solarwinds msp

Что НЕ/ПЛОХО оптимизирует компилятор

или почему всё ещё надо думать, когда пишешь код

Александр Зайцев

Просто разработчик в Solarwinds. Люблю C++. В свободное время любитель покопаться в Open Source проектах. Постоянный посетитель багтрекеров компиляторов. Люблю когда компилятор хорошо оптимизирует. Расстраиваюсь, когда оптимизирует плохо.

Фазы компиляции

Как мы обычно оптимизируем код

solarwinds msp

- Пишем код
- Включаем O2/O3/O-что-нибудь
  - Подбираем флаги компиляции более внимательно и смотрим на результат
- Включаем march/mtune
- Включаем Link-Time Optimization (LTO)
- Применяем Profile-Guided Optimization (PGO)
- Используем доп. средства (например, Bolt)
- Радуемся!

Как мы обычно “оптимизируем” код

solarwinds msp

- Пишем код
- Включаем O2/O3/O-что-нибудь
  - Подбираем флаги компиляции более внимательно и смотрим на результат
- Включаем march/mtune
- Включаем Link-Time Optimization (LTO)
- Применяем Profile-Guided Optimization (PGO)
- Используем доп. средства (например, Bolt)
- Радуемся!

<u>Всю</u> грязную работу за нас делает <u>идеальный</u> компилятор!

* Но идеальных компиляторов нет :)

Как мы обычно оптимизируем код: проблема

solarwinds msp

- Скомпилировали код
- Медленно работает
- Применили всё с предыдущего слайда
- Медленно работает
- Понимаем, что придётся думать
- Профилируем, находим узкие места нашей программы
- **Переписываем**
- Повторяем до тех пор, пока нас не будет всё устраивать

Как мы обычно оптимизируем код: проблема

solarwinds msp

- Скомпилировали код
- Медленно работает
- Применили всё с предыдущего слайда
- Медленно работает
- Понимаем, что придётся думать
- Профилируем, находим узкие места нашей программы
- **Переписываем**
- Повторяем до тех пор, пока нас не будет всё устраивать

Проблемы

- О УЖАС! Компилятор не догадался сделать здесь X, а вот там Y!
- Это происходит постоянно/на какой-то определённой платформе/на каком-то определённом коде и т.д.
- Нам приходится/заставляют как-то оптимизировать
- Мы хотим меньше работать и поэтому недовольны

Проблемы компилятора

solarwinds msp

- Нужно превратить исходный код во что-то (IR, ASM, и т.д.)
- Скорее всего нужно оптимизировать по каким-то параметрам (скорость, размер)
- Нужно оптимизировать хорошо :)
- Нужно оптимизировать за **разумное** время
  - Разные оптимизации занимают разное время и разный вклад в итоговый прирост производительности
  - Хотите контролировать время компиляции? https://github.com/ldionne/metabench
- Компиляторы тоже далеко не идеальны - их тоже пишут люди
  - Компиляторы очень сложны => легко ошибиться
- Не успевает за изменениями в “железе”

Динамические аллокации

solarwinds msp

```
void f()
{
    int* ptr = new int[5];
    ptr[0] = 42;
    delete [] ptr;
}
```

gcc 8.2 -std=c++17 -O3

```
f():
sub rsp, 8
mov edi, 20
call operator new[](unsigned long)
mov DWORD PTR [rax], 42
mov rdi, rax
add rsp, 8
jmp operator delete[](void*)
```

clang 6 -std=c++17 -O3

- Аллокации - это может быть долго
- Начиная с C++14 можно оптимизировать (читай выкидывать) аллокации памяти
- Компиляторы пока что не/плохо умеют такие оптимизации проводить
- Это сразу выкидывает все оптимизации со стандартными динамически аллоцирующими контейнерами
- Реализация находится в зачаточном состоянии
- Полезно знать - Clang делает это и до C++14 :)

Предвычисления по время компиляции

solarwinds msp

```
bool foo()
{
    std::array<int,5> arr{5,4,3,2,1};
    std::sort(arr.begin(), arr.end());
    std::stable_sort(arr.begin(), arr.end());
    return std::is_sorted(arr.begin(), arr.end());
}
```

gcc и clang:
Много-много кода :(

- Написано много кода без constexpr - его тоже надо оптимизировать
- Больше на этапе компиляции - меньше во время выполнения
- Компиляторы имеют свои внутренние лимиты на вычисления во время компиляции
- Не можем вычислить всё во время компиляции - время тоже имеет значение

Оптимизация математических вычислений

solarwinds msp

GCC, файл match.pd, фрагмент

```
/* X * 1, X / 1 -> X.  */
(for op (mult trunc_div ceil_div
floor_div round_div exact_div)
  (simplify
    (op @0 integer_onep)
    (non_lvalue @0)))
```

- Включается с помощью -ffast-math (или аналогов)
- Как далеко мы должны оптимизировать?
- Сейчас все правила написаны руками
  - Но все правила ведь не напишешь :)
- Реализации:
  - GCC - https://github.com/gcc-mirror/gcc/blob/master/gcc/match.pd
  - LLVM - https://github.com/llvm-mirror/llvm/tree/master/lib/Transforms/InstCombine

Оптимизация математических вычислений

solarwinds msp

- Компилятор не имеет представления о наших правилах математики
- Ручная запись - **жалкая** попытка научить его хоть чему-то
- Оптимизации **требуют** верификации
  - Компилятор ведь должен производить только правильные оптимизации
- Нужна интеграция движков, которые привнесут математические правила в компилятор
  - Souper: https://github.com/google/souper

Оптимизация по контрактам

solarwinds msp

```
template <typename T>
T func(T a, T b)
[[expects: a > b]] {
    return a + b;
}
```

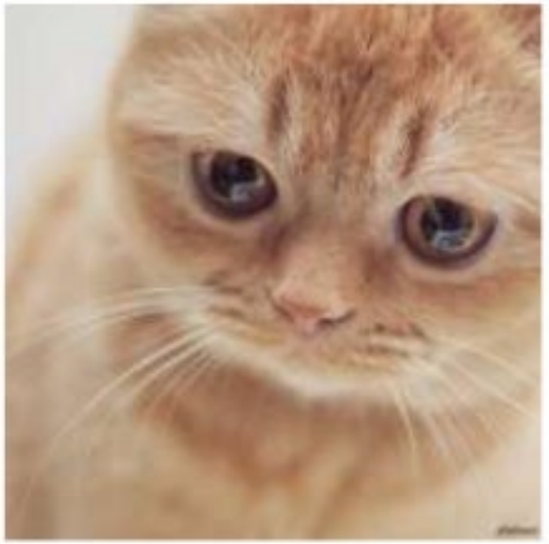
Играться вот тут:

- Контракты - **языковое** средство, с помощью которого вы можете описать пред/постусловия для вашей сущности и контролировать состояние в ходе работы
- Компиляторы могут использовать контракты для оптимизации (но они этого не делают) - реализации контрактов нигде нет
  - Ну почти нигде - https://github.com/arcosuc3m/clang-contracts
- Начиная с C++20 у нас есть контракты :)
- Контракты были и раньше, только компилятор про них ничего не знал

Пример с контрактами

```
int foo(std::vector<int> v) {
    std::sort(v.begin(), v.end());
    std::nth_element(v.begin(), v.begin(), v.end());
    return v[0];
}
```

Имеем вызов обеих функций :(



- На текущий момент нет информации, что функция делает **семантически**
- Сжатие функций происходит в основном за счёт инлайнинга
  - В сложных случаях (как этот) не работает
- Как помочь - ждать реализации в оптимизации в компиляторах и размечать функции

Алгоритмические оптимизации

```
int foo(std::vector<int> v) {
    std::sort(v.begin(), v.end());
    return v[0];
}
```

- Должен ли компилятор таким заниматься?
  - Увеличивается время анализа
  - Программа ускоряется
  - Проводить такой анализ очень сложно
- Сменить алгоритм на другой, решающий эту же задачу, только быстрее/с меньшим потреблением памяти
- Для развлечения: придумайте случаи, где вы можете придумать быстрее, проверьте компилятор на смышлёность и откройте баг :) (если я вас ещё не опередил :)

Оптимизации использования контейнеров

solarwinds msp

```
int foo(int n){
    std::vector<int> v(5);
    for(auto& d : v) d = rand();
    std::sort(v.begin(), v.end()); // чтобы не догадался :)
    int result = 0;
    for(auto& d : v) result += d;
    return result;
}
```

gcc && clang:

```
bla-bla-bla
call operator new(unsigned long) ; серьёзно?
bla-bla-bla
```

- Программисты могут ошибаться при выборе контейнера
- Компилятор **должен** оптимизировать такие случаи
  - Только в случае, если он может доказать, что иначе будет быстрее

Примеры:

- std::vector -> boost::small_vector -> std::array - Small Vector Optimization :)
- std::list -> std::forward_list
- std::map -> std::set
- Только там, где оптимизация не приведёт к изменению наблюдаемого поведения

Алиасинг

```
void foo(int* a, int* b, int* c) {
    *a += *c;
    *b += *c;
}
```

Плохо

```
foo:
mov eax, DWORD PTR [rdx]
add DWORD PTR [rdi], eax
mov eax, DWORD PTR [rdx]
add DWORD PTR [rsi], eax
ret
```

Хорошо

```
foo:
mov eax, DWORD PTR [rdx]
add DWORD PTR [rdi], eax
add DWORD PTR [rsi], eax
ret
```

- Aliasing - разные указатели указывают на непересекающиеся участки памяти (грубо говоря)
- Strict Aliasing - “Существенно” разные указатели указывают на различные участки памяти
  - Из-за “хорошего” кода часто включают -fno-strict-aliasing
- Нет restrict (в отличие от C)
  - И не видать, что его примут в ближайшем будущем
- Компилятор вынужден догадываться, какие указатели не пересекаются
  - Компилятору сложно догадаться
  - MSVC и не пытается догадаться :)

Исключения

```
void f() {
    try {
        throw 42;
    } catch(...) {
    printf("42");
    }
}
```

clang 6.0.0 -std=c++17 -O3 (same for gcc)

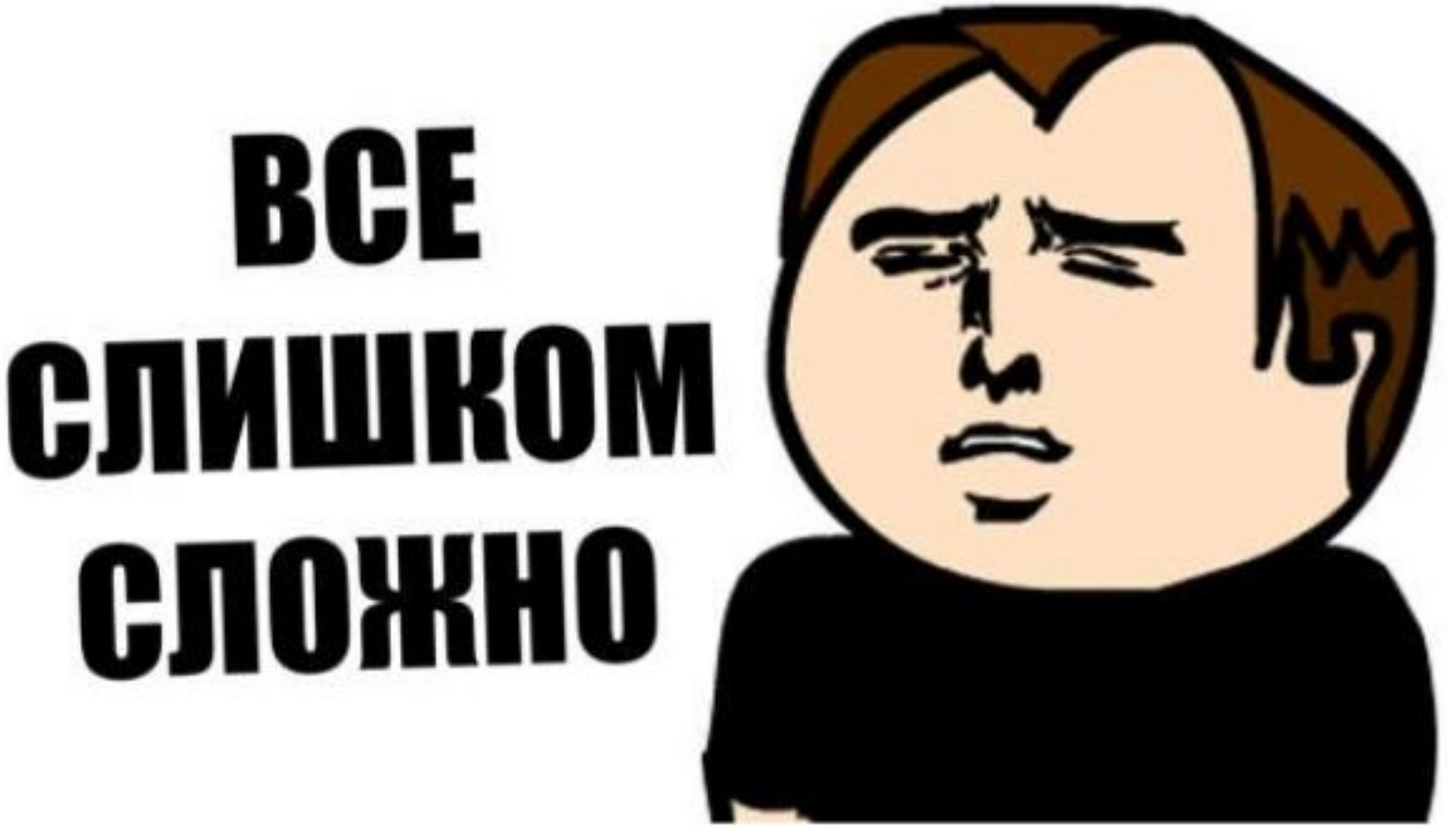
```
f(): # @f()
push rax
mov edi, 4
call __cxa_allocate_exception
mov dword ptr [rax], 42
mov esi, offset typeinfo for int
xor edx, edx
mov rdi, rax
call __cxa_throw
mov rdi, rax
call __cxa_begin_catch
mov edi, offset .L.str
xor eax, eax
call printf
pop rax
jmp __cxa_end_catch # TAILCALL
.L.str:
.asciz "42"
```

- Оптимизировать инструкции по обработке исключений там, где их быть не может
- Оптимизация исключений, которые ни к чему не приводят

Но ведь это всё синтетические примеры! Жизнь - другая!

solarwinds msp

- У всех жизнь разная :)
- “Копейка рубль бережёт”
- Влияние каждой оценивается только тестами
  - Тестируйте, тестируйте и ещё раз тестируйте



Ссылки

Багтрекеры

- https://bugs.llvm.org/
- https://gcc.gnu.org/bugzilla/
- Future Directions for Compiler Optimizations - https://arxiv.org/pdf/1809.02161.pdf
  - В этой работе рекомендую ходить по ссылкам на другие работы
- https://blog.regehr.org/

А что делать сейчас?

solarwinds msp

- “Знай то, что ты используешь” - все компиляторы разные и умеют оптимизировать по-разному
- “На компилятор надейся, а сам не плошай” - мы всё ещё умнее компиляторов во многих аспектах (во многих != всегда)
- Помогать в развитии (отправка багов, участие в тестировании/разработке)
- Ну а для простых смертных...

Вопросы вам (не мне :)!

solarwinds msp

Какие оптимизации на ваш взгляд компилятор **НЕ** должен делать? Когда лучше остановиться и довериться программисту?

Спасибо за внимание!

solarwinds msp

А теперь вопросы мне :)

Александр Зайцев

zamazan4ik@tut.by (основной)
zamazan4ik@gmail.com
Telegram: @zamazan4ik
Twitter: @zamazan4ik

Любые упоминания в Интернете zamazan4ik или ZaMaZaN4iK - скорее всего я :)

Векторизация кода

solarwinds msp

- Они не настолько плохие, как вы можете подумать!
- Зачастую компилятор довольно неплохо векторизует код
  - Особенно если мы говорим про ICC/GCC :)
- Открытые компиляторы отстают от процессоров
- Векторизация довольно сложная штука в плане реализации оптимизации
  - Как пример векторизация в целом независимых кусков кода
- Если ваш компилятор не смог и для вас это важно, то сначала попробуйте ему помочь!
  - И только после этого векторизуйте сами

Что ещё может огорчить?

solarwinds msp

- Неоптимальные/разные реализации вещей в STL (мало имеет отношения к компиляторам)
- Пропущенные оптимизации из-за: багов, особенностей реализации
- Неочевидные оптимизации: UB, memset
  - Подробности читайте в статьях PVS-Studio
- “Неправильные” оптимизации, ломающие программу - баги есть везде :)
- Скорость внедрения новых оптимизаций