]  [0.1]  [1]    [2]    [3]    [4]    [5]    [6]    Instructions:
 -      -      -      -      -     0.50   0.50    -     ldrb    w1, [x7, x0]
 -     0.66   0.34    -      -      -      -      -     add     w3, w3, #6
0.32   0.68    -      -      -      -      -      -     mov     w6, w0
0.66   0.34    -      -      -      -      -      -     cmp     w1, #191
 -      -     1.00    -      -      -      -      -     cset    w2, le
0.03   0.97    -      -      -      -      -      -     cmp     w0, #4
0.33   0.65   0.02    -      -      -      -      -     cset    w5, eq
```

Использование ресурсов

Процессор под микроскопом...но всё ли мы видим?

Ограничения MCA



Можем затормозить здесь (по “научному” – frontend-bound)

MCA не моделирует!

Время может зависеть от внешних ресурсов (память, внутренние шины)

Нет динамической информации
Нет кода, обрабатывающего динамическую информацию

Обрабатываем доступы в память

```
// Добавляем гибкости
InstrBuilder createInstruction(const MCInst &MCI, …,
  std::function<void(InstrDesc&)> Customizer = std::function<void(InstrDesc&)>());
…
Desc = Customizer ? createInstrDescImpl(MCI, IVec, Customizer) : // создаём уникальное описание
                                  getOrCreateInstrDesc(MCI, IVec);
…
// Храним отдельно, запрещая переиспользование
std::vector<std::unique_ptr<const InstrDesc>> CustomDescriptors;
…
// Модифицируем инструкцию
 if (Customizer) {
    Customizer(*ID);
    return *CustomDescriptors.emplace_back(std::move(ID));
  }


// Конкретно для памяти
desc.MaxLatency = memory_access_latency;
for (auto& write : desc.Writes) write.Latency = memory_access_latency;
```

Что получилось

Было

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeER   .    .   .   ldrb   w1, [x7, x0]
[0,1]     DeE---R   .    .   .   add    w3, w3, #6
[0,2]     DeE---R   .    .   .   mov    w6, w0
[0,3]     D====eER  .    .   .   cmp    w1, #191
[0,4]     .D====eER .    .   .   cset   w2, le
[0,5]     .DeE----R .    .   .   cmp    w0, #4
[0,6]     .D=eE---R .    .   .   cset   w5, eq
```

Меняем latency на 8

А если первая инструкция идёт в L2?

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeeeeeER    .    .   .   ldrb  w1, [x7, x0]
[0,1]     DeE-------R    .    .   .   add   w3, w3, #6
[0,2]     DeE-------R    .    .   .   mov   w6, w0
[0,3]     D========eER   .    .   .   cmp   w1, #191
[0,4]     .D========eER  .    .   .   cset  w2, le
[0,5]     .DeE--------R  .    .   .   cmp   w0, #4
[0,6]     .D=eE-------R  .    .   .   cset  w5, eq
```

Что с производительностью?

Что получилось

Было

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeER   .    .   .   ldrb   w1, [x7, x0]
[0,1]     DeE---R   .    .   .   add    w3, w3, #6
[0,2]     DeE---R   .    .   .   mov    w6, w0
[0,3]     D====eER  .    .   .   cmp    w1, #191
[0,4]     .D====eER .    .   .   cset   w2, le
[0,5]     .DeE----R .    .   .   cmp    w0, #4
[0,6]     .D=eE---R .    .   .   cset   w5, eq
```

Меняем latency на 8

А если первая инструкция идёт в L2?

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeeeeeER    .    .   .   ldrb   w1, [x7, x0]
[0,1]     DeE-------R    .    .   .   add    w3, w3, #6
[0,2]     DeE-------R    .    .   .   mov    w6, w0
[0,3]     D========eER   .    .   .   cmp    w1, #191
[0,4]     .D========eER  .    .   .   cset   w2, le
[0,5]     .DeE--------R  .    .   .   cmp    w0, #4
[0,6]     .D=eE-------R  .    .   .   cset   w5, eq
```

Что с производительностью?

```
Iterations:        100
Instructions:      1200
Total Cycles:      374
Total uOps:        1200
```

```
Iterations:        100
Instructions:      1200
Total Cycles:      375
Total uOps:        1200
```

Практически не поменялась!

Что получилось

Меняем latency на 8

Было

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeER   .    .   .   ldrb   w1, [x7, x0]
[0,1]     DeE---R   .    .   .   add    w3, w3, #6
[0,2]     DeE---R   .    .   .   mov    w6, w0
[0,3]     D====eER  .    .   .   cmp    w1, #191
[0,4]     .D====eER .    .   .   cset   w2, le
[0,5]     .DeE----R .    .   .   cmp    w0, #4
[0,6]     .D=eE---R .    .   .   cset   w5, eq
```

А если первая инструкция идёт в L2?

```
Timeline view:
                    0123456789
Index     0123456789

[0,0]     DeeeeeeeeER    .    .   .   ldrb   w1, [x7, x0]
[0,1]     DeE-------R    .    .   .   add    w3, w3, #6
[0,2]     DeE-------R    .    .   .   mov    w6, w0
[0,3]     D========eER   .    .   .   cmp    w1, #191
[0,4]     .D========eER  .    .   .   cset   w2, le
[0,5]     .DeE--------R  .    .   .   cmp    w0, #4
[0,6]     .D=eE-------R  .    .   .   cset   w5, eq
```

Что с производительностью?

```
Iterations:        100
Instructions:      1200
Total Cycles:      374
Total uOps:        1200
```

```
Iterations:        100
Instructions:      1200
Total Cycles:      375
Total uOps:        1200
```

Практически не поменялась!

Latency = 300

```
Iterations:        100
Instructions:      1200
Total Cycles:      3043
Total uOps:        1200
```

А вот теперь заметно!

Что дальше

- Добавить недостающее
  - Instruction cache, Branch prediction…
- Сделать вывод понятнее
  - чтобы обычный программист на C++ разобрался
- Ещё идеи?

Сделаем – расскажем ☺

Заключение

- LLVM – не фреймворк, а конструктор
  - Переиспользуйте, меняйте, делитесь изменениями!
- Можно анализировать производительность без профилировки
  - с помощью LLVM MCA
- Модифицировать код LLVM не так страшно!