Анализ автовекторизации в LLVM

Конфигурация

CPU: AMD Ryzen 7 2700 Eight-Core Processor 3.20 GHz
Instruction set: x86, x86-64, MMX, MMX+, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A, AVX, AVX2, FMA, AES, SHA

Исследуемый цикл

```
for (int i=0; i < N; i++) r[i] = i;
```

Сравнение LLVM IR

Без векторизации и раскрутки	Без векторизации, с раскруткой

4:
; preds = %5
  ret void
 	
5:
; preds = %5, %1
  %6 = phi i64 [0, %1], [%9, %5]
  %7 = getelementptr inbounds i32, i32* %3, i64 %6
  %8 = trunc i64 %6 to i32
  store i32 %8, i32* %7, align 4, !tbaa !7
  %9 = add nuw nsw i64 %6, 1
  %10 = icmp eq i64 %9, 1048576
  br i1 %10, label %4, label %5
}
	4:
	; preds = %5
	  ret void
	
	5:
	; preds = %5, %1
	  %6 = phi i64 [0, %1], [%30, %5]
	  %7 = getelementptr inbounds i32, i32* %3, i64 %6
	  %8 = trunc i64 %6 to i32
	  store i32 %8, i32* %7, align 4, !tbaa !7
	  %9 = or i64 %6, 1
	  %10 = getelementptr inbounds i32, i32* %3, i64 %9
	  %11 = trunc i64 %9 to i32
	  store i32 %11, i32* %10, align 4, !tbaa !7
	  %12 = or i64 %6, 2
	  %13 = getelementptr inbounds i32, i32* %3, i64 %12
	  %14 = trunc i64 %12 to i32
	  store i32 %14, i32* %13, align 4, !tbaa !7
	  %15 = or i64 %6, 3
	  %16 = getelementptr inbounds i32, i32* %3, i64 %15
	  %17 = trunc i64 %15 to i32
	  store i32 %17, i32* %16, align 4, !tbaa !7
	  %18 = or i64 %6, 4
	  %19 = getelementptr inbounds i32, i32* %3, i64 %18
	  %20 = trunc i64 %18 to i32
	  store i32 %20, i32* %19, align 4, !tbaa !7
	  %21 = or i64 %6, 5
	  %22 = getelementptr inbounds i32, i32* %3, i64 %21
	  %23 = trunc i64 %21 to i32
	  store i32 %23, i32* %22, align 4, !tbaa !7
	  %24 = or i64 %6, 6
	  %25 = getelementptr inbounds i32, i32* %3, i64 %24
	  %26 = trunc i64 %24 to i32
	  store i32 %26, i32* %25, align 4, !tbaa !7
	  %27 = or i64 %6, 7
	  %28 = getelementptr inbounds i32, i32* %3, i64 %27
	  %29 = trunc i64 %27 to i32
	  store i32 %29, i32* %28, align 4, !tbaa !7
	  %30 = add nuw nsw i64 %6, 8
	  %31 = icmp eq i64 %30, 1048576

	br i1 %31, label %4, label %5 }

Табл 1.

Без векторизации и раскрутки	С векторизации, без раскрутки
4: ; preds = %5 ret void 5: ; preds = %5, %1 %6 = phi i64 [0, %1], [%9, %5] %7 = getelementptr inbounds i32, i32* %3, i64 %6 %8 = trunc i64 %6 to i32 store i32 %8, i32* %7, align 4, !tbaa !7 %9 = add nuw nsw i64 %6, 1 %10 = icmp eq i64 %9, 1048576 br i1 %10, label %4, label %5 }	4: ; preds = %4, %1 %5 = phi i64 [0, %1], [%9, %4] %6 = phi <4 x i32> [<i32 0, i32 1, i32 2, i32 3>, %1], [%10, %4] %7 = getelementptr inbounds i32, i32* %3, i64 %5 %8 = bitcast i32* %7 to <4 x i32>* store <4 x i32> %6, <4 x i32>* %8, align 4, !tbaa !7 %9 = add i64 %5, 4 %10 = add <4 x i32> %6, <i32 4, i32 4, i32 4, i32 4> %11 = icmp eq i64 %9, 1048576 br i1 %11, label %12, label %4, !llvm.loop !9 12: ; preds = %4 ret void }

Табл 2.

Без векторизации и раскрутки	С векторизации с раскруткой
4: ; preds = %5 ret void 5: ; preds = %5, %1 %6 = phi i64 [0, %1], [%9, %5] %7 = getelementptr inbounds i32, i32* %3, i64 %6 %8 = trunc i64 %6 to i32 store i32 %8, i32* %7, align 4, !tbaa !7 %9 = add nuw nsw i64 %6, 1 %10 = icmp eq i64 %9, 1048576	4: ; preds = %4, %1 %5 = phi i64 [0, %1], [%33, %4] %6 = phi <4 x i32> [<i32 0, i32 1, i32 2, i32 3>, %1], [%34, %4] %7 = getelementptr inbounds i32, i32* %3, i64 %5 %8 = add <4 x i32> %6, <i32 4, i32 4, i32 4, i32 4> %9 = bitcast i32* %7 to <4 x i32>* store <4 x i32> %6, <4 x i32>* %9, align 4, !tbaa !7 %10 = getelementptr inbounds i32, i32* %7, i64 4 %11 = bitcast i32* %10 to <4 x i32>*

br i1 %10, label %4, label %5 }	store <4 x i32> %8, <4 x i32>* %11, align 4, !tbaa !7 %12 = or i64 %5, 8 %13 = add <4 x i32> %6, <i32 8, i32 8, i32 8, i32 8> %14 = getelementptr inbounds i32, i32* %3, i64 %12 %15 = add <4 x i32> %6, <i32 12, i32 12, i32 12, i32 12> %16 = bitcast i32* %14 to <4 x i32>* store <4 x i32> %13, <4 x i32>* %16, align 4, !tbaa !7 %17 = getelementptr inbounds i32, i32* %14, i64 4 %18 = bitcast i32* %17 to <4 x i32>* store <4 x i32> %15, <4 x i32>* %18, align 4, !tbaa !7 %19 = or i64 %5, 16 %20 = add <4 x i32> %6, <i32 16, i32 16, i32 16, i32 16> %21 = getelementptr inbounds i32, i32* %3, i64 %19 %22 = add <4 x i32> %6, <i32 20, i32 20, i32 20, i32 20> %23 = bitcast i32* %21 to <4 x i32>* store <4 x i32> %20, <4 x i32>* %23, align 4, !tbaa !7 %24 = getelementptr inbounds i32, i32* %21, i64 4 %25 = bitcast i32* %24 to <4 x i32>* store <4 x i32> %22, <4 x i32>* %25, align 4, !tbaa !7 %26 = or i64 %5, 24 %27 = add <4 x i32> %6, <i32 24, i32 24, i32 24, i32 24> %28 = getelementptr inbounds i32, i32* %3, i64 %26 %29 = add <4 x i32> %6, <i32 28, i32 28, i32 28, i32 28> %30 = bitcast i32* %28 to <4 x i32>* store <4 x i32> %27, <4 x i32>* %30, align 4, !tbaa !7 %31 = getelementptr inbounds i32, i32* %28, i64 4 %32 = bitcast i32* %31 to <4 x i32>* store <4 x i32> %29, <4 x i32>* %32, align 4, !tbaa !7 %33 = add nuw nsw i64 %5, 32 %34 = add <4 x i32> %6, <i32 32, i32 32, i32 32, i32 32> %35 = icmp eq i64 %33, 1048576 br i1 %35, label %36, label %4, !llvm.loop !9 36: ; preds = %4 ret void }

Табл 3.

Анализ LLVM IR

1. При раскрутке количество итераций цикла уменьшается в восемь раз, за счёт того, что в рамках одной итерации проводиться сразу восемь итераций неоптимизированного цикла.

2. При векторизации так же в четыре раза уменьшается число итераций цикла, но делается это за счёт того, что вместо одиночной операции присваивания происходит приравнивание двух векторов памяти, что помимо прочего может быть выполнено быстрее, чем несколько отдельных операции.
3. При использовании раскрутки и векторизации, число итераций уменьшается уже в тридцать два раза за счёт использования обоих методов. Теперь за одну итерацию у нас происходит восемь операций векторного приравнивания, выполняемых для четырёх элементов за раз.

Измерение времени исполнения

Конфигурация	**Время, мс**	**Ускорение**
Без автовекторизации и раскрутки	262.884	-
Без автовекторизации, с раскруткой	268.009	0.98
С автовекторизации, без раскруткой	71.0897	3.69
С автовекторизации, без раскруткой	72.0001	3.65

Анализ времени исполнения

Наибольший прирост производительности дало использование автовекторизации без раскрутки, эффект от раскрутки оказался скорее негативным, но учитывая его минимальное влияние это можно списать на погрешность измерений и при большей продолжительности тестовой сессии, может быть достигнут положительный результат. Коэффициент ускорения от применения векторизации оказался очень значителен, и близок к ожидаемому ускорению в четыре раза. Такой результат был получен в первую очередь за счёт простоты цикла и отсутствия каких-либо зависимостей в данных между его итерациями.