Анализ автовекторизации в LLVM

# Исполнитель исследования

ФИО: Алибеков Мурад Рамазанович

Номер группы: 381806-1

GitHub: <https://github.com/AlibekovMurad5202>

# Конфигурация

CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz

Instruction set: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, FMA3

# Исследуемый цикл

|  |
| --- |
| void run() { |
| for (int i=0; i < N; i++) { |
| r[i] = a[i] \* b[i]; |
| } |
| } |

# Сравнение LLVM IR[[1]](#footnote-1)

|  |  |
| --- | --- |
| Без векторизации и раскрутки | Без векторизации, с раскруткой |
| for.body:  %3 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %4 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %3, %4  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8 | for.body:  %3 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %4 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %3, %4  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8  **%5 = load float, float\* %add.ptr.i.1, align 4, !tbaa !8**  **%6 = load float, float\* %add.ptr.i14.1, align 4, !tbaa !8**  **%mul.1 = fmul float %5, %6**  **store float %mul.1, float\* %add.ptr.i12.1, align 4, !tbaa !8**  **%7 = load float, float\* %add.ptr.i.2, align 4, !tbaa !8**  **%8 = load float, float\* %add.ptr.i14.2, align 4, !tbaa !8**  **%mul.2 = fmul float %7, %8**  **store float %mul.2, float\* %add.ptr.i12.2, align 4, !tbaa !8**  **%9 = load float, float\* %add.ptr.i.3, align 4, !tbaa !8**  **%10 = load float, float\* %add.ptr.i14.3, align 4, !tbaa !8**  **%mul.3 = fmul float %9, %10**  **store float %mul.3, float\* %add.ptr.i12.3, align 4, !tbaa !8** |

Табл 1.

|  |  |
| --- | --- |
| Без векторизации и раскрутки | С векторизацией, без раскрутки |
| for.body:  %3 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %4 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %3, %4  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8 | vector.body:  %3 = getelementptr inbounds float, float\* %0, i64 %index  %4 = bitcast float\* %3 to <4 x float>\*  **%wide.load = load <4 x float>, <4 x float>\* %4, align 4, !tbaa !8, !alias.scope !10**  %5 = getelementptr inbounds float, float\* %1, i64 %index  %6 = bitcast float\* %5 to <4 x float>\*  **%wide.load24 = load <4 x float>, <4 x float>\* %6, align 4, !tbaa !8, !alias.scope !13**  **%7 = fmul <4 x float> %wide.load, %wide.load24**  %8 = getelementptr inbounds float, float\* %2, i64 %index  %9 = bitcast float\* %8 to <4 x float>\*  **store <4 x float> %7, <4 x float>\* %9, align 4, !tbaa !8, !alias.scope !15, !noalias !17**  for.body:  %11 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %12 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %11, %12  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8 |

Табл 2.

|  |  |
| --- | --- |
| Без векторизации и раскрутки | С векторизацией, c раскруткой |
| for.body:  %3 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %4 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %3, %4  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8 | vector.body:  %3 = getelementptr inbounds float, float\* %0, i64 %index  %4 = bitcast float\* %3 to <4 x float>\*  **%wide.load = load <4 x float>, <4 x float>\* %4, align 4, !tbaa !8, !alias.scope !10**  %5 = getelementptr inbounds float, float\* %3, i64 4  %6 = bitcast float\* %5 to <4 x float>\*  **%wide.load24 = load <4 x float>, <4 x float>\* %6, align 4, !tbaa !8, !alias.scope !10**  %7 = getelementptr inbounds float, float\* %1, i64 %index  %8 = bitcast float\* %7 to <4 x float>\*  **%wide.load25 = load <4 x float>, <4 x float>\* %8, align 4, !tbaa !8, !alias.scope !13**  %9 = getelementptr inbounds float, float\* %7, i64 4  %10 = bitcast float\* %9 to <4 x float>\*  **%wide.load26 = load <4 x float>, <4 x float>\* %10, align 4, !tbaa !8, !alias.scope !13**  **%11 = fmul <4 x float> %wide.load, %wide.load25**  **%12 = fmul <4 x float> %wide.load24, %wide.load26**  %13 = getelementptr inbounds float, float\* %2, i64 %index  %14 = bitcast float\* %13 to <4 x float>\*  **store <4 x float> %11, <4 x float>\* %14, align 4, !tbaa !8, !alias.scope !15, !noalias !17**  %15 = getelementptr inbounds float, float\* %13, i64 4  %16 = bitcast float\* %15 to <4 x float>\*  **store <4 x float> %12, <4 x float>\* %16, align 4, !tbaa !8, !alias.scope !15, !noalias !17**  %17 = getelementptr inbounds float, float\* %0, i64 %index.next  %18 = bitcast float\* %17 to <4 x float>\*  **%wide.load.1 = load <4 x float>, <4 x float>\* %18, align 4, !tbaa !8, !alias.scope !10**  %19 = getelementptr inbounds float, float\* %17, i64 4  %20 = bitcast float\* %19 to <4 x float>\*  **%wide.load24.1 = load <4 x float>, <4 x float>\* %20, align 4, !tbaa !8, !alias.scope !10**  %21 = getelementptr inbounds float, float\* %1, i64 %index.next  %22 = bitcast float\* %21 to <4 x float>\*  **%wide.load25.1 = load <4 x float>, <4 x float>\* %22, align 4, !tbaa !8, !alias.scope !13**  %23 = getelementptr inbounds float, float\* %21, i64 4  %24 = bitcast float\* %23 to <4 x float>\*  **%wide.load26.1 = load <4 x float>, <4 x float>\* %24, align 4, !tbaa !8, !alias.scope !13**  **%25 = fmul <4 x float> %wide.load.1, %wide.load25.1**  **%26 = fmul <4 x float> %wide.load24.1, %wide.load26.1**  %27 = getelementptr inbounds float, float\* %2, i64 %index.next  %28 = bitcast float\* %27 to <4 x float>\*  **store <4 x float> %25, <4 x float>\* %28, align 4, !tbaa !8, !alias.scope !15, !noalias !17**  %29 = getelementptr inbounds float, float\* %27, i64 4  %30 = bitcast float\* %29 to <4 x float>\*  **store <4 x float> %26, <4 x float>\* %30, align 4, !tbaa !8, !alias.scope !15, !noalias !17**  for.body:  %32 = load float, float\* %add.ptr.i, align 4, !tbaa !8  %33 = load float, float\* %add.ptr.i14, align 4, !tbaa !8  %mul = fmul float %32, %33  store float %mul, float\* %add.ptr.i12, align 4, !tbaa !8  **%34 = load float, float\* %add.ptr.i.1, align 4, !tbaa !8**  **%35 = load float, float\* %add.ptr.i14.1, align 4, !tbaa !8**  **%mul.1 = fmul float %34, %35**  **store float %mul.1, float\* %add.ptr.i12.1, align 4, !tbaa !8**  **%36 = load float, float\* %add.ptr.i.2, align 4, !tbaa !8**  **%37 = load float, float\* %add.ptr.i14.2, align 4, !tbaa !8**  **%mul.2 = fmul float %36, %37**  **store float %mul.2, float\* %add.ptr.i12.2, align 4, !tbaa !8**  **%38 = load float, float\* %add.ptr.i.3, align 4, !tbaa !8**  **%39 = load float, float\* %add.ptr.i14.3, align 4, !tbaa !8**  **%mul.3 = fmul float %38, %39**  **store float %mul.3, float\* %add.ptr.i12.3, align 4, !tbaa !8** |

Табл 3.

# Анализ LLVM IR

Из листинга видно, что в версии без автовекторизации и раскрутки цикла в каждой итерации выполняются 4 основные инструкции (не считая вспомогательных, не представленных в данном листинге), 3 из которых – работа с памятью (что повлияет на время исполнения):

* загрузка из памяти первого числа
* загрузка из памяти второго числа
* их умножение
* запись в память результата умножения

В “раскрученном” коде (коде с раскруткой циклов (unroll\_loops)) за одну итерацию выполняется в 4 раза больше инструкций. Это представлено в Табл 1 увеличенным объемом кода внутри for.body

В векторизованном коде за одну итерацию выполняется такое же число инструкций. Однако обрабатывается уже в 4 раза больше данных. Это представлено в Табл 2 участком vector.body

В последней (Табл 3) версии (с автовекторизацией и раскруткой одновременно) объединяются подходы, примененные в двух предыдущих случаях, а именно – за одну итерацию обрабатывается в 4 раза больше инструкций и в 4 раза больше данных.

# Измерение времени исполнения[[2]](#footnote-2)

|  |  |  |
| --- | --- | --- |
| **Конфигурация** | **Время, мс** | **Ускорение, раз** |
| Без автовекторизации и раскрутки | 759.532 | - |
| Без автовекторизации, с раскруткой | 751.070 | 1.011 |
| С автовекторизацией, без раскрутки | 486.019 | 1.563 |
| С автовекторизацией, с раскруткой | 482.611 | 1.574 |

# Анализ времени исполнения

Наиболее производительной, как ожидалось, оказалась конфигурация с автовекторизацией и раскруткой. Ускорение по сравнению с версией без этих двух оптимизаций составило больше, чем в полтора раза!

Однако можно заметить интересный эффект: версии с раскруткой и без неё практически идентичны во времени (ускорение меньше 2%). Причиной является следующее: из анализа IR представления можно было заметить, что в “оригинальной” версии 3 из 4 основных инструкций – работа с памятью (загрузка из памяти и запись в память), которые являются дорогостоящими, с точки зрения времени исполнения. Сама операция умножения является относительно быстрой. При использовании раскрутки увеличивается число инструкций, а, следовательно, увеличивается и число обращений к памяти за одну итерацию. Как следствие этого, ускорение, приобретенное за счет раскрутки циклов практически нивелируется накладными расходами на обращение к памяти.

В отличии от раскрутки, векторизация не увеличивает число инструкций (в том числе и обращений к памяти), а увеличивает размер данных, обрабатываемых за одну итерацию. Поэтому основное ускорение приобретается в данной программе благодаря использованию автовекторизации.

1. В целях улучшения читаемости вспомогательные инструкции были удалены из листингов [↑](#footnote-ref-1)
2. Измерение времени исполнения и ускорение высчитывались на основе среднего времени исполнения по результатам 11 тестовых запусков [↑](#footnote-ref-2)