Анализ автовекторизации в LLVM

Конфигурация

CPU: AMD(R) Athlon™x4 840 quad core processor x 4

Instruction set: MMX,MMX+, SSE, SSE2 , SSE3, SSSE3,SSE4 / SSE4.1 + SSE4.2 , SSE4a, AES , AVX , BMI1 , F16C , FMA3, FMA4 , TBM, XOP, AMD64 , AMD-V , Turbo Core 3.0 technology

Исследуемый цикл

for (int i=0; i < N; i++) { `r[i] = a[i] * b[i];`}}

Сравнение LLVM IR

Без векторизации и раскрутки	Без векторизации, с раскруткой
9: ; preds = %9, %1 %10 = phi i64 [0, %1], [%17, %9] %11 = getelementptr inbounds i32, i32* %3, i64 %10 **%12 = load i32, i32* %11, align 4, !tbaa !7** %13 = getelementptr inbounds i32, i32* %5, i64 %10 **%14 = load i32, i32* %13, align 4, !tbaa !7** **%15 = mul nsw i32 %14, %12** %16 = getelementptr inbounds i32, i32* %7, i64 %10 **store i32 %15, i32* %16, align 4, !tbaa !7** %17 = add nuw nsw i64 %10, 1 %18 = icmp eq i64 %17, 1048576 br i1 %18, label %8, label %9, !llvm.loop !9 }	9: ; preds = %9, %1 %10 = phi i64 [0, %1], [%38, %9] %11 = getelementptr inbounds i32, i32* %3, i64 %10 **%12 = load i32, i32* %11, align 4, !tbaa !7** %13 = getelementptr inbounds i32, i32* %5, i64 %10 **%14 = load i32, i32* %13, align 4, !tbaa !7** **%15 = mul nsw i32 %14, %12** %16 = getelementptr inbounds i32, i32* %7, i64 %10 **store i32 %15, i32* %16, align 4, !tbaa !7** %17 = or i64 %10, 1 %18 = getelementptr inbounds i32, i32* %3, i64 %17 **%19 = load i32, i32* %18, align 4, !tbaa !7** %20 = getelementptr inbounds i32, i32* %5, i64 %17 **%21 = load i32, i32* %20, align 4, !tbaa !7** **%22 = mul nsw i32 %21, %19** %23 = getelementptr inbounds i32, i32* %7, i64 %17 **store i32 %22, i32* %23, align 4, !tbaa !7** %24 = or i64 %10, 2 %25 = getelementptr inbounds i32, i32* %3, i64 %24 **%26 = load i32, i32* %25, align 4, !tbaa !7** %27 = getelementptr inbounds i32, i32* %5, i64 %24 **%28 = load i32, i32* %27, align 4, !tbaa !7** **%29 = mul nsw i32 %28, %26** %30 = getelementptr inbounds i32, i32* %7, i64 %24 **store i32 %29, i32* %30, align 4, !tbaa !7** %31 = or i64 %10, 3 %32 = getelementptr inbounds i32, i32* %3, i64 %31 **%33 = load i32, i32* %32, align 4, !tbaa !7** %34 = getelementptr inbounds i32, i32* %5, i64 %31 **%35 = load i32, i32* %34, align 4, !tbaa !7** **%36 = mul nsw i32 %35, %33** %37 = getelementptr inbounds i32, i32* %7, i64 %31 **store i32 %36, i32* %37, align 4, !tbaa !7** %38 = add nuw nsw i64 %10, 4 %39 = icmp eq i64 %38, 1048576 br i1 %39, label %8, label %9 }

Табл 1.

Без векторизации и раскрутки	С векторизации, без раскрутки
9: ; preds = %9, %1 %10 = phi i64 [0, %1], [%17, %9] %11 = getelementptr inbounds i32, i32* %3, i64 %10 **%12 = load i32, i32* %11, align 4, !tbaa !7**	18: ; preds = %1, %18 %19 = phi i64 [%29, %18], [0, %1] %20 = getelementptr inbounds i32, i32* %3, i64 %19 %21 = bitcast i32* %20 to <4 x i32>*

%13 = getelementptr inbounds i32, i32* %5, i64 %10 **%14 = load i32, i32* %13, align 4, !tbaa !7** **%15 = mul nsw i32 %14, %12** %16 = getelementptr inbounds i32, i32* %7, i64 %10 **store i32 %15, i32* %16, align 4, !tbaa !7** %17 = add nuw nsw i64 %10, 1 %18 = icmp eq i64 %17, 1048576 br i1 %18, label %8, label %9, !llvm.loop !9 }	**%22 = load <4 x i32>, <4 x i32>* %21, align 4, !tbaa !7, !alias.scope !9** %23 = getelementptr inbounds i32, i32* %5, i64 %19 %24 = bitcast i32* %23 to <4 x i32>* **%25 = load <4 x i32>, <4 x i32>* %24, align 4, !tbaa !7, !alias.scope !12** **%26 = mul nsw <4 x i32> %25, %22** %27 = getelementptr inbounds i32, i32* %7, i64 %19 %28 = bitcast i32* %27 to <4 x i32>* **store <4 x i32> %26, <4 x i32>* %28, align 4, !tbaa !7, !alias.scope !14, !noalias !16** %29 = add i64 %19, 4 %30 = icmp eq i64 %29, 1048576 br i1 %30, label %31, label %18, !llvm.loop !17 32: ; preds = %1, %32 %33 = phi i64 [%40, %32], [0, %1] %34 = getelementptr inbounds i32, i32* %3, i64 %33 %35 = load i32, i32* %34, align 4, !tbaa !7 %36 = getelementptr inbounds i32, i32* %5, i64 %33 %37 = load i32, i32* %36, align 4, !tbaa !7 %38 = mul nsw i32 %37, %35 %39 = getelementptr inbounds i32, i32* %7, i64 %33 store i32 %38, i32* %39, align 4, !tbaa !7 %40 = add nuw nsw i64 %33, 1 %41 = icmp eq i64 %40, 1048576 br i1 %41, label %31, label %32, !llvm.loop !20 }

Табл 2.

Без векторизации и раскрутки	С векторизации с раскруткой
9: ; preds = %9, %1 %10 = phi i64 [0, %1], [%17, %9] %11 = getelementptr inbounds i32, i32* %3, i64 %10 **%12 = load i32, i32* %11, align 4, !tbaa !7** %13 = getelementptr inbounds i32, i32* %5, i64 %10 **%14 = load i32, i32* %13, align 4, !tbaa !7** **%15 = mul nsw i32 %14, %12** %16 = getelementptr inbounds i32, i32* %7, i64 %10 **store i32 %15, i32* %16, align 4, !tbaa !7** %17 = add nuw nsw i64 %10, 1 %18 = icmp eq i64 %17, 1048576 br i1 %18, label %8, label %9, !llvm.loop !9 }	18: ; preds = %1, %18 %19 = phi i64 [%57, %18], [0, %1] %20 = getelementptr inbounds i32, i32* %3, i64 %19 %21 = bitcast i32* %20 to <4 x i32>* **%22 = load <4 x i32>, <4 x i32>* %21, align 4, !tbaa !7, !alias.scope !9** %23 = getelementptr inbounds i32, i32* %20, i64 4 %24 = bitcast i32* %23 to <4 x i32>* **%25 = load <4 x i32>, <4 x i32>* %24, align 4, !tbaa !7, !alias.scope !9** %26 = getelementptr inbounds i32, i32* %5, i64 %19 %27 = bitcast i32* %26 to <4 x i32>* **%28 = load <4 x i32>, <4 x i32>* %27, align 4, !tbaa !7, !alias.scope !12** %29 = getelementptr inbounds i32, i32* %26, i64 4 %30 = bitcast i32* %29 to <4 x i32>* **%31 = load <4 x i32>, <4 x i32>* %30, align 4, !tbaa !7, !alias.scope !12** **%32 = mul nsw <4 x i32> %28, %22** **%33 = mul nsw <4 x i32> %31, %25** %34 = getelementptr inbounds i32, i32* %7, i64 %19 %35 = bitcast i32* %34 to <4 x i32>* **store <4 x i32> %32, <4 x i32>* %35, align 4, !tbaa !7, !alias.scope !14, !noalias !16** %36 = getelementptr inbounds i32, i32* %34, i64 4 %37 = bitcast i32* %36 to <4 x i32>* **store <4 x i32> %33, <4 x i32>* %37, align 4, !tbaa !7, !alias.scope !14, !noalias !16** %38 = or i64 %19, 8 %39 = getelementptr inbounds i32, i32* %3, i64 %38 %40 = bitcast i32* %39 to <4 x i32>* **%41 = load <4 x i32>, <4 x i32>* %40, align 4, !tbaa !7, !alias.scope !9** %42 = getelementptr inbounds i32, i32* %39, i64 4 %43 = bitcast i32* %42 to <4 x i32>* **%44 = load <4 x i32>, <4 x i32>* %43, align 4, !tbaa !7, !alias.scope !9** %45 = getelementptr inbounds i32, i32* %5, i64 %38 %46 = bitcast i32* %45 to <4 x i32>* **%47 = load <4 x i32>, <4 x i32>* %46, align 4, !tbaa !7, !alias.scope !12** %48 = getelementptr inbounds i32, i32* %45, i64 4 %49 = bitcast i32* %48 to <4 x i32>*

	%50 = load <4 x i32>, <4 x i32>* %49, align 4, !tbaa !7, !alias.scope !12 **%51 = mul nsw <4 x i32> %47, %41** **%52 = mul nsw <4 x i32> %50, %44** %53 = getelementptr inbounds i32, i32* %7, i64 %38 %54 = bitcast i32* %53 to <4 x i32>* **store <4 x i32> %51, <4 x i32>* %54, align 4, !tbaa !7, !alias.scope !14, !noalias !16** %55 = getelementptr inbounds i32, i32* %53, i64 4 %56 = bitcast i32* %55 to <4 x i32>* **store <4 x i32> %52, <4 x i32>* %56, align 4, !tbaa !7, !alias.scope !14, !noalias !16** %57 = add nuw nsw i64 %19, 16 %58 = icmp eq i64 %57, 1048576 br i1 %58, label %59, label %18, !llvm.loop !17 60: ; preds = %1, %60 %61 = phi i64 [%89, %60], [0, %1] %62 = getelementptr inbounds i32, i32* %3, i64 %61 **%63 = load i32, i32* %62, align 4, !tbaa !7** %64 = getelementptr inbounds i32, i32* %5, i64 %61 **%65 = load i32, i32* %64, align 4, !tbaa !7** **%66 = mul nsw i32 %65, %63** %67 = getelementptr inbounds i32, i32* %7, i64 %61 **store i32 %66, i32* %67, align 4, !tbaa !7** %68 = or i64 %61, 1 %69 = getelementptr inbounds i32, i32* %3, i64 %68 **%70 = load i32, i32* %69, align 4, !tbaa !7** %71 = getelementptr inbounds i32, i32* %5, i64 %68 **%72 = load i32, i32* %71, align 4, !tbaa !7** **%73 = mul nsw i32 %72, %70** %74 = getelementptr inbounds i32, i32* %7, i64 %68 **store i32 %73, i32* %74, align 4, !tbaa !7** %75 = or i64 %61, 2 %76 = getelementptr inbounds i32, i32* %3, i64 %75 **%77 = load i32, i32* %76, align 4, !tbaa !7** %78 = getelementptr inbounds i32, i32* %5, i64 %75 **%79 = load i32, i32* %78, align 4, !tbaa !7** **%80 = mul nsw i32 %79, %77** %81 = getelementptr inbounds i32, i32* %7, i64 %75 **store i32 %80, i32* %81, align 4, !tbaa !7** %82 = or i64 %61, 3 %83 = getelementptr inbounds i32, i32* %3, i64 %82 **%84 = load i32, i32* %83, align 4, !tbaa !7** %85 = getelementptr inbounds i32, i32* %5, i64 %82 **%86 = load i32, i32* %85, align 4, !tbaa !7** **%87 = mul nsw i32 %86, %84** %88 = getelementptr inbounds i32, i32* %7, i64 %82 **store i32 %87, i32* %88, align 4, !tbaa !7** %89 = add nuw nsw i64 %61, 4 %90 = icmp eq i64 %89, 1048576 br i1 %90, label %59, label %60, !llvm.loop !19 }

Табл 3.

Анализ LLVM IR

Как видно из таблицы 1 при раскрутке и без векторизации цикла за одну итерацию выполняется в 4 раза больше инструкций, чем в версии без раскрутки и без векторизации. (выделены в цветом в таблице 1).

В таблице 2 можно увидеть, что в версии с векторизацией и без раскрутки за одну итерацию обрабатывается в 4 раза больше данных (выделены в цветом в таблице 2) ,но при этом число инструкций такое же, как в версии без раскрутки и без векторизации.

В таблице 3 с векторизацией и раскруткой цикла видно, что за одну итерацию обрабатывается в 4 раза больше инструкций и в 4 раза больше данных , а значит могут раскручиваться итерации векторизованного цикла, т.е. происходит уменьшение числа итераций. (выделены в цветом в таблице 3).

Измерение времени исполнения

Конфигурация	**Время, мс**	**Ускорение**
Без автовекторизации и раскрутки	2860.08 ms	–
Без автовекторизации, с раскруткой	2781.44 ms	1,02
С автовекторизации, без раскруткой	1945.05 ms	1,47
С автовекторизации, без раскруткой	1877.02 ms	1,52

Анализ времени исполнения

Наиболее производительной оказалась конфигурация с векторизацией и раскруткой.

Ускорение версии с раскруткой и без неё практически идентичны во времени. Так вышло из-за того, что в версии без раскрутки и без векторизации выполняются 4 основных инструкции:

1) Загрузить из памяти 1-ое число;
2) Загрузить из памяти 2-ое число;
3) Перемножить эти числа;
4) Записать в память результат.

Как можно заметить, 3 из 4 это операции с памятью, а т.к. они являются затратными по времени исполнения, то при использовании раскрутки увеличивается и число инструкций, а тогда получается, что увеличивается и число обращений к памяти на одну итерацию. И как итог, ускорение от раскрутки перекрывают расходы на обращение в память.

Векторизация же увеличивает не число инструкций, а значит и не увеличивается число обращений к памяти, а увеличивает размер данных, обрабатываемых за одну итерацию. Следовательно и значительное ускорение приобретается из-за использования векторизации.