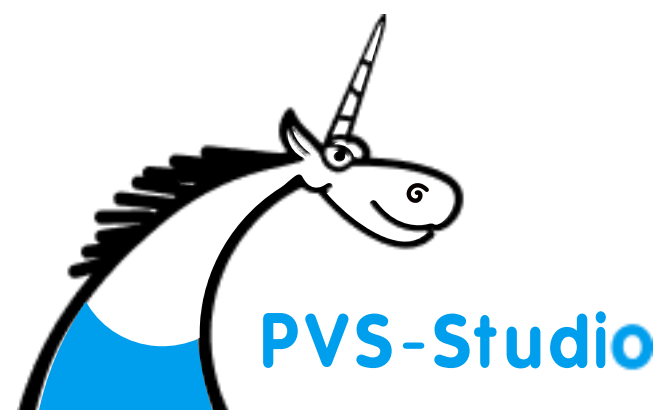


Не всё золото, что блестит

на примере эффективности сгенерированного
C++ кода



Андрей Карпов
CBDO



Андрей Карпов

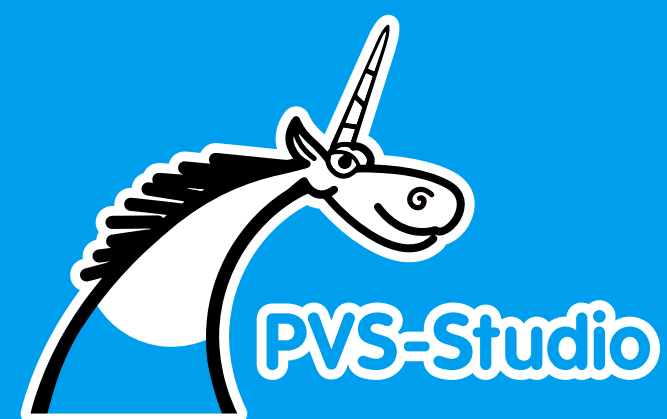
Директор по развитию бизнеса ООО «ПВС»

- Пишу про качество C++ кода
- Один из основателей PVS-Studio
- Интересуюсь разработкой безопасного программного обеспечения (РБПО)
- Скептически смотрю на хайп вокруг ИИ
- pvs-studio.ru | t.me/programming_tales



PVS-Studio

Пока не выглядит так, что
генеративный ИИ (GenAI)
уменьшает количество работы



Ты должен был бороться с *****, а не примкнуть к нему



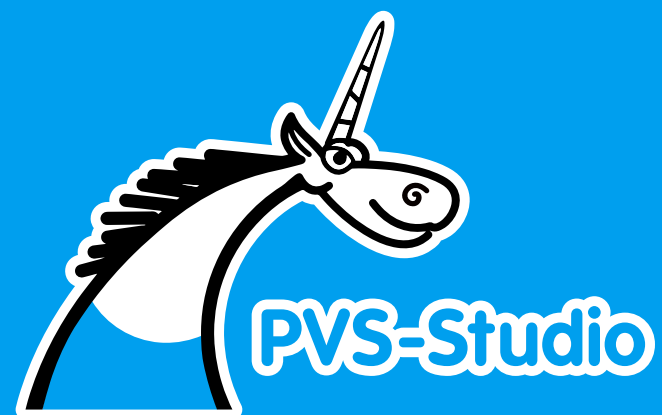
- Найм персонала
- Обучение в целом
- Рост квалификации вайб-кодеров
- Новые классы уязвимостей
- Нейрослоп на Хабре и вообще везде
- Больше работы тем, кто верифицирует публикации, материалы
- И т. д.

Например, новый взгляд на bus-фактор

6

- Проблема не только в том, что исчезнет человек, который многое настраивал
- Даже если есть все артефакты (ТЗ, промты, код)
- Bus-фактору подвержена сама ИИ!
 - Будет запрещена
 - Создатели ИИ будут поглощены кем-то или обанкротятся
 - Станет слишком дорогой
 - Сильно изменится
- Не получится просто взять и «перегенерировать» или поправить то, что уже есть и работает

Сегодня рассмотрим проблему излишнего доверия к созданному коду



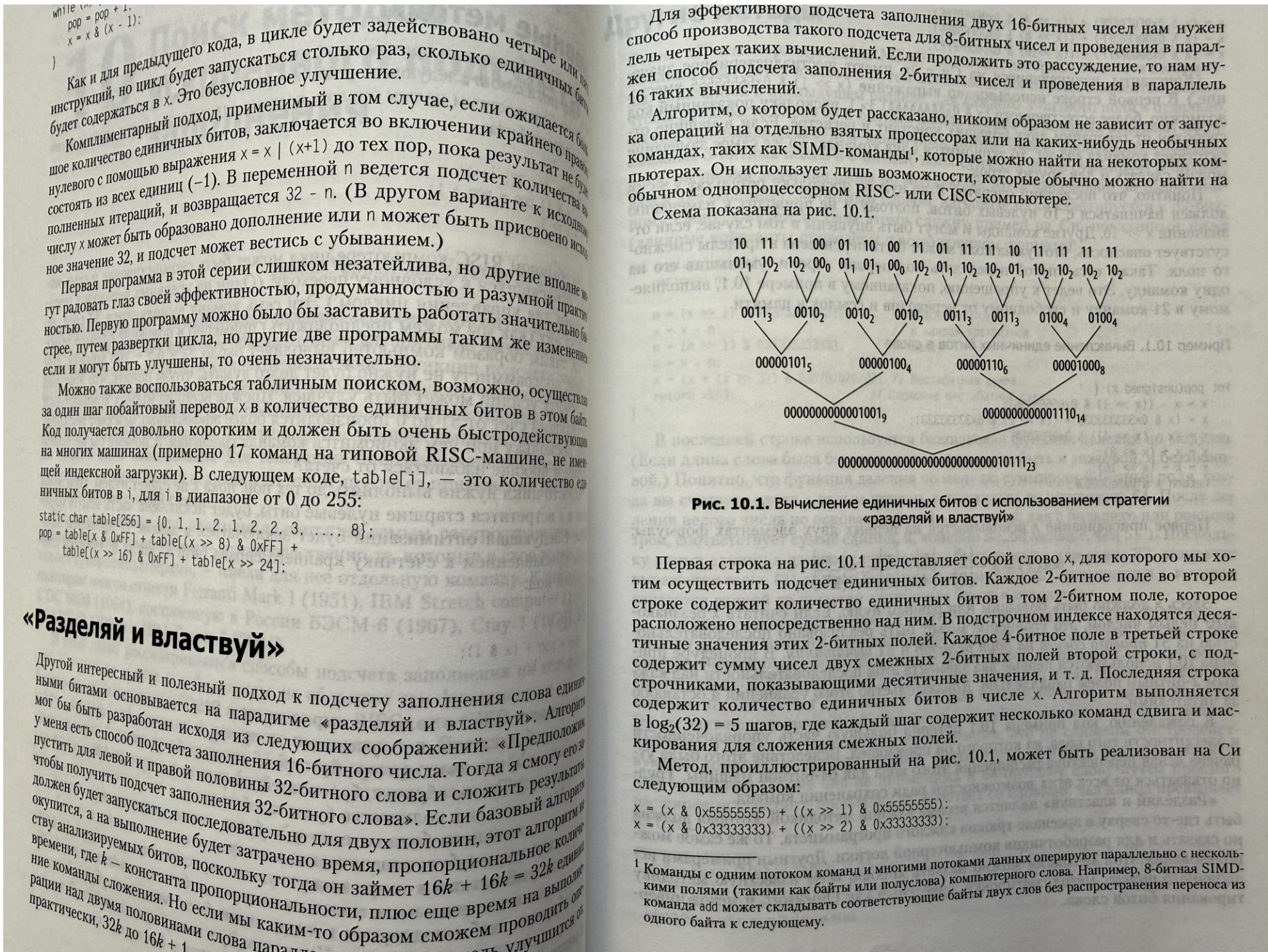
GenAI далёк от стадии плато в цикле зрелости технологий

8

- Придётся терпеть и объяснять вроде бы очевидные вещи



- Раньше – читаешь книги
- Узнаёшь, что есть разные алгоритмы подсчёта единичных бит
- Если нужен быстрый алгоритм в работе, вспоминаешь или идёшь искать готовое решение



- ИИ выдаёт сразу несколько вариантов оптимизированного кода
- Причём предлагает даже такие методы, про которые ты не знаешь
- В общем, предлагает готовый код, который не хуже, чем ты сам сделал. Возможно, даже лучше

- GenAI хорошие и эффективные алгоритмы сделает, вот он их сколько знает
- Зачем было только книгу читать...
- Книги теперь вообще не нужны
- Пишет быстрее меня
- Пишет лучше меня
- «Скрипач не нужен» ©

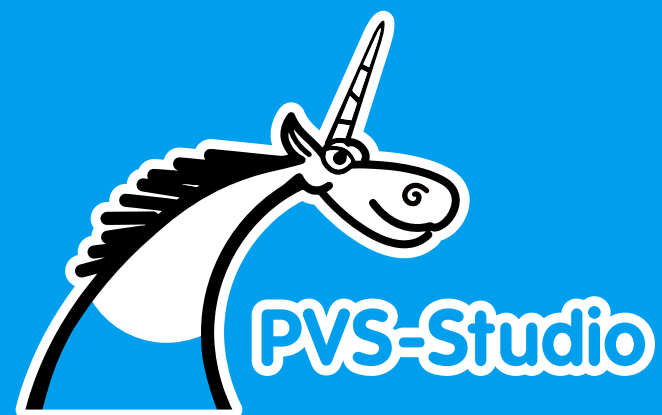


Это и есть очарование и завышенные ожидания

12

- Но это возможно только благодаря тому, что есть кому и с чем сравнить
- Я (человек) оценил эффективность конкретного предложенного решения
- Но кто сказал, что так будет на больших задачах?
- **Роль обзоров кода и проверки не уменьшается, а только растёт!**

Рассмотрим проблематику на примере оптимизаций C++ кода



Неоптимизированный медленный С и С++ код не нужен

14

- Медленный С и С++ код – противоестественно! 😊
- Если не нужна скорость, есть другие языки
- Много задач, где ограниченные ресурсы: память и/или время
- Там нужен тот самый С и С++
- Они для этого и служат, чтобы создавать эффективные программы

Если считаете, что медленный С и С++ код – это нормально

15

- Прошу покинуть доклад 😊
- У нас тут своя атмосфера



- ИИ пытается генерировать оптимизированный код
- Но не «понимает», что это такое
- На C++ это становится особенно заметно



Выделяется бесцельное копирование данных

17

- Копирование данных – дорогая операция
- Особенно кринж, что копирование лишнее и только раздувает код



- Далее из «[Давайте заглянем в этот самый вайб-код](#)»


```
int bt_set_local_name(const char *name)
{
    uint8_t params[248] = {0};
    int len = 0;

    while (name[len] && len < 247) {
        params[len] = name[len];
        len++;
    }

    return hci_send_cmd(HCI_OP_WRITE_LOCAL_NAME, params, 248);
}
```

Данные копируются в промежуточный буфер

```
static int hci_send_cmd(uint16_t opcode, void *params, uint8_t plen)
{
    uint8_t buf[256];

    buf[0] = HCI_COMMAND_PKT;

    struct hci_command_hdr *hdr = (struct hci_command_hdr *)&buf[1];
    hdr->opcode = opcode;
    hdr->plen = plen;

    if (plen > 0 && params) {
        for (int i = 0; i < plen; i++) {
            buf[4 + i] = ((uint8_t *)params)[i];
        }
    }
    ....
}
```

Только ради того, чтобы скопировать в следующий буфер

Для копирования может создаваться однотипный неэффективный код

20

```
uint8_t *src = buf + offset_in_block;
uint8_t *dst = (uint8_t *)inode;
for (size_t i = 0; i < sizeof(struct ext4_inode); i++) {
    dst[i] = src[i];
}
```

```
uint8_t *src = sb_buf;
uint8_t *dst = (uint8_t *)&fs->sb;
for (size_t i = 0; i < sizeof(struct ext4_superblock); i++) {
    dst[i] = src[i];
}
```

```
uint8_t *src = (uint8_t *)ptr;
uint8_t *dst = (uint8_t *)new_ptr;
for (size_t i = 0; i < old_size; i++) {
    dst[i] = src[i];
}
```


Тяготение к раздуванию кодовой базы

21

```
static void str_copy(char *dst, const char *src, int max) {
    int i = 0;
    while (src[i] && i < max - 1) {
        dst[i] = src[i];
        i++;
    }
    dst[i] = '\0';
}
```

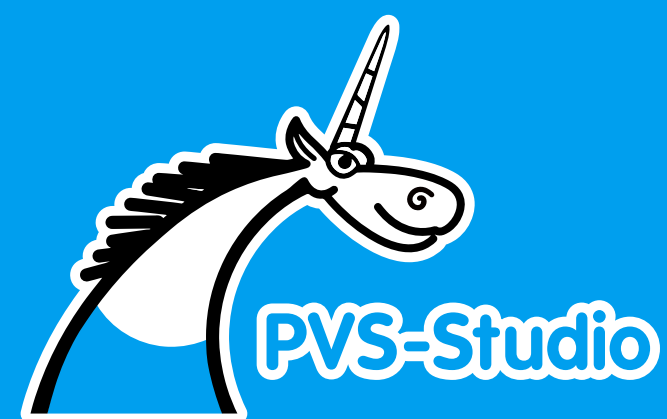
```
static void str_cpy(char *dst, const char *src, int max) {
    int i = 0;
    while (src[i] && i < max - 1) {
        dst[i] = src[i];
        i++;
    }
    dst[i] = '\0';
}
```

```
static void strcpy_safe(char *dst, const char *src, size_t max) {
    size_t i = 0;
    while (src[i] && i < max - 1) {
        dst[i] = src[i];
        i++;
    }
    dst[i] = '\0';
}
```

```
static inline char *strncpy_safe(char *dst, const char *src,
                                  size_t n) {
    size_t i;
    for (i = 0; i < n - 1 && src[i]; i++) {
        dst[i] = src[i];
    }
    dst[i] = '\0';
    return dst;
}
```

Это всё – в одном крошечном проекте

Не очень интересно, давайте
что-то посложнее



Посмотрим код, который действительно вводит в заблуждение

23

- Когда код выглядит красиво, умно и оптимизированно
- Это не значит, что он таковым является
- **Нужны способы/процессы верификации результата**
- [Ревью вайб-кода с гнильцой, который притворяется оптимизированным C++ кодом](#)



Заявка на SIMD-friendly код

24

```
// Check if a span contains only blank characters (space, tab, \r, \n)
// SIMD-friendly: processes 8 bytes at a time
inline bool IsSpanBlank(const char* data, size_t len) {
    size_t i = 0;
    // Process 8 bytes at a time
    for (; i + 8 <= len; i += 8) {
        // For each byte, check if it's NOT a blank character
        // Blank chars: space(0x20), tab(0x09), \n(0x0A), \r(0x0D)
        bool all_blank = true;
        for (size_t j = 0; j < 8; ++j) {
            unsigned char c = static_cast<unsigned char>(data[i + j]);
            bool is_blank = (c == ' ' || c == '\t' || c == '\n' || c == '\r');
            all_blank = all_blank && is_blank;
        }
        if (!all_blank) return false;
    }
    // Handle remaining bytes
    for (; i < len; ++i) {
        unsigned char c = static_cast<unsigned char>(data[i]);
        if (c != ' ' && c != '\t' && c != '\n' && c != '\r') {
            return false;
        }
    }
    return true;
}
```

Не всматривайтесь, сейчас разберём фрагменты

- Комментарий к функции `IsSpanBlank` обещает, что код составлен из расчёта облегчить компилятору задачу его векторизации, например, с помощью SSE-инструкций

Check if a span contains only blank characters (space, tab, \r, \n)
SIMD-friendly: processes 8 bytes at a time

Реализация: вложенный цикл по 8 байт

26

```
// Process 8 bytes at a time
for (; i + 8 <= len; i += 8) {
    // For each byte, check if it's NOT a blank character
    // Blank chars: space(0x20), tab(0x09), \n(0x0A), \r(0x0D)
    bool all_blank = true;
    for (size_t j = 0; j < 8; ++j) {
        unsigned char c = static_cast<unsigned char>(data[i + j]);
        bool is_blank = (c == ' ' || c == '\t' || c == '\n' || c == '\r');
        all_blank = all_blank && is_blank;
    }
    if (!all_blank) return false;
}
```

Код создаёт впечатление, что ИИ знает, что делает

27

- Внутренний цикл всегда обрабатывает по 8 символов, и предполагается, что эта часть может быть оптимизирована с помощью SIMD-инструкций
- Мне код показался подозрительным тем, что на самом деле байты обрабатываются всё так же последовательно
- Вложенный цикл не несёт чего-то полезного
- Такая подсказка компилятору не поможет. Когда могут, компиляторы сами циклы векторизуют

Время экспертного мнения. Если не нужна суперскорость, я бы написал так

28

```
inline bool IsSpanBlank_Simple(const char* data, size_t len) {  
    for (size_t i = 0; i < len; ++i) {  
        unsigned char c = static_cast<unsigned char>(data[i]);  
        if (c != ' ' && c != '\t' && c != '\n' && c != '\r') {  
            return false;  
        }  
    }  
    return true;  
}
```

Нормальный, короткий, достаточный «ручной» код

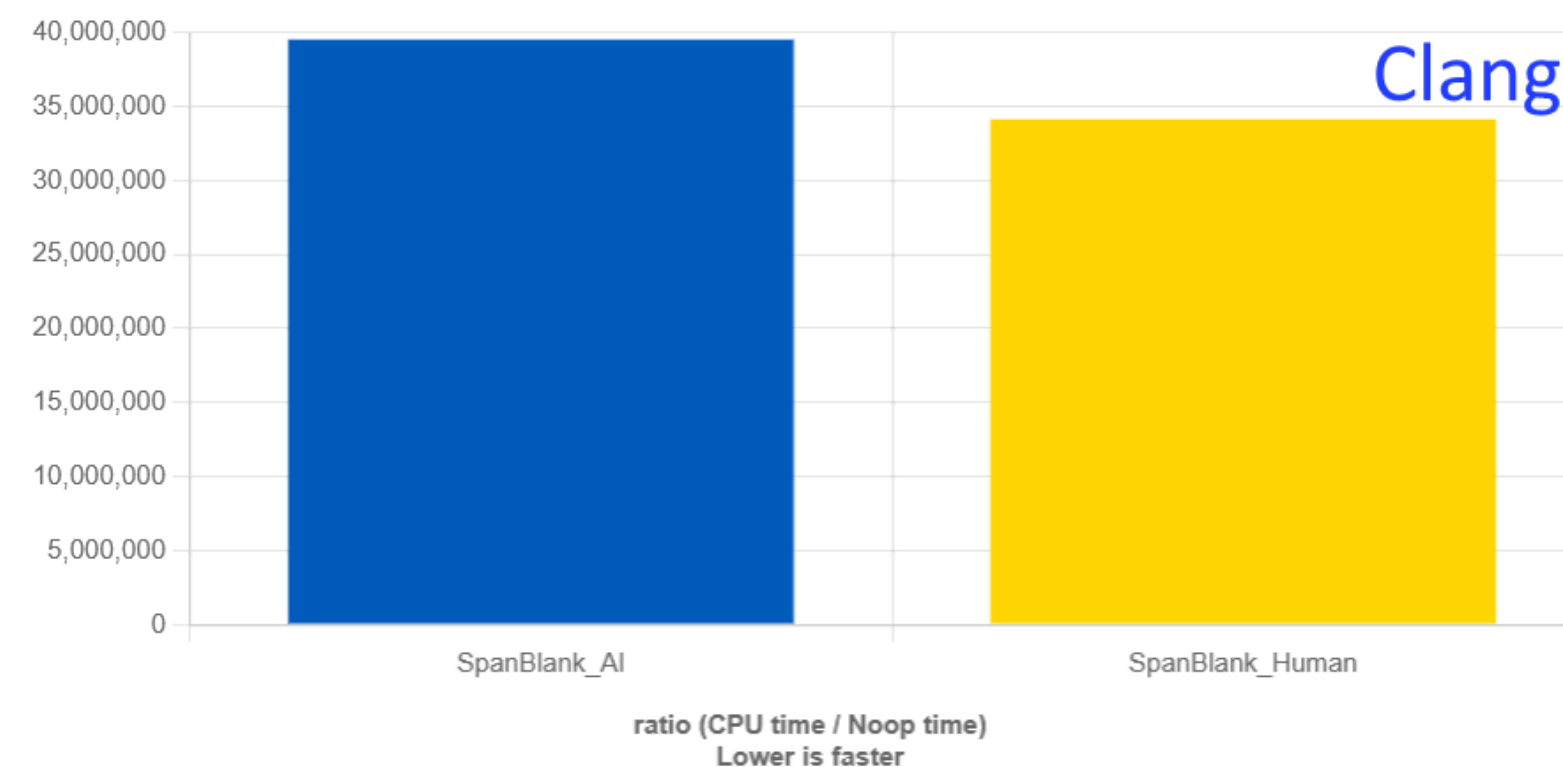
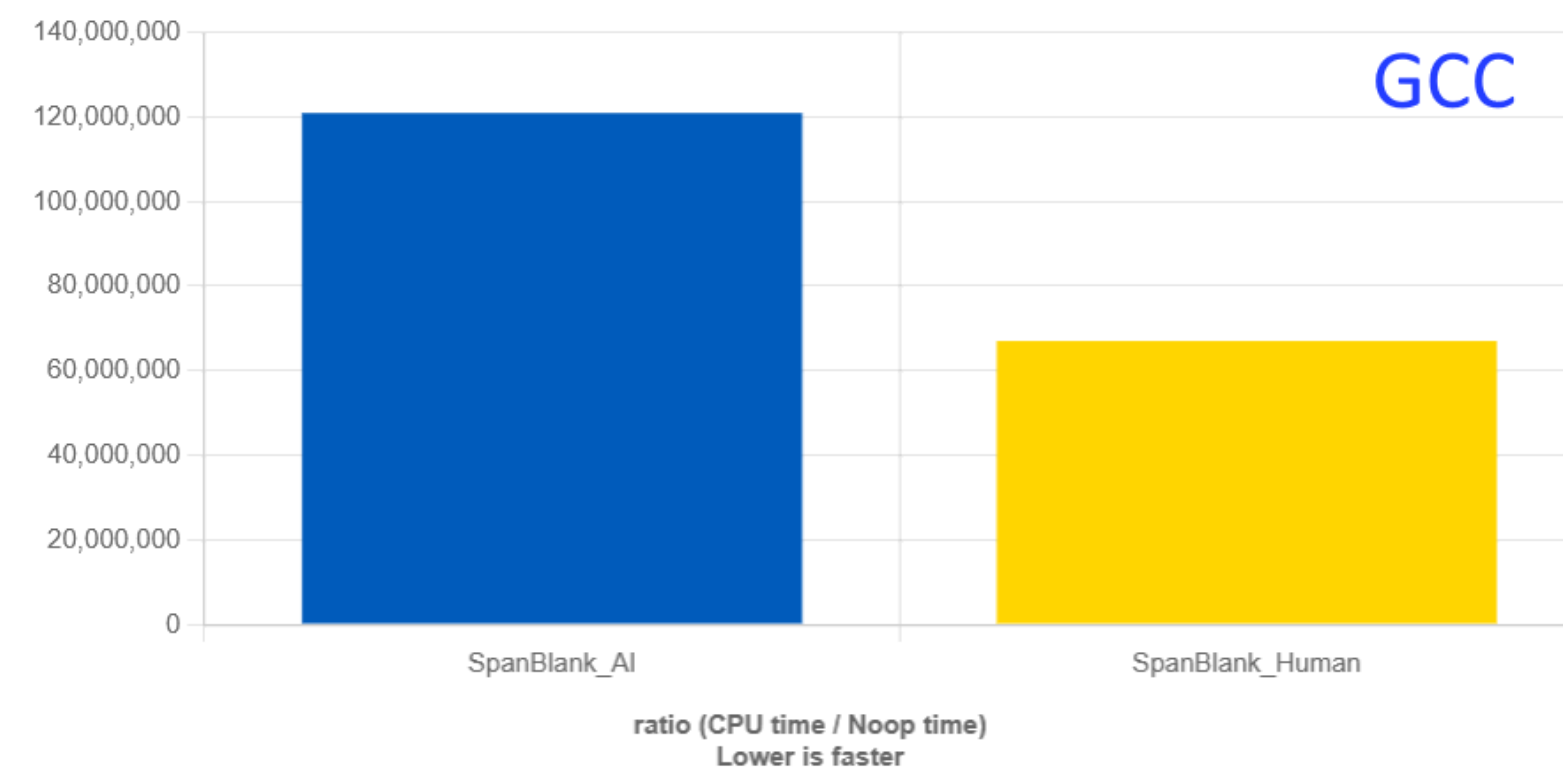
- Я начну искать решения в интернете
- Попрошу ИИ сгенерировать оптимизированные варианты функции
- Попрошу ИИ сгенерировать вариант с явным использованием SIMD
- **Проведу обзор, замеры, выберу наиболее подходящий вариант реализации**

- Ты начнёшь оптимизировать, только когда приспичит
- А здесь ИИ (Claude Opus) сразу оптимизированный код написал!
- А кто сказал, что сгенерирован оптимизированный, а не просто длинный бестолковый код?
- И не забываем классику:
«Преждевременная оптимизация — корень всех зол» ©

Давайте сравним

31

- Подробности см. в приведённой ранее [статье](#)
- Вариант с вложенным циклом хуже, чем простейший



- Short-circuit evaluation в исходном варианте не помогает, а только вредит
- Ситуацию можно улучшить, заменив в изначальном варианте `||` на `|`

```
bool is_blank = (c == ' ' | c == '\t' | c == '\n' | c == '\r');
```



```
inline bool IsSpanBlank_Deepseek(const char* data, size_t len) noexcept {  
    // Создаём битовую маску для допустимых символов  
    // Используем 128-битную маску для ASCII  
    static constexpr uint64_t blank_mask_low = []() {  
        uint64_t mask = 0;  
        mask |= 1ULL << ' '  
        mask |= 1ULL << '\t';  
        mask |= 1ULL << '\n';  
        mask |= 1ULL << '\r';  
        return mask;  
    }();  
  
    for (size_t i = 0; i < len; ++i) {  
        unsigned char uc = static_cast<unsigned char>(data[i]);  
        if (uc >= 64) return false; // Все blank-символы в ASCII < 64  
        if (!((blank_mask_low >> uc) & 1)) return false;  
    }  
    return true;  
}
```

SSE оптимизация от DeepSeek

34

```
inline bool IsSpanBlank_Deepseek_SSE(const char* data, size_t len) noexcept {
    // Создаём векторы с допустимыми символами
    const __m128i space = _mm_set1_epi8(' ');
    const __m128i tab = _mm_set1_epi8('\t');
    const __m128i newline = _mm_set1_epi8('\n');
    const __m128i carriage = _mm_set1_epi8('\r');

    size_t i = 0;

    // Обработываем по 16 байт
    for (; i + 15 < len; i += 16) {
        __m128i chunk = _mm_loadu_si128(reinterpret_cast<const __m128i*>(data + i));

        // Проверяем, что каждый байт является одним из допустимых
        __m128i eq_space = _mm_cmpeq_epi8(chunk, space);
        __m128i eq_tab = _mm_cmpeq_epi8(chunk, tab);
        __m128i eq_nl = _mm_cmpeq_epi8(chunk, newline);
        __m128i eq_cr = _mm_cmpeq_epi8(chunk, carriage);

        __m128i is_blank = _mm_or_si128(_mm_or_si128(eq_space, eq_tab),
                                         _mm_or_si128(eq_nl, eq_cr));

        // Получаем маску и проверяем
        int mask = _mm_movemask_epi8(is_blank);
        if (mask != 0xFFFF) return false;
    }

    // Обработываем остаток
    for (; i < len; ++i) {
        char c = data[i];
        if (c != ' ' && c != '\t' && c != '\n' && c != '\r') return false;
    }

    return true;
}
```

- | | |
|---|------------------|
| 1. Самый простой вариант с одним циклом: | 1.39 сек. |
| 2. Изначальный "оптимизированный код" с вложенным циклом: | 1.86 сек. |
| 3. В исходном варианте операторы <code> </code> заменены на <code> </code> : | 0.351 сек. |
| 4. Вариант от DeepSeek с оптимизацией на битовых масках: | 0.817 сек. |
| 5. Вариант от DeepSeek с оптимизацией на SSE2 интристиках: | 0.0449 сек. |

Изначальный «оптимизированный» код, сгенерированный Claude Opus, худший из всех вариантов

- | | |
|---|--------------------|
| 1. Самый простой вариант с одним циклом: | 0.175 сек. |
| 2. Изначальный "оптимизированный код" с вложенным циклом: | 0.309 сек. |
| 3. В исходном варианте операторы <code> </code> заменены на <code> </code> : | 0.253 сек. |
| 4. Вариант от DeepSeek с оптимизацией на битовых масках: | 0.207 сек. |
| 5. Вариант от DeepSeek с оптимизацией на SSE2 интристиках: | 0.0396 сек. |

Изначальный «оптимизированный» код, сгенерированный Claude Opus, худший из всех вариантов

Самый простой вариант – самый лучший (кроме SSE)!

- Умный код с комментарием, заявляющий, что он оптимизированный, не обязательно таков на самом деле ☺
- **Проверять, проверять, проверять!**
- А если не проверять, то лучше отдавать предпочтение более простым вариантам
- Их хотя бы будет легче поддерживать

- Но ты ведь воспользовался DeepSeek для генерации оптимизированного кода
- Я и не говорю, что не надо пользоваться ИИ
- Я говорю:
 - **Не доверяй по умолчанию!**
 - **Проверяй!**
- Теперь ещё больше ценны эксперты, которые будут валидировать и брать на себя ответственность за результат

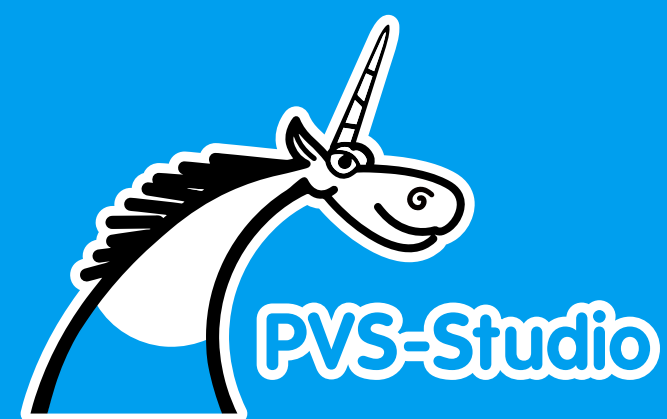
Где брать экспертов?

39

- Понятия не имею. Эта проблематика не только в сфере IT
- Хотя бы не спешите терять старых, безудержно внедряя ИИ
- Кто не пропустит зло, если кругом только энтузиасты вайб-кодинга?



Ещё о тяге к избыточным операциям копирования



Создание строки в верхнем регистре перед сравнением

41

```
std::pmr::string upper;
for (size_t j = 0; j < 9; ++j) {
    upper += static_cast<char>(
        std::toupper(static_cast<unsigned char>(trimmed[j])));
}
if (upper.starts_with("<!DOCTYPE")) {
    goto html_interrupt;
}
```

Пока не критичное место

Хотя уже напрашивается функция сравнения строк без учёта регистра

42

```
bool starts_with_insensitive(std::string_view str,  
                             std::string_view prefix_lower_case)  
{  
    if (str.size() < prefix_lower_case.size()) return false;  
  
    const auto pred = [](unsigned char lhs, unsigned char rhs)  
    {  
        return lhs == std::tolower(rhs);  
    };  
  
    return std::equal(prefix_lower_case.begin(), prefix_lower_case.end(),  
                      str.begin(),  
                      pred);  
}
```

Тогда можно изящно написать:

43

```
if (starts_with_insensitive(trimmed, "<!doctype"))  
{  
    goto html_interrupt;  
}
```

Но всё это ради другого!

```
static const
std::pmr::vector<std::pmr::string> type1_tags = {
    "script", "pre", "style", "textarea"
};
```

```
for (const auto& tag : type1_tags) {
    std::pmr::string open_tag = "<" + tag;
```

Создаются новые строки с добавлением <

А затем ещё строки, чтобы сравнивать без учёта регистра

45

```
std::pmr::string lower;
for (size_t j = 0; j < open_tag.size(); ++j) {
    lower += static_cast<char>(
        std::tolower(
            static_cast<unsigned char>(trimmed[j])));
}
```

```
static const std::pmr::vector<std::pmr::string> type1_tags = {
    "script", "pre", "style", "textarea"
};

for (const auto& tag : type1_tags) {
    std::pmr::string open_tag = "<" + tag;
    if (trimmed.size() >= open_tag.size()) {
        std::pmr::string lower;
        for (size_t j = 0; j < open_tag.size(); ++j) {
            lower += static_cast<char>(
                std::tolower(static_cast<unsigned char>(trimmed[j])));
        }
        if (lower == open_tag) {
            // ....
        }
    }
}
```



Всю эту ***** можно заменить на

47

```
static constexpr std::array type1_tags = {  
    "<script"sv, "<pre"sv, "<style"sv, "<textarea"sv  
};
```

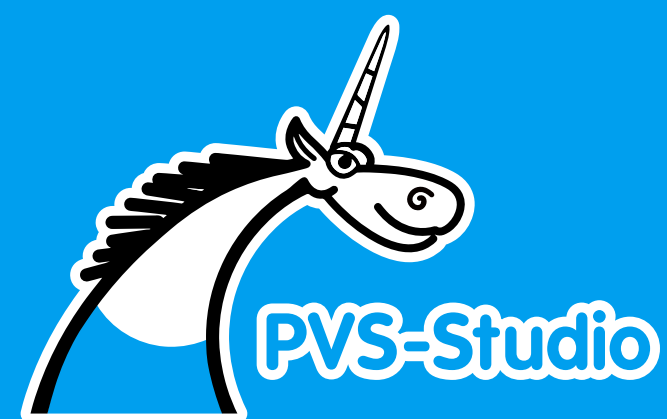
```
for (auto tag : type1_tags) {  
    if (starts_with_insensitive(  
        trimmed, tag.c_str(), tag.size())) {
```

Короче! Быстрее работает!

- А ведь этот говнокод со временем станет базой для обучения следующих версий моделей
- И начнётся новый виток
- Нас могут ждать интересные времена сингулярности вайб-говнокодинга 😊



Ну зато простых ошибок нет



- «У него были хорошие учителя» 😊
- Некоторые разновидности багов реинкарнируются в сгенерированном коде
- Какую-то статистику пока привести не могу, только отдельные наблюдения



Разыменование нулевого указателя

51

```
QGVPage* QGIView::getQGVPage(TechDraw::DrawView* dView)
{
    ViewProviderPage* vpp = getViewProviderPage(dView);
    if (!vpp) {
        return vpp->getQGVPage();
    }
    return nullptr;
}
```

Старое: баг найден PVS-Studio в FreeCAD

```
void enhance::modules::autototem::run()
{
    ...
    auto env = enhance::instance->get_env();
    if (!env)
    {
        env->DeleteLocalRef(player);
        return;
    }
}
```

Сгенерированный код. Как говорится, найди отличия 😊

```
class StaffLayout
{
protected:
    int m_connectingLineLength;
    ....
};

void StaffLayout::resizeStaffLineRow(int row, double x, double length)
{
    ....
    if (m_pageMode != LinearMode && m_connectingLineLength > 0.1) {
```

Старое: баг найден PVS-Studio в Rosegarden

```
int sdk::minecraft_client::get_attack_cooldown() { ... }
```

```
void enhance::modules::shield_breaker::run()  
{  
    ...  
    if (sdk::instance->get_attack_cooldown() > 0.1f)  
        ...  
}
```

Сгенерированный код ([enhance-client](#))

- Нет. Что на входе, то и на выходе
- Пожалуй, код от GenAI содержит меньше опечаток и ляпов
- Но это ничего не меняет
- Всё так же нужен статический анализ кода, обзоры кода, юнит-тесты, регрессионные тесты и так далее
- Иначе в самый неподходящий момент «разработчик» останется с багом, которые непонятно как искать и править

- Плохой способ для большого ответственного проекта
- Означает перепроверку больших частей, а не только изменённого места
- Генерация кода – это дешево, но не в больших проектах
- В итоге это будет дороже, чем просто взять и поправить/доработать нужные фрагменты

- Хорошая цитата:

И вот это, на мой взгляд, сейчас одна из главных проблем. Люди, которые не понимают, как устроено программирование во всей его полноте, сегодня с очень большой уверенностью рассказывают всем остальным, как нужно делать программирование.

Миша Ларченко. [Хороший код больше не важен? Почему разработка катится не туда — мнение техлида](#)

Выводы



- Но это – не чудо, а инструмент
- Что-то проще/быстрее, а что-то сложнее/проблемно
- Возрастает потребность в контроле качества, надёжности
- Вопрос ответственности
- Актуальны старые процессы РБПО:
 - Статический анализ кода
 - Динамический анализ
 - SCA
 - Обзоры кода
 - И т.д.

Чек-лист как делать хорошо

60

- Разработка безопасного программного обеспечения (РБПО)
по ГОСТ Р 56939–2024



Попробовать PVS-Studio

61

- Промокод на месяц: [quality_genai](#)
- Поддерживает: C, C++, C#, Java
- Скоро: Go, JavaScript, TypeScript
- Совместим с ГОСТ Р 71207—2024 (Статический анализ кода)
- Может применяться для РБПО согласно ГОСТ Р 56939—2024
- Включён в Реестр российского ПО: запись № 9837



Остались вопросы

