ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине «Параллельные вычисления»

Выполнил Ковалев Даниил
СКБ171, вариант 10
МИЭМ НИУ ВШЭ

Содержание

1 Общее

1.1 Описание вычислительной системы

- Процессор: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- Количество ядер: 6
- Операционная система: виртуальная машина с Arch Linux
- Оперативная память: 12 ГБ
- Компилятор 1: Intel ICC: `icpc (ICC) 2021.4.0 20210910`
- Компилятор 2: Intel LLVM-based compiler: `Intel(R) oneAPI DPC++/C++ Compiler 2021.4.0 (2021.4.0.20210924)`
- Компилятор 3: GNU GCC: `g++ (GCC) 11.1.0`

1.2 Методика измерений

Получение значения счетчика тактов процессора производились с помощью инструкции `rdtsc`. Получение значения реального времени производилось с помощью библиотеки `std::chrono`. Код, реализующий эти операции, см. в приложении.

Для большей репрезентативности и улучшения воспроизводимости результатов частота процессора была заблокирована на номинальном уровне 2.20 ГГц, а использование технологии Turbo Boost — выключено.

В таблицах с результатами помимо числа тактов и наносекунд, потраченных на выполнение той или иной операции, приведено значение рассчитанной по этим числам частоты процессора.

2 Задача 1.1

Оценка максимальной производительности микропроцессора на заданных операциях (целочисленное деление, векторное целочисленное деление). Код, выполняющий поставленную задачу, см. в приложении.

2.1 Постановка задачи

- Написать программу, выполняющую многократно (в цикле) заданную операцию.

- Замерить время выполнения цикла. По результатам замера получить оценку производительности микропроцессора на заданной операции (в тактах процессора):
  - используя последовательность зависимых операций («латентность»),
  - используя последовательность независимых операций («темп выдачи результатов»).

2.2 Теоретическая часть

Использованная скалярная операция — операция деления двух знаковых 64-битных целых чисел (`std::int64_t`). Соответствует ассемблерной инструкции `idiv`.

Использованная векторная операция — `_mm256_div_epi64` из расширения AVX. Доступна с библиотекой libsvml, поставляющейся с компиляторами Intel. Производит четыре 64-битных целочисленных деления в рамках одной операции.

2.3 Результаты

Intel ICC

Операция	**Такты**	**Наносекунды**	**ГГц**
Независимая скалярная	28.4806	12.8986	2.2080
Зависимая скалярная	42.3657	19.1871	2.2080
Независимая векторная	22.7078	10.2842	2.2080
Зависимая векторная	32.4164	14.6812	2.2080

Intel LLVM-based compiler

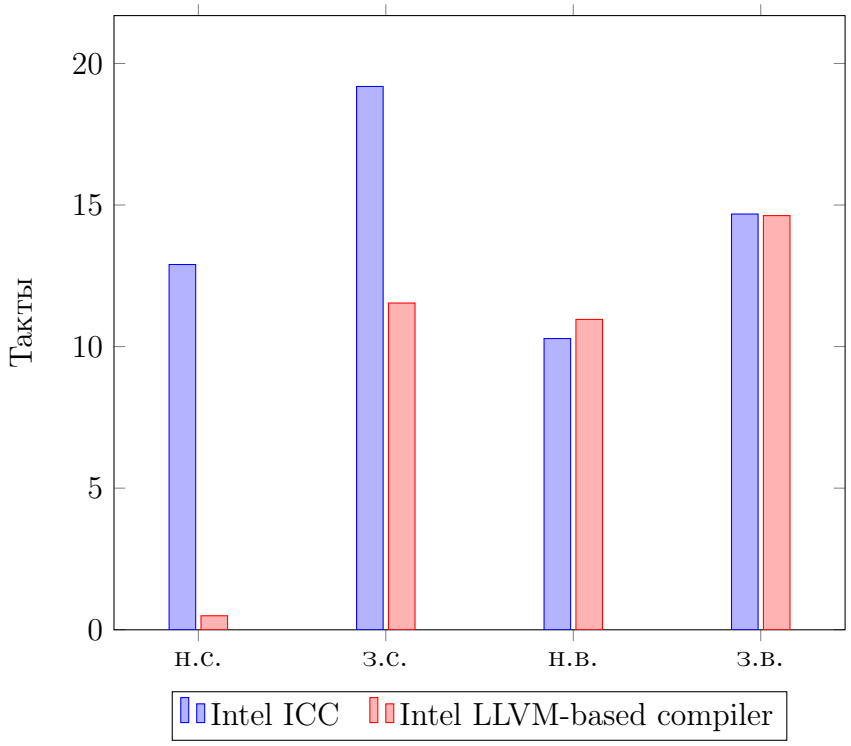
Операция	**Такты**	**Наносекунды**	**ГГц**
Независимая скалярная	1.0880	0.49274	2.2081
Зависимая скалярная	45.6729	20.6851	2.2080
Независимая векторная	24.1993	10.9597	2.2080
Зависимая векторная	32.2940	14.6258	2.2040

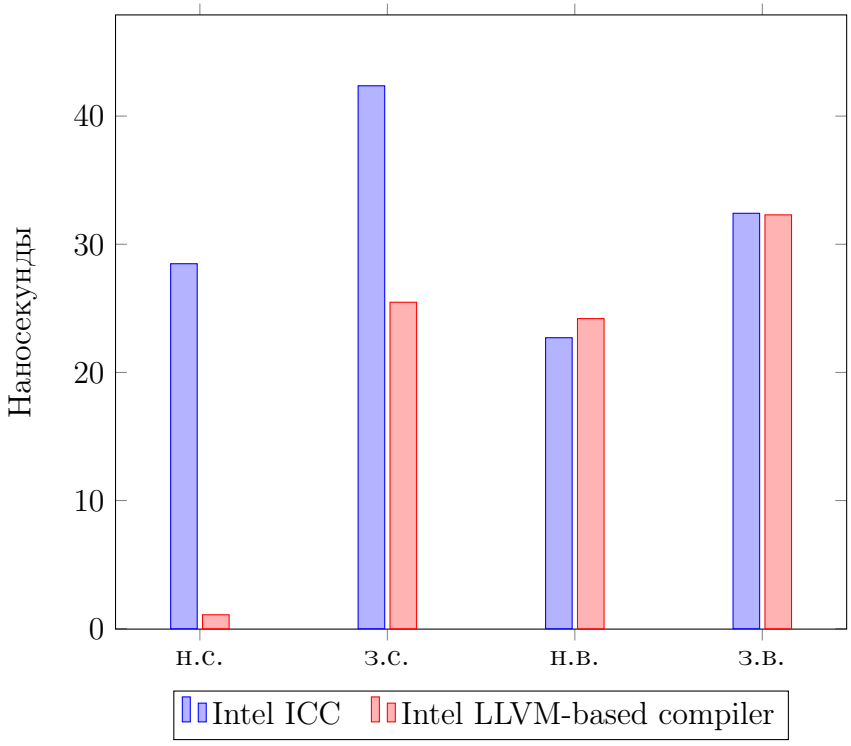
GNU GCC — результатов нет, т.к. векторная операция доступна только в библиотеке libsvml, поставляющейся с компиляторами Intel.

В диаграммах используются следующие обозначения:

- н.с. — независимая скалярная операция

- з.с. — зависимая скалярная операция
- н.в. — независимая векторная операция
- з.в. — зависимая векторная операция





2.4 Выводы

- На всех проведенных тестах фактическая частота процессора совпадает с номинальной 2.20 ГГц.
- Аномально малое время выполнения независимой скалярной операции в программе, скомпилированной Intel LLVM-based compiler, можно объяснить при помощи анализа соответствующего ассемблерного листинга (см. приложение). Несмотря на указание не оптимизировать выполнение операции внутри цикла, компилятор вынес ее за его рамки, оставив в теле цикла лишь изменение счетчика. Таким образом, измеренный показатель в 1.0880 такта на операцию итерацию — это стоимость изменение счетчика цикла и короткого jump-a.
- За исключением описанной и объясненной выше аномалии, все результаты соответствуют теоретическим предсказаниям: при прочих равных векторные операции быстрее скалярных, независимые быстрее зависимых.

3 Задача 1.2

Освоение векторизации и распараллеливания программ на системе с общей памятью. Производилась оптимизация программы, выполняющей численное интегрирование таблично заданной функции. Код, выполняющий поставленную задачу, см. в приложении.

3.1 Постановка задачи

Оптимизировать заданную программу автоматически или полуавтоматически с помощью компилятора, а также при желании с помощью intrinsics. Сравнить времена работы следующих вариантов программы:

- исходная программа, без оптимизации
- исходная программа, оптимизированная только с помощью ключей компилятора,
- программа, векторизованная полуавтоматически (с помощью директив и ключей компилятора и незначительной правки кода),
- программа, векторизованная и распараллеленная полуавтоматически,
- ...

Во всех случаях использовать ключи оптимизации, дающие наименьшее время работы программы, и наиболее позднее (эффективное) из доступных векторных расширений. По возможности обеспечить использование команд выровненного чтения и записи векторов. В программе с ручной векторизацией минимизировать количество обращений к памяти. На каждом этапе ручной оптимизации проверить, что время работы программы уменьшилось.

3.2 Теоретическая часть

В рамках решения этой задачи использовались следующие флаги и директивы компилятора.

- Флаг `-O0` — не оптимизировать код.
- Флаг `-O3` — использовать 3-й (максимальный) уровень оптимизации.
- Флаг `-march=native` — компилировать код под машину, на которой происходит сборка. Включает возможность использовать специфичные для конкретного процессора инструкции.
- Директива `#pragma omp parallel` — разделить выполнение итераций цикла между потоками.
- Директива `#pragma omp simd` — векторизовать цикл.

Было проведены замеры кода со следующими наборами флаг и директив:

- «Наивная» реализация без оптимизаций — флаг `-O0`
- «Наивная» реализация с оптимизацией — флаг `-O3`.
- «Наивная» реализация с оптимизацией под конкретную машину — флаги `-O3 -march=native`.
- Векторизованная реализация — флаги `-O3 -march=native` и директива `#pragma omp simd`.
- Распараллеленная реализация — флаги `-O3 -march=native` и директива `#pragma omp parallel`.

3.3 Результаты

Intel ICC

Интегрирование	**Такты**	**Наносекунды**	**ГГц**
«Наивное» `-O0`	339824176.06	153925526.02	2.2077
«Наивное» `-O3`	11229922.87	5106233.29	2.1993
«Наивное» `-O3 -march=native`	10845158.04	4932022.54	2.1989
Векторизованное	10828490.51	4924846.01	2.1987
Параллельное (1 п-к)	18179023.56	8253084.30	2.2027
Параллельное (2 п-ка)	9589733.79	4361244.32	2.1989
Параллельное (3 п-ка)	6985018.73	3181948.56	2.1952
Параллельное (4 п-ка)	6216703.93	2834112.84	2.1935
Параллельное (5 п-в)	5939075.67	2708988.63	2.1924
Параллельное (6 п-в)	6061977.57	2766919.99	2.1909

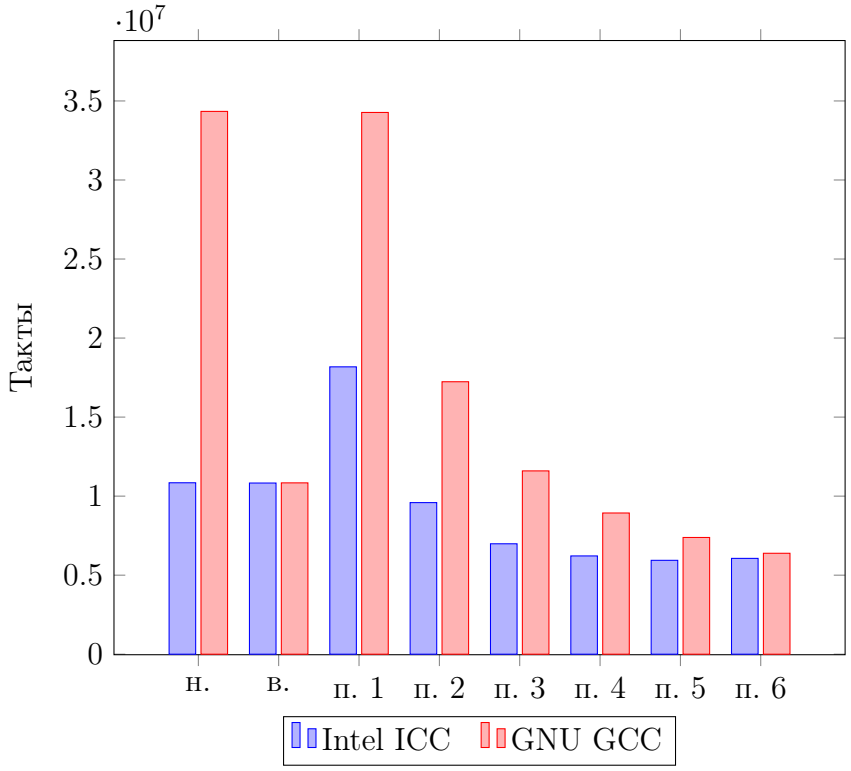
Intel LLVM-based compiler — результатов нет из-за проблем с подключением OpenMP.

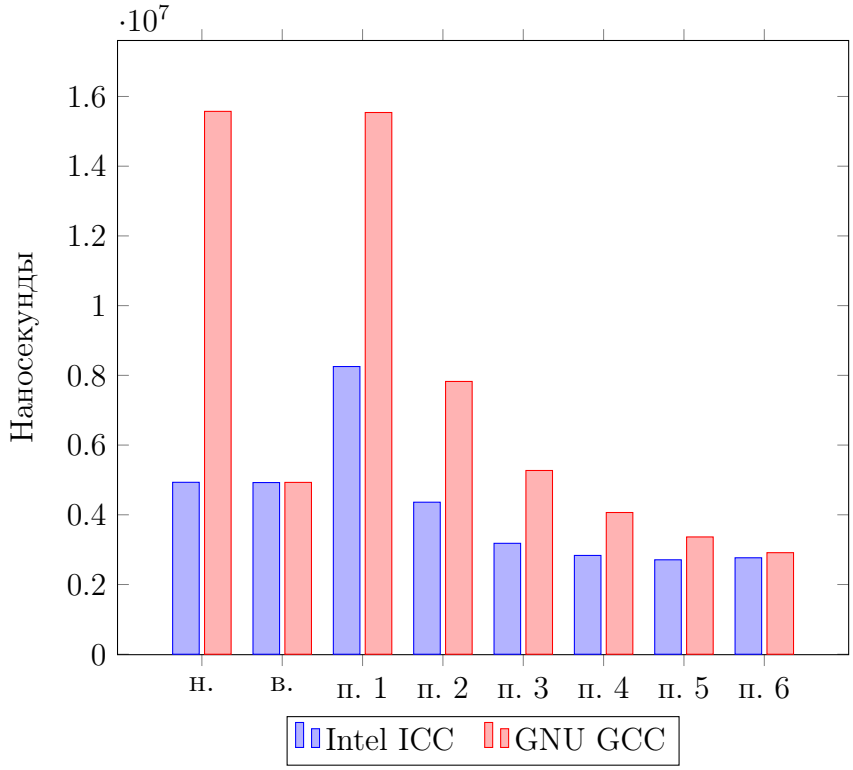
GNU GCC

Интегрирование	**Такты**	**Наносекунды**	**ГГц**
«Наивное» `-O0`	296990216.49	134525747.74	2.2077
«Наивное» `-O3`	34226829.07	15521588.21	2.2051
«Наивное» `-O3 -march=native`	34339403.18	15572950.02	2.2051
Векторизованное	10838969.24	4929447.89	2.1988
Параллельное (1 п-к)	34272412.32	15540118.80	2.2054
Параллельное (2 п-ка)	17236436.64	7825217.52	2.2027
Параллельное (3 п-ка)	11596463.97	5271108.65	2.2000
Параллельное (4 п-ка)	8933118.66	4065504.48	2.1973
Параллельное (5 п-в)	7384450.33	3364648.08	2.1947
Параллельное (6 п-в)	6384912.18	2913110.82	2.1918

В диаграммах используются следующие обозначения:

- н. — «наивная» реализация интегрирования, флаги `-O3 -march=native`
- в. — векторизованная при помощи OpenMP реализация интегрирования
- п. x — распараллеленная на x потоков при помощи OpenMP реализация интегрирования





Ниже приведены графики зависимости числа тактов процессора, потраченных на выполнение распараллеленной при помощи OpenMP операции интегрирования, от числа потоков. Используются следующие обозначения:

- Теор. 1 — график функции

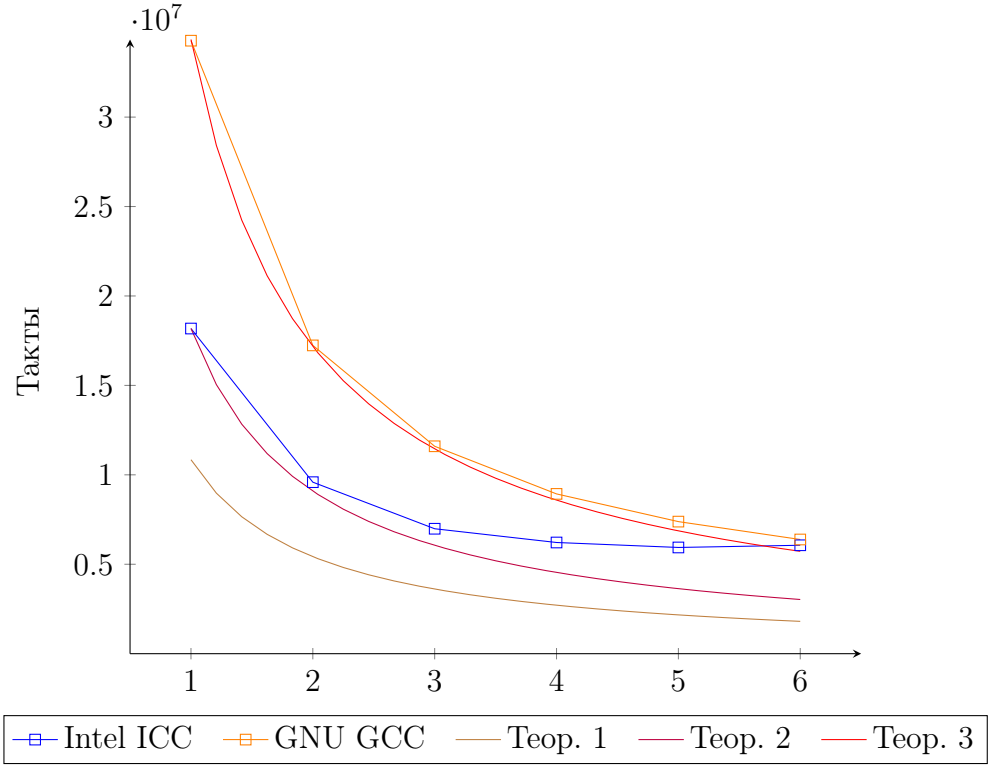
$$\frac{\text{Такты на «наивное» интегрирование (Intel ICC)}}{x}$$

- Теор. 2 — график функции

$$\frac{\text{Такты на параллельное интегрирование в 1 поток (Intel ICC)}}{x}$$

- Теор. 3 — график функции

$$\frac{\text{Такты на «наивное» интегрирование (GNU GCC)}}{x}$$



Ниже приведены графики зависимости времени (в наносекундах), потраченного на выполнение распараллеленной при помощи OpenMP операции интегрирования, от числа потоков. Используются следующие обозначения:

- Теор. 1 — график функции

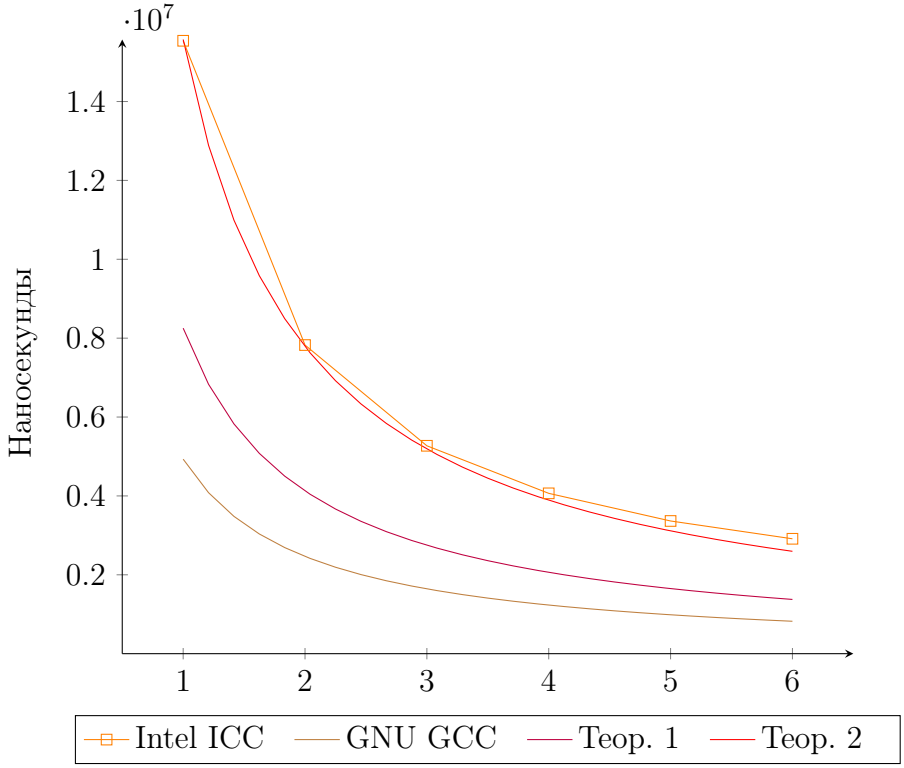
$$\frac{\text{Время на «наивное» интегрирование (Intel ICC)}}{x}$$

- Теор. 2 — график функции

$$\frac{\text{Время на параллельное интегрирование в 1 поток (Intel ICC)}}{x}$$

- Теор. 3 — график функции

$$\frac{\text{Время на «наивное» интегрирование (GNU GCC)}}{x}$$



3.4 Выводы

- На всех проведенных тестах фактическая частота процессора совпадает с номинальной 2.20 ГГц.
- Зависимость времени выполнения распараллеленной операции интегрирования от числа потоков близка к обратной пропорциональности.
- Совпадение результатов векторизованной и оптимизированной «наивной» реализаций при компиляции с помощью Intel ICC можно объяснить, проанализировав соответствующие ассемблерные листинги (см. приложение). Выясняется, что компилятор провел векторизацию самостоятельно без явного указания соответствующей директивы.
- Для программы, скомпилированной Intel ICC, параллельное выполнение в 1 поток практически в 2 раза медленнее оптимизированной «наивной» реализации. Анализ соответствующих ассемблерных листингов дает следующее объяснение: при компиляции оптимизированной «наивной» реализации были использованы векторные инструкции, в то время как в распараллеленной реализации используются скалярные.
- За исключением описанных и объясненных выше аномалий, все результаты соответствуют теоретическим предсказаниям: при увеличении уровня оптимизации «наивная» реализация начинает работать быстрее, при прочих равных векторизованное и распараллеленное интегрирование быстрее «наивного», а увеличение числа потоков приводит к обратно пропорцио-

нальному уменьшению времени работы.

- Код, скомпилированный Intel ICC, на всех тестах (кроме выполнения неоптимизированной «наивной» реализации) показывает более хорошие результаты, чем код, скомпилированный GNU GCC.

4 Приложение

4.1 Исходные коды

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)

project(task1 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_library(tools INTERFACE tools.hpp)

option(BUILD_TASK_1_1 "Build task 1.1 (requires Intel compiler)" ON)
option(BUILD_TASK_1_2 "Build task 1.2 (requires OpenMP)"         ON)

if(BUILD_TASK_1_1)
    if(NOT(CMAKE_CXX_COMPILER_ID STREQUAL "Intel" OR CMAKE_CXX_COMPILER_ID
    ↪  STREQUAL "IntelLLVM"))
        message(WARNING "Compiler is not ICPC or Intel LLVM, you might face
        ↪  problems when compiling")
    endif()
    add_executable(task1.1 task1.1.cpp)
    target_link_libraries(task1.1 tools)
endif()

if(BUILD_TASK_1_2)
    find_package(OpenMP REQUIRED)
    add_executable(task1.2 task1.2.cpp)
    target_link_libraries(task1.2 tools)
    target_link_libraries(task1.2 OpenMP::OpenMP_CXX)
endif()
```

`tools.hpp`

```
#ifndef PARALLEL_COMPUTING_TASK1_HPP_
#define PARALLEL_COMPUTING_TASK1_HPP_

#include <chrono>
#include <cstdint>
#include <cstdlib>
```

```
7
8  namespace my
9  {
10
11 template <typename T>
12 inline __attribute__((always_inline)) void do_not_optimize(T& value)
13 {
14 #if defined(__clang__)
15     asm volatile("" : "+r,m"(value) : : "memory");
16 #else
17     asm volatile("" : "+m,r"(value) : : "memory");
18 #endif
19 }
20
21 inline __attribute__((always_inline)) std::uint64_t ticks()
22 {
23     std::uint64_t tsc;
24     asm volatile("mfence; "         // memory barrier
25                  "rdtsc; "          // read of tsc
26                  "shl $32,%%rdx; "  // shift higher 32 bits stored in rdx up
27                  "or %%rdx,%%rax"   // and or onto rax
28                  : "=a"(tsc)        // output to tsc
29                  :
30                  : "%rcx", "%rdx", "memory");
31     return tsc;
32 }
33
34 struct BenchmarkResult
35 {
36     double ticks;
37     double nanoseconds;
38 };
39
40 class Timer
41 {
42 public:
43     Timer(BenchmarkResult& result, std::size_t iterations_count = 1)
44         : m_result(result)
45         , m_iterations_count(iterations_count)
46     {
47         m_time_before = std::chrono::high_resolution_clock::now();
48         m_ticks_before = ticks();
49     }
50
51     ~Timer()
52     {
53         try
54         {
55             using namespace std::chrono;
56             std::uint64_t ticks_after = ticks();
57             high_resolution_clock::time_point time_after =
↪   high_resolution_clock::now();
```

```
58              m_result.ticks = (ticks_after - m_ticks_before) /
    ↪  static_cast<double>(m_iterations_count);
59              m_result.nanoseconds = duration_cast<nanoseconds>(time_after -
    ↪  m_time_before).count() / static_cast<double>(m_iterations_count);
60          }
61          catch (...)
62          {
63              std::exit(EXIT_FAILURE);
64          }
65      }
66
67  private:
68      BenchmarkResult& m_result;
69      std::size_t m_iterations_count;
70      std::uint64_t m_ticks_before;
71      std::chrono::high_resolution_clock::time_point m_time_before;
72  };
73
74  }  // namespace my
75
76  #endif  // PARALLEL_COMPUTING_TASK1_HPP_
```

task1.1.cpp

```
1   #include "tools.hpp"
2
3   #include <immintrin.h>
4
5   #include <cstdint>
6   #include <cstdlib>
7   #include <cstring>
8   #include <functional>
9   #include <iomanip>
10  #include <iostream>
11  #include <limits>
12  #include <string>
13  #include <string_view>
14
15  static_assert (sizeof(long long) == sizeof(std::int64_t), "long long and
    ↪  std::int64_t do not match");
16  static_assert (sizeof(__m256i) % sizeof(std::int64_t) == 0, "Size of __m256i is
    ↪  not multiple of size of std::int64_t");
17  static constexpr std::size_t SCALARS_IN_VECTOR = sizeof(__m256i) /
    ↪  sizeof(std::int64_t);
18
19  static constexpr std::size_t ITERATIONS_COUNT = 10'000'000'000;
20
21  my::BenchmarkResult independent_scalar_operation(std::int64_t n1, std::int64_t
    ↪  n2)
22  {
23      asm volatile("# independent_scalar_operation enter");
24      my::BenchmarkResult result;
```

```
25      {
26          my::Timer timer(result, ITERATIONS_COUNT);
27          for (std::size_t i = 0; i < ITERATIONS_COUNT; ++i)
28          {
29              std::int64_t n3 = n1 / n2;
30              my::do_not_optimize(n3);
31          }
32      }
33      asm volatile("# independent_scalar_operation exit");
34      return result;
35  }
36
37  my::BenchmarkResult dependent_scalar_operation(std::int64_t n1, std::int64_t
    ↪  n2)
38  {
39      asm volatile("# dependent_scalar_operation enter");
40      my::BenchmarkResult result;
41      {
42          my::Timer timer(result, ITERATIONS_COUNT);
43          for (std::size_t i = 0; i < ITERATIONS_COUNT; ++i)
44          {
45              n1 /= n2;
46          }
47          my::do_not_optimize(n1);
48      }
49      asm volatile("# dependent_scalar_operation exit");
50      return result;
51  }
52
53  void put_scalar_into_vector(std::int64_t scalar, __m256i& vector)
54  {
55      std::int64_t array[SCALARS_IN_VECTOR];
56      for (std::size_t i = 0; i < SCALARS_IN_VECTOR; ++i)
57          array[i] = scalar;
58      std::memcpy(&vector, array, sizeof(vector));
59  }
60
61  my::BenchmarkResult independent_vector_operation(std::int64_t n1, std::int64_t
    ↪  n2)
62  {
63      asm volatile("# independent_vector_operation enter\n");
64
65      __m256i v1, v2;
66      put_scalar_into_vector(n1, v1);
67      put_scalar_into_vector(n2, v2);
68
69      static constexpr std::size_t vector_iterations_count = ITERATIONS_COUNT /
    ↪  SCALARS_IN_VECTOR;
70      my::BenchmarkResult result;
71      {
72          my::Timer timer(result, ITERATIONS_COUNT);
73          for (std::size_t i = 0; i < vector_iterations_count; ++i)
74          {
```

```
            __m256i v3 = _mm256_div_epi64(v1, v2);
            my::do_not_optimize(v3);
        }
    }
    asm volatile("# independent_vector_operation exit");
    return result;
}

my::BenchmarkResult dependent_vector_operation(std::int64_t n1, std::int64_t
↪  n2)
{
    asm volatile("# dependent_vector_operation enter\n");

    __m256i v1, v2;
    put_scalar_into_vector(n1, v1);
    put_scalar_into_vector(n2, v2);

    static constexpr std::size_t vector_iterations_count = ITERATIONS_COUNT /
↪  SCALARS_IN_VECTOR;
    my::BenchmarkResult result;
    {
        my::Timer timer(result, ITERATIONS_COUNT);
        for (std::size_t i = 0; i < vector_iterations_count; ++i)
        {
            v1 = _mm256_div_epi64(v1, v2);
        }
        my::do_not_optimize(v1);
    }
    asm volatile("# dependent_vector_operation exit");
    return result;
}

struct Params
{
    std::int64_t n1;
    std::int64_t n2;
};

Params parse_cmd_line(int argc, char* argv[])
{
    if (argc != 3)
    {
        throw std::invalid_argument("Error: you must pass 2 integers as
↪  CLI-arguments");
    }

    auto int_from_string = [](const std::string& s) -> std::int64_t
    {
        try
        {
            std::size_t pos;
            int n = std::stoll(s, &pos);
            if (pos != s.size())
```

```
125             {
126                 throw std::invalid_argument("stoll");
127             }
128             return n;
129         }
130         catch (const std::invalid_argument& e)
131         {
132             throw std::invalid_argument("Error: cannot convert \"" + s + "\" to
    ↪  std::int64_t: \"" + e.what() + "\"");
133         }
134         catch (const std::out_of_range& e)
135         {
136             throw std::out_of_range("Error: value \"" + s + "\" is out of range
    ↪  of type std::int64_t: \"" + e.what() + "\"");
137         }
138     };
139 
140     return Params{ .n1 = int_from_string(std::string(argv[1])),
141                    .n2 = int_from_string(std::string(argv[2])) };
142 }
143 
144 void print_table_row(std::string_view label, my::BenchmarkResult result)
145 {
146     std::cout << "| " << label << " | "
147               << std::setw(12) << result.ticks << " | "
148               << std::setw(9)  << result.nanoseconds << " |" << std::endl
149               << "+--------------------+--------------+-----------+" <<
    ↪  std::endl;
150 }
151 
152 int main(int argc, char* argv[]) try
153 {
154     Params params = parse_cmd_line(argc, argv);
155 
156     my::BenchmarkResult independent_scalar_result =
    ↪  independent_scalar_operation(params.n1, params.n2);
157     my::BenchmarkResult dependent_scalar_result =
    ↪  dependent_scalar_operation(params.n1, params.n2);
158     my::BenchmarkResult independent_vector_result =
    ↪  independent_vector_operation(params.n1, params.n2);
159     my::BenchmarkResult dependent_vector_result =
    ↪  dependent_vector_operation(params.n1, params.n2);
160 
161     std::cout << "+--------------------+--------------+-----------+" <<
    ↪  std::endl
162               << "|      operation     | ticks / iter | ns / iter |" <<
    ↪  std::endl
163               << "+--------------------+--------------+-----------+" <<
    ↪  std::endl;
164     print_table_row("independent scalar", independent_scalar_result);
165     print_table_row(" dependent scalar ", dependent_scalar_result);
166     print_table_row("independent vector", independent_vector_result);
167     print_table_row(" dependent vector ", dependent_vector_result);
```

```
168
169     return EXIT_SUCCESS;
170 }
171 catch (const std::exception& e)
172 {
173     std::cerr << e.what() << "\nAborting\n";
174     return EXIT_FAILURE;
175 }
```

task1.2.cpp

```
1  #include "tools.hpp"
2
3  #include <immintrin.h>
4
5  #include <cstdint>
6  #include <cstdlib>
7  #include <cstring>
8  #include <functional>
9  #include <iomanip>
10 #include <iostream>
11 #include <limits>
12 #include <string>
13 #include <string_view>
14
15 static_assert (sizeof(long long) == sizeof(std::int64_t), "long long and
   ↪  std::int64_t do not match");
16 static_assert (sizeof(__m256i) % sizeof(std::int64_t) == 0, "Size of __m256i is
   ↪  not multiple of size of std::int64_t");
17 static constexpr std::size_t SCALARS_IN_VECTOR = sizeof(__m256i) /
   ↪  sizeof(std::int64_t);
18
19 static constexpr std::size_t ITERATIONS_COUNT = 10'000'000'000;
20
21 my::BenchmarkResult independent_scalar_operation(std::int64_t n1, std::int64_t
   ↪  n2)
22 {
23     asm volatile("# independent_scalar_operation enter");
24     my::BenchmarkResult result;
25     {
26         my::Timer timer(result, ITERATIONS_COUNT);
27         for (std::size_t i = 0; i < ITERATIONS_COUNT; ++i)
28         {
29             std::int64_t n3 = n1 / n2;
30             my::do_not_optimize(n3);
31         }
32     }
33     asm volatile("# independent_scalar_operation exit");
34     return result;
35 }
36
37 my::BenchmarkResult dependent_scalar_operation(std::int64_t n1, std::int64_t
   ↪  n2)
```

```
{
    asm volatile("# dependent_scalar_operation enter");
    my::BenchmarkResult result;
    {
        my::Timer timer(result, ITERATIONS_COUNT);
        for (std::size_t i = 0; i < ITERATIONS_COUNT; ++i)
        {
            n1 /= n2;
        }
        my::do_not_optimize(n1);
    }
    asm volatile("# dependent_scalar_operation exit");
    return result;
}

void put_scalar_into_vector(std::int64_t scalar, __m256i& vector)
{
    std::int64_t array[SCALARS_IN_VECTOR];
    for (std::size_t i = 0; i < SCALARS_IN_VECTOR; ++i)
        array[i] = scalar;
    std::memcpy(&vector, array, sizeof(vector));
}

my::BenchmarkResult independent_vector_operation(std::int64_t n1, std::int64_t
↪   n2)
{
    asm volatile("# independent_vector_operation enter\n");

    __m256i v1, v2;
    put_scalar_into_vector(n1, v1);
    put_scalar_into_vector(n2, v2);

    static constexpr std::size_t vector_iterations_count = ITERATIONS_COUNT /
↪   SCALARS_IN_VECTOR;
    my::BenchmarkResult result;
    {
        my::Timer timer(result, ITERATIONS_COUNT);
        for (std::size_t i = 0; i < vector_iterations_count; ++i)
        {
            __m256i v3 = _mm256_div_epi64(v1, v2);
            my::do_not_optimize(v3);
        }
    }
    asm volatile("# independent_vector_operation exit");
    return result;
}

my::BenchmarkResult dependent_vector_operation(std::int64_t n1, std::int64_t
↪   n2)
{
    asm volatile("# dependent_vector_operation enter\n");

    __m256i v1, v2;
```

```
88      put_scalar_into_vector(n1, v1);
89      put_scalar_into_vector(n2, v2);
90
91      static constexpr std::size_t vector_iterations_count = ITERATIONS_COUNT /
    ↪   SCALARS_IN_VECTOR;
92      my::BenchmarkResult result;
93      {
94          my::Timer timer(result, ITERATIONS_COUNT);
95          for (std::size_t i = 0; i < vector_iterations_count; ++i)
96          {
97              v1 = _mm256_div_epi64(v1, v2);
98          }
99          my::do_not_optimize(v1);
100     }
101     asm volatile("# dependent_vector_operation exit");
102     return result;
103 }
104
105 struct Params
106 {
107     std::int64_t n1;
108     std::int64_t n2;
109 };
110
111 Params parse_cmd_line(int argc, char* argv[])
112 {
113     if (argc != 3)
114     {
115         throw std::invalid_argument("Error: you must pass 2 integers as
    ↪   CLI-arguments");
116     }
117
118     auto int_from_string = [](const std::string& s) -> std::int64_t
119     {
120         try
121         {
122             std::size_t pos;
123             int n = std::stoll(s, &pos);
124             if (pos != s.size())
125             {
126                 throw std::invalid_argument("stoll");
127             }
128             return n;
129         }
130         catch (const std::invalid_argument& e)
131         {
132             throw std::invalid_argument("Error: cannot convert \"" + s + "\" to
    ↪   std::int64_t: \"" + e.what() + "\"");
133         }
134         catch (const std::out_of_range& e)
135         {
136             throw std::out_of_range("Error: value \"" + s + "\" is out of range
    ↪   of type std::int64_t: \"" + e.what() + "\"");
```

```
137         }
138     };
139 
140     return Params{ .n1 = int_from_string(std::string(argv[1])),
141                    .n2 = int_from_string(std::string(argv[2])) };
142 }
143 
144 void print_table_row(std::string_view label, my::BenchmarkResult result)
145 {
146     std::cout << "| " << label << " | "
147               << std::setw(12) << result.ticks << " | "
148               << std::setw(9)  << result.nanoseconds << " |" << std::endl
149               << "+--------------------+-------------+-----------+" <<
    ↪  std::endl;
150 }
151 
152 int main(int argc, char* argv[]) try
153 {
154     Params params = parse_cmd_line(argc, argv);
155 
156     my::BenchmarkResult independent_scalar_result =
    ↪  independent_scalar_operation(params.n1, params.n2);
157     my::BenchmarkResult dependent_scalar_result =
    ↪  dependent_scalar_operation(params.n1, params.n2);
158     my::BenchmarkResult independent_vector_result =
    ↪  independent_vector_operation(params.n1, params.n2);
159     my::BenchmarkResult dependent_vector_result =
    ↪  dependent_vector_operation(params.n1, params.n2);
160 
161     std::cout << "+--------------------+-------------+-----------+" <<
    ↪  std::endl
162               << "|      operation     | ticks / iter | ns / iter |" <<
    ↪  std::endl
163               << "+--------------------+-------------+-----------+" <<
    ↪  std::endl;
164     print_table_row("independent scalar", independent_scalar_result);
165     print_table_row(" dependent scalar ", dependent_scalar_result);
166     print_table_row("independent vector", independent_vector_result);
167     print_table_row(" dependent vector ", dependent_vector_result);
168 
169     return EXIT_SUCCESS;
170 }
171 catch (const std::exception& e)
172 {
173     std::cerr << e.what() << "\nAborting\n";
174     return EXIT_FAILURE;
175 }
```

4.2 Ассемблерные листинги

4.2.1 Intel ICC

Скалярное деление (независимые операции)

```
# Begin ASM
# # independent_scalar_operation enter
# End ASM
                                # LOE
..B1.153:                       # Preds ..B1.3
                                # Execution count [1.00e+00]
        movq      $0, 56(%rsp)                                  #156.93
                                # LOE
..B1.4:                         # Preds ..B1.153
                                # Execution count [1.00e+00]
..___tag_value_main.15:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #156.93
..___tag_value_main.16:
                                # LOE rax
..B1.5:                         # Preds ..B1.4
                                # Execution count [1.00e+00]
        movq      %rax, 56(%rsp)                                #156.93
        movq      $0x174876e800, %r12                           #156.93
        mfence                                                  #156.93
        rdtsc                                                   #156.93
        shlq      $32, %rdx                                     #156.93
        orq       %rdx, %rax                                    #156.93
        movq      %rax, 96(%rsp)                                #156.93[spill]
        xorl      %ecx, %ecx                                    #156.93
                                # LOE rcx r12
..B1.6:                         # Preds ..B1.6 ..B1.5
                                # Execution count [5.56e+00]
        movq      160(%rsp), %rax                               #156.82
        incq      %rcx                                          #156.93
        cqto                                                    #156.93
        idivq     168(%rsp)                                     #156.93
        movq      %rax, (%rsp)                                  #156.93
        cmpq      %rcx, %r12                                    #156.93
        ja        ..B1.6        # Prob 82%                      #156.93
                                # LOE rcx r12
..B1.7:                         # Preds ..B1.6
                                # Execution count [1.00e+00]
        mfence                                                  #156.93
        rdtsc                                                   #156.93
        shlq      $32, %rdx                                     #156.93
        orq       %rdx, %rax                                    #156.93
        movq      %rax, %rbx                                    #156.93
..___tag_value_main.18:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #156.93
```

```
..___tag_value_main.19:
                                # LOE rax rbx r12
..B1.155:                       # Preds ..B1.7
                                # Execution count [1.00e+00]
        movq      %rax, 64(%rsp)                                #156.93
                                # LOE rbx r12
..B1.8:                         # Preds ..B1.155
                                # Execution count [1.00e+00]
        movq      64(%rsp), %rax                                #156.93
        vxorpd    %xmm0, %xmm0, %xmm0                           #156.93
        subq      56(%rsp), %rax                                #156.93
        vcvtsi2sdq %rax, %xmm0, %xmm0                           #156.93
        vdivsd    .L_2il0floatpacket.8(%rip), %xmm0, %xmm1      #156.93
        vmovsd    %xmm1, 104(%rsp)                              #156.93[spill]
                                # LOE rbx r12
..B1.159:                       # Preds ..B1.8
                                # Execution count [1.00e+00]
# Begin ASM
# # independent_scalar_operation exit
# End ASM
```

Скалярное деление (зависимые операции)

```
# Begin ASM
# # dependent_scalar_operation enter
# End ASM
                                # LOE rbx r12
..B1.156:                       # Preds ..B1.157
                                # Execution count [1.00e+00]
        movq      $0, 24(%rsp)                                  #157.89
                                # LOE rbx r12
..B1.9:                         # Preds ..B1.156
                                # Execution count [1.00e+00]
..___tag_value_main.21:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #157.89
..___tag_value_main.22:
                                # LOE rax rbx r12
..B1.10:                        # Preds ..B1.9
                                # Execution count [1.00e+00]
        movq      %rax, 24(%rsp)                                #157.89
        xorl      %esi, %esi                                    #157.89
        mfence                                                  #157.89
        rdtsc                                                   #157.89
        shlq      $32, %rdx                                     #157.89
        orq       %rdx, %rax                                    #157.89
        movq      %rax, 112(%rsp)                               #157.89[spill]
        movq      168(%rsp), %rcx                               #157.89
        movq      32(%rsp), %rax                                #157.89
                                # LOE rax rcx rbx rsi r12
..B1.11:                        # Preds ..B1.11 ..B1.10
                                # Execution count [5.56e+00]
```

```
        cqto                                                    #157.89
        incq      %rsi                                          #157.89
        idivq     %rcx                                          #157.89
        cmpq      %rsi, %r12                                    #157.89
        ja        ..B1.11       # Prob 82%                      #157.89
                                # LOE rax rcx rbx rsi r12
..B1.12:                        # Preds ..B1.11
                                # Execution count [1.00e+00]
        movq      %rax, 32(%rsp)                                #157.89
        mfence                                                  #157.89
        rdtsc                                                   #157.89
        shlq      $32, %rdx                                     #157.89
        orq       %rdx, %rax                                    #157.89
        movq      %rax, %r14                                    #157.89
..___tag_value_main.24:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #157.89
..___tag_value_main.25:
                                # LOE rax rbx r14
..B1.161:                       # Preds ..B1.12
                                # Execution count [1.00e+00]
        movq      %rax, 32(%rsp)                                #157.89
                                # LOE rbx r14
..B1.13:                        # Preds ..B1.161
                                # Execution count [1.00e+00]
        movq      32(%rsp), %rax                                #157.89
        vxorpd    %xmm0, %xmm0, %xmm0                           #157.89
        subq      24(%rsp), %rax                                #157.89
        vcvtsi2sdq %rax, %xmm0, %xmm0                           #157.89
        vdivsd    .L_2il0floatpacket.8(%rip), %xmm0, %xmm1      #157.89
        vmovsd    %xmm1, 120(%rsp)                              #157.89[spill]
                                # LOE rbx r14
..B1.163:                       # Preds ..B1.13
                                # Execution count [1.00e+00]
# Begin ASM
# # dependent_scalar_operation exit
# End ASM
```

Векторное деление (независимые операции)

```
# Begin ASM
# # independent_vector_operation enter
# End ASM
                                # LOE rbx r14
..B1.162:                       # Preds ..B1.163
                                # Execution count [1.00e+00]
        vpbroadcastq 160(%rsp), %ymm0                           #158.82
        vmovups   %ymm0, 64(%rsp)                               #158.93
                                # LOE rbx r14
..B1.14:                        # Preds ..B1.162
                                # Execution count [1.00e+00]
        vpbroadcastq 168(%rsp), %ymm0                           #158.93
```

```
        vmovups   %ymm0, 32(%rsp)                                #158.93
                                # LOE rbx r14
..B1.15:                        # Preds ..B1.14
                                # Execution count [1.00e+00]
        movq      $0, 24(%rsp)                                  #158.93
                                # LOE rbx r14
..B1.16:                        # Preds ..B1.15
                                # Execution count [1.00e+00]
        vzeroupper                                              #158.93
..___tag_value_main.27:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #158.93
..___tag_value_main.28:
                                # LOE rax rbx r14
..B1.17:                        # Preds ..B1.16
                                # Execution count [1.00e+00]
        movq      %rax, 24(%rsp)                                #158.93
        xorl      %esi, %esi                                    #158.93
        mfence                                                  #158.93
        rdtsc                                                   #158.93
        shlq      $32, %rdx                                     #158.93
        orq       %rdx, %rax                                    #158.93
        movq      $0x5d21dba00, %r15                            #158.93
        movq      %rax, %r13                                    #158.93
                                # LOE rbx rsi r13 r14 r15
..B1.18:                        # Preds ..B1.165 ..B1.17
                                # Execution count [5.56e+00]
        vmovdqu   64(%rsp), %ymm0                               #158.93
        vmovdqu   32(%rsp), %ymm1                               #158.93
..___tag_value_main.29:
        call      *__svml_i64div4@GOTPCREL(%rip)                #158.93
..___tag_value_main.30:
                                # LOE rbx rsi r13 r14 r15 ymm0
..B1.165:                       # Preds ..B1.18
                                # Execution count [5.56e+00]
        vmovdqu   %ymm0, 128(%rsp)                              #158.93
        incq      %rsi                                          #158.93
        cmpq      %rsi, %r15                                    #158.93
        ja        ..B1.18       # Prob 82%                      #158.93
                                # LOE rbx rsi r13 r14 r15
..B1.19:                        # Preds ..B1.165
                                # Execution count [1.00e+00]
        mfence                                                  #158.93
        rdtsc                                                   #158.93
        shlq      $32, %rdx                                     #158.93
        orq       %rdx, %rax                                    #158.93
        movq      %rax, %r12                                    #158.93
        vzeroupper                                              #158.93
..___tag_value_main.31:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #158.93
..___tag_value_main.32:
                                # LOE rax rbx r12 r13 r14 r15
```

```
..B1.166:                       # Preds ..B1.19
                                # Execution count [1.00e+00]
        movq      %rax, 64(%rsp)                                #158.93
                                # LOE rbx r12 r13 r14 r15
..B1.20:                        # Preds ..B1.166
                                # Execution count [1.00e+00]
        movq      64(%rsp), %rax                                #158.93
        vxorpd    %xmm0, %xmm0, %xmm0                           #158.93
        subq      24(%rsp), %rax                                #158.93
        vcvtsi2sdq %rax, %xmm0, %xmm0                           #158.93
        vdivsd    .L_2il0floatpacket.8(%rip), %xmm0, %xmm1      #158.93
        vmovsd    %xmm1, 128(%rsp)                              #158.93[spill]
                                # LOE rbx r12 r13 r14 r15
..B1.168:                       # Preds ..B1.20
                                # Execution count [1.00e+00]
# Begin ASM
# # independent_vector_operation exit
# End ASM
```

Векторное деление (зависимые операции)

```
# Begin ASM
# # dependent_vector_operation enter
# End ASM
                                # LOE rbx r12 r13 r14 r15
..B1.167:                       # Preds ..B1.168
                                # Execution count [1.00e+00]
        vpbroadcastq 160(%rsp), %ymm0                           #159.78
        vmovups   %ymm0, 32(%rsp)                               #159.89
                                # LOE rbx r12 r13 r14 r15
..B1.21:                        # Preds ..B1.167
                                # Execution count [1.00e+00]
        vpbroadcastq 168(%rsp), %ymm0                           #159.89
        vmovups   %ymm0, (%rsp)                                 #159.89
                                # LOE rbx r12 r13 r14 r15
..B1.22:                        # Preds ..B1.21
                                # Execution count [1.00e+00]
        movq      $0, 88(%rsp)                                  #159.89
                                # LOE rbx r12 r13 r14 r15
..B1.23:                        # Preds ..B1.22
                                # Execution count [1.00e+00]
        vzeroupper                                              #159.89
..___tag_value_main.34:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #159.89
..___tag_value_main.35:
                                # LOE rax rbx r12 r13 r14 r15
..B1.24:                        # Preds ..B1.23
                                # Execution count [1.00e+00]
        movq      %rax, 88(%rsp)                                #159.89
        xorl      %esi, %esi                                    #159.89
        mfence                                                  #159.89
```

```
        rdtsc                                                   #159.89
        shlq      $32, %rdx                                     #159.89
        orq       %rdx, %rax                                    #159.89
        movq      %rax, 136(%rsp)                               #159.89[spill]
        vmovdqu   32(%rsp), %ymm0                               #159.89
        vmovdqu   (%rsp), %ymm8                                 #159.89
                                # LOE rbx rsi r12 r13 r14 r15 ymm0 ymm8
..B1.25:                        # Preds ..B1.170 ..B1.24
                                # Execution count [5.56e+00]
        vmovdqa   %ymm8, %ymm1                                  #159.89
..___tag_value_main.37:
        call      *__svml_i64div4@GOTPCREL(%rip)                #159.89
..___tag_value_main.38:
                                # LOE rbx rsi r12 r13 r14 r15 ymm0 ymm8
..B1.170:                       # Preds ..B1.25
                                # Execution count [5.56e+00]
        incq      %rsi                                          #159.89
        cmpq      %rsi, %r15                                    #159.89
        ja        ..B1.25       # Prob 82%                      #159.89
                                # LOE rbx rsi r12 r13 r14 r15 ymm0 ymm8
..B1.26:                        # Preds ..B1.170
                                # Execution count [1.00e+00]
        vmovdqu   %ymm0, 32(%rsp)                               #159.89
        mfence                                                  #159.89
        rdtsc                                                   #159.89
        shlq      $32, %rdx                                     #159.89
        orq       %rdx, %rax                                    #159.89
        movq      %rax, %r15                                    #159.89
        vzeroupper                                              #159.89
..___tag_value_main.39:
#       std::chrono::_V2::system_clock::now() noexcept
        call      _ZNSt6chrono3_V212system_clock3nowEv          #159.89
..___tag_value_main.40:
                                # LOE rax rbx r12 r13 r14 r15
..B1.171:                       # Preds ..B1.26
                                # Execution count [1.00e+00]
        movq      %rax, (%rsp)                                  #159.89
                                # LOE rbx r12 r13 r14 r15
..B1.27:                        # Preds ..B1.171
                                # Execution count [1.00e+00]
        movq      (%rsp), %rax                                  #159.89
        vxorpd    %xmm0, %xmm0, %xmm0                           #159.89
        subq      88(%rsp), %rax                                #159.89
        vcvtsi2sdq %rax, %xmm0, %xmm0                           #159.89
        vdivsd    .L_2il0floatpacket.8(%rip), %xmm0, %xmm1      #159.89
        vmovsd    %xmm1, 16(%rsp)                               #159.89[spill]
                                # LOE rbx r12 r13 r14 r15
..B1.173:                       # Preds ..B1.27
                                # Execution count [1.00e+00]
# Begin ASM
# # dependent_vector_operation exit
# End ASM
```

«Наивное» интегрирование

```
# Begin ASM
# # integrate_dummy enter
# End ASM
                                # LOE rbx rbp r12 r13 r14 r15
..B17.32:                       # Preds ..B17.33
                                # Execution count [1.00e+00]
        movq      (%r12), %rdx                                  #21.25
        movq      8(%r12), %rax                                 #21.25
        vxorpd    %xmm0, %xmm0, %xmm0                           #20.16
        vmovsd    %xmm0, 8(%rsp)                                #20.16[spill]
        cmpq      %rax, %rdx                                    #21.25
        je        ..B17.24      # Prob 50%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.2:                        # Preds ..B17.32
                                # Execution count [9.00e-01]
        subq      %rdx, %rax                                    #21.25
        addq      $7, %rax                                      #21.25
        shrq      $3, %rax                                      #21.25
        cmpq      $16, %rax                                     #21.25
        jb        ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.3:                        # Preds ..B17.2
                                # Execution count [9.00e-01]
        cmpq      $29, %rax                                     #21.25
        jb        ..B17.28      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.4:                        # Preds ..B17.3
                                # Execution count [9.00e-01]
        movq      %rdx, %rdi                                    #21.25
        andq      $31, %rdi                                     #21.25
        je        ..B17.11      # Prob 50%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.5:                        # Preds ..B17.4
                                # Execution count [9.00e-01]
        testq     $7, %rdi                                      #21.25
        jne       ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.6:                        # Preds ..B17.5
                                # Execution count [4.50e-01]
        negq      %rdi                                          #21.25
        addq      $32, %rdi                                     #21.25
        shrq      $3, %rdi                                      #21.25
        lea       16(%rdi), %rcx                                #21.25
        cmpq      %rcx, %rax                                    #21.25
        jb        ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.7:                        # Preds ..B17.6
                                # Execution count [9.00e-01]
        movq      %rax, %rcx                                    #21.25
        xorl      %esi, %esi                                    #21.25
        subq      %rdi, %rcx                                    #21.25
```

```
52          andq      $15, %rcx                                     #21.25
53          negq      %rcx                                          #21.25
54          addq      %rax, %rcx                                    #21.25
55          testq     %rdi, %rdi                                    #21.25
56          je        ..B17.12      # Prob 9%                       #21.25
57                                    # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                      ↪  xmm0
58  ..B17.8:                          # Preds ..B17.7
59                                    # Execution count [9.00e-01]
60          vmovapd   %xmm0, %xmm1                                  #
61          vmovsd    (%rsp), %xmm2                                 #[spill]
62                                    # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                      ↪  xmm1 xmm2
63  ..B17.9:                          # Preds ..B17.9 ..B17.8
64                                    # Execution count [5.00e+00]
65          vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #23.9
66          incq      %rsi                                          #21.25
67          vaddsd    %xmm1, %xmm0, %xmm1                           #23.9
68          cmpq      %rdi, %rsi                                    #21.25
69          jb        ..B17.9       # Prob 82%                      #21.25
70                                    # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                      ↪  xmm1 xmm2
71  ..B17.10:                         # Preds ..B17.9
72                                    # Execution count [9.00e-01]
73          vmovsd    %xmm1, 8(%rsp)                                #[spill]
74          jmp       ..B17.12      # Prob 100%                     #
75                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
76  ..B17.11:                         # Preds ..B17.4
77                                    # Execution count [4.05e-01]
78          movq      %rax, %rcx                                    #21.25
79          andq      $15, %rcx                                     #21.25
80          negq      %rcx                                          #21.25
81          addq      %rax, %rcx                                    #21.25
82                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
83  ..B17.12:                         # Preds ..B17.11 ..B17.10 ..B17.7 ..B17.28
84                                    # Execution count [9.00e-01]
85          vmovsd    8(%rsp), %xmm1                                #20.16[spill]
86          vxorpd    %ymm3, %ymm3, %ymm3                           #20.16
87          vxorpd    %xmm0, %xmm0, %xmm0                           #20.16
88          vmovdqa   %ymm3, %ymm2                                  #20.16
89          vmovsd    %xmm1, %xmm0, %xmm4                           #20.16
90          vbroadcastsd (%rsp), %ymm0                              #17.8[spill]
91          vmovdqa   %ymm2, %ymm1                                  #20.16
92          vmovaps   %xmm4, %xmm4                                  #20.16
93                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                      ↪  ymm1 ymm2 ymm3 ymm4
94  ..B17.13:                         # Preds ..B17.13 ..B17.12
95                                    # Execution count [5.00e+00]
96          vfmadd231pd (%rdx,%rdi,8), %ymm0, %ymm4                 #23.9
97          vfmadd231pd 32(%rdx,%rdi,8), %ymm0, %ymm3               #23.9
98          vfmadd231pd 64(%rdx,%rdi,8), %ymm0, %ymm2               #23.9
99          vfmadd231pd 96(%rdx,%rdi,8), %ymm0, %ymm1               #23.9
100         addq      $16, %rdi                                     #21.25
```

```
101         cmpq      %rcx, %rdi                                    #21.25
102         jb        ..B17.13      # Prob 82%                      #21.25
103                                   # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                      ↪  ymm1 ymm2 ymm3 ymm4
104 ..B17.14:                        # Preds ..B17.13
105                                   # Execution count [9.00e-01]
106         vaddpd    %ymm3, %ymm4, %ymm0                           #20.16
107         vaddpd    %ymm1, %ymm2, %ymm1                           #20.16
108         vaddpd    %ymm1, %ymm0, %ymm2                           #20.16
109         vextractf128 $1, %ymm2, %xmm3                           #20.16
110         vaddpd    %xmm3, %xmm2, %xmm4                           #20.16
111         vunpckhpd %xmm4, %xmm4, %xmm5                           #20.16
112         vaddsd    %xmm5, %xmm4, %xmm6                           #20.16
113         vmovsd    %xmm6, 8(%rsp)                                #20.16[spill]
114                                   # LOE rax rdx rcx rbx rbp r13 r14 r15
115 ..B17.15:                        # Preds ..B17.14 ..B17.25
116                                   # Execution count [1.00e+00]
117         lea       1(%rcx), %rsi                                 #21.25
118         cmpq      %rax, %rsi                                    #21.25
119         ja        ..B17.24      # Prob 50%                      #21.25
120                                   # LOE rax rdx rcx rbx rbp r13 r14 r15
121 ..B17.16:                        # Preds ..B17.15
122                                   # Execution count [9.00e-01]
123         subq      %rcx, %rax                                    #21.25
124         cmpq      $4, %rax                                      #21.25
125         jb        ..B17.27      # Prob 10%                      #21.25
126                                   # LOE rax rdx rcx rbx rbp r13 r14 r15
127 ..B17.17:                        # Preds ..B17.16
128                                   # Execution count [9.00e-01]
129         vmovsd    8(%rsp), %xmm1                                #20.16[spill]
130         movq      %rax, %rsi                                    #21.25
131         vxorpd    %xmm0, %xmm0, %xmm0                           #20.16
132         lea       (%rdx,%rcx,8), %rdi                           #21.25
133         vmovsd    %xmm1, %xmm0, %xmm1                           #20.16
134         andq      $-4, %rsi                                     #21.25
135         vbroadcastsd (%rsp), %ymm0                              #17.8[spill]
136         xorl      %r8d, %r8d                                    #21.25
137         vmovaps   %xmm1, %xmm1                                  #20.16
138                                   # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                      ↪  r15 ymm0 ymm1
139 ..B17.18:                        # Preds ..B17.18 ..B17.17
140                                   # Execution count [5.00e+00]
141         vmulpd    (%rdi,%r8,8), %ymm0, %ymm2                    #23.9
142         addq      $4, %r8                                       #21.25
143         vaddpd    %ymm1, %ymm2, %ymm1                           #23.9
144         cmpq      %rsi, %r8                                     #21.25
145         jb        ..B17.18      # Prob 82%                      #21.25
146                                   # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                      ↪  r15 ymm0 ymm1
147 ..B17.19:                        # Preds ..B17.18
148                                   # Execution count [9.00e-01]
149         vextractf128 $1, %ymm1, %xmm0                           #20.16
150         vaddpd    %xmm0, %xmm1, %xmm2                           #20.16
```

```
        vunpckhpd %xmm2, %xmm2, %xmm3                           #20.16
        vaddsd    %xmm3, %xmm2, %xmm4                           #20.16
        vmovsd    %xmm4, 8(%rsp)                                #20.16[spill]
                                # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
..B17.20:                       # Preds ..B17.19 ..B17.27
                                # Execution count [1.00e+00]
        cmpq      %rax, %rsi                                    #21.25
        jae       ..B17.24      # Prob 9%                       #21.25
                                # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
..B17.21:                       # Preds ..B17.20
                                # Execution count [9.00e-01]
        vmovsd    8(%rsp), %xmm1                                #21.25[spill]
        lea       (%rdx,%rcx,8), %rdx                           #21.25
        vmovsd    (%rsp), %xmm2                                 #21.25[spill]
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B17.22:                       # Preds ..B17.22 ..B17.21
                                # Execution count [5.00e+00]
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #23.9
        incq      %rsi                                          #21.25
        vaddsd    %xmm1, %xmm0, %xmm1                           #23.9
        cmpq      %rax, %rsi                                    #21.25
        jb        ..B17.22      # Prob 82%                      #21.25
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B17.23:                       # Preds ..B17.22
                                # Execution count [9.00e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
                                # LOE rbx rbp r13 r14 r15
..B17.24:                       # Preds ..B17.23 ..B17.20 ..B17.15 ..B17.32
                                # Execution count [1.00e+00]
# Begin ASM
# # integrate_dummy exit
# End ASM
```

Векторизованное интегрирование

```
# Begin ASM
# # integrate_omp_simd enter
# End ASM
                                # LOE rbx rbp r12 r13 r14 r15
..B19.33:                       # Preds ..B19.34
                                # Execution count [1.00e+00]
        movq      (%r12), %rdx                                  #47.24
        movq      8(%r12), %rax                                 #47.24
        vxorpd    %xmm0, %xmm0, %xmm0                           #45.16
        subq      %rdx, %rax                                    #47.24
        vmovsd    %xmm0, 8(%rsp)                                #45.16[spill]
        sarq      $3, %rax                                      #47.24
        je        ..B19.25      # Prob 50%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.2:                        # Preds ..B19.33
                                # Execution count [5.00e-01]
        je        ..B19.25      # Prob 50%                      #47.5
```

```
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.3:                        # Preds ..B19.2
                                # Execution count [4.50e-01]
        cmpq      $16, %rax                                     #47.5
        jb        ..B19.26      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.4:                        # Preds ..B19.3
                                # Execution count [4.50e-01]
        cmpq      $29, %rax                                     #47.5
        jb        ..B19.29      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.5:                        # Preds ..B19.4
                                # Execution count [4.50e-01]
        movq      %rdx, %rdi                                    #47.5
        andq      $31, %rdi                                     #47.5
        je        ..B19.12      # Prob 50%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.6:                        # Preds ..B19.5
                                # Execution count [4.50e-01]
        testq     $7, %rdi                                      #47.5
        jne       ..B19.26      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.7:                        # Preds ..B19.6
                                # Execution count [2.25e-01]
        negq      %rdi                                          #47.5
        addq      $32, %rdi                                     #47.5
        shrq      $3, %rdi                                      #47.5
        lea       16(%rdi), %rcx                                #47.5
        cmpq      %rcx, %rax                                    #47.5
        jb        ..B19.26      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.8:                        # Preds ..B19.7
                                # Execution count [4.50e-01]
        movq      %rax, %rcx                                    #47.5
        xorl      %esi, %esi                                    #47.5
        subq      %rdi, %rcx                                    #47.5
        andq      $15, %rcx                                     #47.5
        negq      %rcx                                          #47.5
        addq      %rax, %rcx                                    #47.5
        testq     %rdi, %rdi                                    #47.5
        je        ..B19.13      # Prob 9%                       #47.5
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm0
..B19.9:                        # Preds ..B19.8
                                # Execution count [4.50e-01]
        vmovapd   %xmm0, %xmm1                                  #
        vmovsd    (%rsp), %xmm2                                 #[spill]
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm1 xmm2
..B19.10:                       # Preds ..B19.10 ..B19.9
                                # Execution count [2.50e+00]
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #49.9
        incq      %rsi                                          #47.5
```

```
69          vaddsd    %xmm1, %xmm0, %xmm1                             #49.9
70          cmpq      %rdi, %rsi                                      #47.5
71          jb        ..B19.10      # Prob 82%                        #47.5
72                                    # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                      ↪  xmm1 xmm2
73  ..B19.11:                         # Preds ..B19.10
74                                    # Execution count [4.50e-01]
75          vmovsd    %xmm1, 8(%rsp)                                  #[spill]
76          jmp       ..B19.13      # Prob 100%                       #
77                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
78  ..B19.12:                         # Preds ..B19.5
79                                    # Execution count [2.03e-01]
80          movq      %rax, %rcx                                      #47.5
81          andq      $15, %rcx                                       #47.5
82          negq      %rcx                                            #47.5
83          addq      %rax, %rcx                                      #47.5
84                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
85  ..B19.13:                         # Preds ..B19.12 ..B19.11 ..B19.8 ..B19.29
86                                    # Execution count [4.50e-01]
87          vmovsd    8(%rsp), %xmm1                                  #45.16[spill]
88          vxorpd    %ymm3, %ymm3, %ymm3                             #45.16
89          vxorpd    %xmm0, %xmm0, %xmm0                             #45.16
90          vmovdqa   %ymm3, %ymm2                                    #45.16
91          vmovsd    %xmm1, %xmm0, %xmm4                             #45.16
92          vbroadcastsd (%rsp), %ymm0                                #42.8[spill]
93          vmovdqa   %ymm2, %ymm1                                    #45.16
94          vmovaps   %xmm4, %xmm4                                    #45.16
95                                    # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                      ↪  ymm1 ymm2 ymm3 ymm4
96  ..B19.14:                         # Preds ..B19.14 ..B19.13
97                                    # Execution count [2.50e+00]
98          vfmadd231pd (%rdx,%rdi,8), %ymm0, %ymm4                   #49.9
99          vfmadd231pd 32(%rdx,%rdi,8), %ymm0, %ymm3                 #49.9
100         vfmadd231pd 64(%rdx,%rdi,8), %ymm0, %ymm2                 #49.9
101         vfmadd231pd 96(%rdx,%rdi,8), %ymm0, %ymm1                 #49.9
102         addq      $16, %rdi                                       #47.5
103         cmpq      %rcx, %rdi                                      #47.5
104         jb        ..B19.14      # Prob 82%                        #47.5
105                                   # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                      ↪  ymm1 ymm2 ymm3 ymm4
106 ..B19.15:                         # Preds ..B19.14
107                                   # Execution count [4.50e-01]
108         vaddpd    %ymm3, %ymm4, %ymm0                             #45.16
109         vaddpd    %ymm1, %ymm2, %ymm1                             #45.16
110         vaddpd    %ymm1, %ymm0, %ymm2                             #45.16
111         vextractf128 $1, %ymm2, %xmm3                             #45.16
112         vaddpd    %xmm3, %xmm2, %xmm4                             #45.16
113         vunpckhpd %xmm4, %xmm4, %xmm5                             #45.16
114         vaddsd    %xmm5, %xmm4, %xmm6                             #45.16
115         vmovsd    %xmm6, 8(%rsp)                                  #45.16[spill]
116                                   # LOE rax rdx rcx rbx rbp r13 r14 r15
117 ..B19.16:                         # Preds ..B19.15 ..B19.26
118                                   # Execution count [5.00e-01]
```

```
119         lea       1(%rcx), %rsi                                 #47.5
120         cmpq      %rax, %rsi                                    #47.5
121         ja        ..B19.25      # Prob 50%                      #47.5
122                                 # LOE rax rdx rcx rbx rbp r13 r14 r15
123 ..B19.17:                       # Preds ..B19.16
124                                 # Execution count [4.50e-01]
125         subq      %rcx, %rax                                    #47.5
126         cmpq      $4, %rax                                      #47.5
127         jb        ..B19.28      # Prob 10%                      #47.5
128                                 # LOE rax rdx rcx rbx rbp r13 r14 r15
129 ..B19.18:                       # Preds ..B19.17
130                                 # Execution count [4.50e-01]
131         vmovsd    8(%rsp), %xmm1                                #45.16[spill]
132         movq      %rax, %rsi                                    #47.5
133         vxorpd    %xmm0, %xmm0, %xmm0                           #45.16
134         lea       (%rdx,%rcx,8), %rdi                           #49.26
135         vmovsd    %xmm1, %xmm0, %xmm1                           #45.16
136         andq      $-4, %rsi                                     #47.5
137         vbroadcastsd (%rsp), %ymm0                              #42.8[spill]
138         xorl      %r8d, %r8d                                    #47.5
139         vmovaps   %xmm1, %xmm1                                  #45.16
140                                 # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                    ↪  r15 ymm0 ymm1
141 ..B19.19:                       # Preds ..B19.19 ..B19.18
142                                 # Execution count [2.50e+00]
143         vmulpd    (%rdi,%r8,8), %ymm0, %ymm2                    #49.9
144         addq      $4, %r8                                       #47.5
145         vaddpd    %ymm1, %ymm2, %ymm1                           #49.9
146         cmpq      %rsi, %r8                                     #47.5
147         jb        ..B19.19      # Prob 82%                      #47.5
148                                 # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                    ↪  r15 ymm0 ymm1
149 ..B19.20:                       # Preds ..B19.19
150                                 # Execution count [4.50e-01]
151         vextractf128 $1, %ymm1, %xmm0                           #45.16
152         vaddpd    %xmm0, %xmm1, %xmm2                           #45.16
153         vunpckhpd %xmm2, %xmm2, %xmm3                           #45.16
154         vaddsd    %xmm3, %xmm2, %xmm4                           #45.16
155         vmovsd    %xmm4, 8(%rsp)                                #45.16[spill]
156                                 # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
157 ..B19.21:                       # Preds ..B19.20 ..B19.28
158                                 # Execution count [4.50e-01]
159         cmpq      %rax, %rsi                                    #47.5
160         jae       ..B19.25      # Prob 9%                       #47.5
161                                 # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
162 ..B19.22:                       # Preds ..B19.21
163                                 # Execution count [4.50e-01]
164         vmovsd    8(%rsp), %xmm1                                #49.26[spill]
165         lea       (%rdx,%rcx,8), %rdx                           #49.26
166         vmovsd    (%rsp), %xmm2                                 #49.26[spill]
167                                 # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
168 ..B19.23:                       # Preds ..B19.23 ..B19.22
169                                 # Execution count [2.50e+00]
```

```
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #49.9
        incq      %rsi                                          #47.5
        vaddsd    %xmm1, %xmm0, %xmm1                           #49.9
        cmpq      %rax, %rsi                                    #47.5
        jb        ..B19.23      # Prob 82%                      #47.5
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B19.24:                       # Preds ..B19.23
                                # Execution count [4.50e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
                                # LOE rbx rbp r13 r14 r15
..B19.25:                       # Preds ..B19.24 ..B19.21 ..B19.16 ..B19.2
    ↪  ..B19.33
                                #
                                # Execution count [1.00e+00]
# Begin ASM
# # integrate_omp_simd exit
# End ASM
```

Параллельное интегрирование

```
# Begin ASM
# # integrate_omp_parallel enter
# End ASM
                                # LOE
..B18.35:                       # Preds ..B18.36
                                # Execution count [1.00e+00]
        movl      $.2.486_2_kmpc_loc_struct_pack.18, %edi       #33.5
        movq      $0, 72(%rsp)                                  #32.16
        call      __kmpc_global_thread_num                      #33.5
                                # LOE eax
..B18.34:                       # Preds ..B18.35
                                # Execution count [1.00e+00]
        movl      %eax, 80(%rsp)                                #33.5
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
        xorl      %eax, %eax                                    #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.385:
        call      __kmpc_ok_to_fork                             #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.386:
                                # LOE eax
..B18.2:                        # Preds ..B18.34
                                # Execution count [1.00e+00]
        testl     %eax, %eax                                    #33.5
        je        ..B18.4       # Prob 50%                      #33.5
                                # LOE
..B18.3:                        # Preds ..B18.2
                                # Execution count [0.00e+00]
        addq      $-16, %rsp                                    #33.5
        .cfi_def_cfa_offset 128
        movl
        ↪  $L__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_33__par_region0_2.40,
        ↪  %edx #33.5
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
```

```
31          lea       16(%rsp), %rax                                #33.5
32          movl      $4, %esi                                      #33.5
33          lea       72(%rax), %rcx                                #33.5
34          lea       56(%rax), %r8                                 #33.5
35          movq      %rax, (%rsp)                                  #33.5
36          lea       64(%rax), %r9                                 #33.5
37          xorl      %eax, %eax                                    #33.5
38  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.388:
39          call      __kmpc_fork_call                              #33.5
40  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.389:
41                                # LOE
42  ..B18.37:                     # Preds ..B18.3
43                                # Execution count [0.00e+00]
44          addq      $16, %rsp                                     #33.5
45          .cfi_def_cfa_offset 112
46          jmp       ..B18.7       # Prob 100%                     #33.5
47                                # LOE
48  ..B18.4:                      # Preds ..B18.2
49                                # Execution count [0.00e+00]
50          movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
51          xorl      %eax, %eax                                    #33.5
52          movl      80(%rsp), %esi                                #33.5
53  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.391:
54          call      __kmpc_serialized_parallel                    #33.5
55  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.392:
56                                # LOE
57  ..B18.5:                      # Preds ..B18.4
58                                # Execution count [0.00e+00]
59          movl
            ↪  $___kmpv_zero_Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_0, %esi
            ↪  #33.5
60          lea       80(%rsp), %rdi                                #33.5
61          lea       -8(%rdi), %rdx                                #33.5
62          lea       -16(%rdx), %rcx                               #33.5
63          lea       -8(%rdx), %r8                                 #33.5
64          lea       (%rsp), %r9                                   #33.5
65  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.393:
66          call
            ↪  L__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_33__par_region0_2.40
            ↪  #33.5
67  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.394:
68                                # LOE
69  ..B18.6:                      # Preds ..B18.5
70                                # Execution count [0.00e+00]
71          movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
72          xorl      %eax, %eax                                    #33.5
73          movl      80(%rsp), %esi                                #33.5
74  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.395:
75          call      __kmpc_end_serialized_parallel                #33.5
76  ..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.396:
77                                # LOE
78  ..B18.7:                      # Preds ..B18.37 ..B18.6
79                                # Execution count [1.00e+00]
```

```
# Begin ASM
# # integrate_omp_parallel exit
# End ASM
```

4.2.2 Intel LLVM-based compiler

Скалярное деление (независимые операции)

```
        #APP
        # independent_scalar_operation enter
        #NO_APP
        callq        _ZNSt6chrono3_V212system_clock3nowEv
        movq        %rax, %r14
        #APP
        mfence
        rdtsc
        shlq         $32, %rdx
        orq         %rdx, %rax
        #NO_APP
        movq         %rax, %r15
        movq         %rbx, %rax
        orq         %r12, %rax
        shrq         $32, %rax
        je          .LBB0_1
# %bb.2:
        movq         %rbx, %rax
        cqto
        idivq         %r12
        jmp         .LBB0_3
.LBB0_1:
        movl         %ebx, %eax
        xorl         %edx, %edx
        divl         %r12d
                                        # kill: def $eax killed $eax def $rax
.LBB0_3:
        movabsq         $100000000000, %rcx             # imm = 0x174876E800
        .p2align         4, 0x90
.LBB0_4:                                # =>This Inner Loop Header: Depth=1
        movq         %rax, (%rsp)
        #APP
        #NO_APP
        decq         %rcx
        jne         .LBB0_4
# %bb.5:
        #APP
        mfence
        rdtsc
        shlq         $32, %rdx
        orq         %rdx, %rax
        #NO_APP
        movq         %rax, %rbx
```

```
        callq   _ZNSt6chrono3_V212system_clock3nowEv
        vmovq   %rbx, %xmm0
        vmovq   %rax, %xmm1
        vpunpcklqdq     %xmm1, %xmm0, %xmm0     # xmm0 = xmm0[0],xmm1[0]
        vmovq   %r14, %xmm1
        vmovq   %r15, %xmm2
        vpunpcklqdq     %xmm1, %xmm2, %xmm1     # xmm1 = xmm2[0],xmm1[0]
        vpsubq   %xmm1, %xmm0, %xmm0
        vpxor   %xmm1, %xmm1, %xmm1
        vpblendw    $3, %xmm0, %xmm1, %xmm1         # xmm1 =
        ↪  xmm0[0,1],xmm1[2,3,4,5,6,7]
        vpor    .LCPI0_0(%rip), %xmm1, %xmm1
        vpsrlq   $32, %xmm0, %xmm2
        vpor    .LCPI0_1(%rip), %xmm2, %xmm2
        vsubpd   .LCPI0_2(%rip), %xmm2, %xmm2
        vaddpd   %xmm2, %xmm1, %xmm1
        vpextrq   $1, %xmm0, %rax
        vcvtsi2sd    %rax, %xmm3, %xmm0
        vmovddup    %xmm0, %xmm0                    # xmm0 = xmm0[0,0]
        vblendpd    $1, %xmm1, %xmm0, %xmm0         # xmm0 =
        ↪  xmm1[0],xmm0[1]
        vdivpd   .LCPI0_3(%rip), %xmm0, %xmm0
        #APP
        # independent_scalar_operation exit
        #NO_APP
```

Скалярное деление (зависимые операции)

```
        #APP
        # dependent_scalar_operation enter
        #NO_APP
        callq   _ZNSt6chrono3_V212system_clock3nowEv
        movq    %rax, %r14
        movabsq    $-100000000000, %rsi            # imm =
        ↪  0xFFFFFFE8B7891800
        #APP
        mfence
        rdtsc
        shlq    $32, %rdx
        orq    %rdx, %rax
        #NO_APP
        movq    %rax, %r15
        jmp    .LBB1_1
        .p2align    4, 0x90
.LBB1_24:                               #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq   %rbx
        movq    %rax, %r12
        addq    $8, %rsi
        je    .LBB1_26
.LBB1_1:                                # =>This Inner Loop Header: Depth=1
        movq    %r12, %rax
```

```
        orq         %rbx, %rax
        shrq         $32, %rax
        je          .LBB1_2
# %bb.3:                                #   in Loop: Header=BB1_1 Depth=1
        movq         %r12, %rax
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_5
.LBB1_6:                                #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_8
.LBB1_9:                                #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_11
.LBB1_12:                               #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_14
.LBB1_15:                               #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_17
.LBB1_18:                               #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        je          .LBB1_20
.LBB1_21:                               #   in Loop: Header=BB1_1 Depth=1
        cqto
        idivq         %rbx
        movq         %rax, %rcx
        orq         %rbx, %rcx
        shrq         $32, %rcx
        jne         .LBB1_24
```

```
77          jmp         .LBB1_23
78          .p2align        4, 0x90
79  .LBB1_2:                                #   in Loop: Header=BB1_1 Depth=1
80          movl        %r12d, %eax
81          xorl        %edx, %edx
82          divl        %ebx
83                                          # kill: def $eax killed $eax def $rax
84          movq        %rax, %rcx
85          orq         %rbx, %rcx
86          shrq        $32, %rcx
87          jne         .LBB1_6
88  .LBB1_5:                                #   in Loop: Header=BB1_1 Depth=1
89                                          # kill: def $eax killed $eax killed
                                            ↪ $rax
90          xorl        %edx, %edx
91          divl        %ebx
92                                          # kill: def $eax killed $eax def $rax
93          movq        %rax, %rcx
94          orq         %rbx, %rcx
95          shrq        $32, %rcx
96          jne         .LBB1_9
97  .LBB1_8:                                #   in Loop: Header=BB1_1 Depth=1
98                                          # kill: def $eax killed $eax killed
                                            ↪ $rax
99          xorl        %edx, %edx
100         divl        %ebx
101                                         # kill: def $eax killed $eax def $rax
102         movq        %rax, %rcx
103         orq         %rbx, %rcx
104         shrq        $32, %rcx
105         jne         .LBB1_12
106 .LBB1_11:                               #   in Loop: Header=BB1_1 Depth=1
107                                         # kill: def $eax killed $eax killed
                                            ↪ $rax
108         xorl        %edx, %edx
109         divl        %ebx
110                                         # kill: def $eax killed $eax def $rax
111         movq        %rax, %rcx
112         orq         %rbx, %rcx
113         shrq        $32, %rcx
114         jne         .LBB1_15
115 .LBB1_14:                               #   in Loop: Header=BB1_1 Depth=1
116                                         # kill: def $eax killed $eax killed
                                            ↪ $rax
117         xorl        %edx, %edx
118         divl        %ebx
119                                         # kill: def $eax killed $eax def $rax
120         movq        %rax, %rcx
121         orq         %rbx, %rcx
122         shrq        $32, %rcx
123         jne         .LBB1_18
124 .LBB1_17:                               #   in Loop: Header=BB1_1 Depth=1
```

```
                                        # kill: def $eax killed $eax killed
                                        ↪ $rax
        xorl    %edx, %edx
        divl    %ebx
                                        # kill: def $eax killed $eax def $rax
        movq    %rax, %rcx
        orq     %rbx, %rcx
        shrq    $32, %rcx
        jne     .LBB1_21
.LBB1_20:                               #   in Loop: Header=BB1_1 Depth=1
                                        # kill: def $eax killed $eax killed
                                        ↪ $rax
        xorl    %edx, %edx
        divl    %ebx
                                        # kill: def $eax killed $eax def $rax
        movq    %rax, %rcx
        orq     %rbx, %rcx
        shrq    $32, %rcx
        jne     .LBB1_24
.LBB1_23:                               #   in Loop: Header=BB1_1 Depth=1
                                        # kill: def $eax killed $eax killed
                                        ↪ $rax
        xorl    %edx, %edx
        divl    %ebx
        movl    %eax, %r12d
        addq    $8, %rsi
        jne     .LBB1_1
.LBB1_26:
        movq    %r12, (%rsp)
        #APP
        #NO_APP
        #APP
        mfence
        rdtsc
        shlq    $32, %rdx
        orq     %rdx, %rax
        #NO_APP
        movq    %rax, %rbx
        callq   _ZNSt6chrono3_V212system_clock3nowEv
        vmovq   %rbx, %xmm0
        vmovq   %rax, %xmm1
        vpunpcklqdq     %xmm1, %xmm0, %xmm0     # xmm0 = xmm0[0],xmm1[0]
        vmovq   %r14, %xmm1
        vmovq   %r15, %xmm2
        vpunpcklqdq     %xmm1, %xmm2, %xmm1     # xmm1 = xmm2[0],xmm1[0]
        vpsubq  %xmm1, %xmm0, %xmm0
        vpxor   %xmm1, %xmm1, %xmm1
        vpblendw        $3, %xmm0, %xmm1, %xmm1         # xmm1 =
        ↪ xmm0[0,1],xmm1[2,3,4,5,6,7]
        vpor    .LCPI1_0(%rip), %xmm1, %xmm1
        vpsrlq  $32, %xmm0, %xmm2
        vpor    .LCPI1_1(%rip), %xmm2, %xmm2
        vsubpd  .LCPI1_2(%rip), %xmm2, %xmm2
```

```
        vaddpd        %xmm2, %xmm1, %xmm1
        vpextrq        $1, %xmm0, %rax
        vcvtsi2sd        %rax, %xmm3, %xmm0
        vmovddup        %xmm0, %xmm0                    # xmm0 = xmm0[0,0]
        vblendpd        $1, %xmm1, %xmm0, %xmm0         # xmm0 =
        ↪  xmm1[0],xmm0[1]
        vdivpd        .LCPI1_3(%rip), %xmm0, %xmm0
        #APP
        # dependent_scalar_operation exit
        #NO_APP
```

Векторное деление (независимые операции)

```
        #APP
        # independent_vector_operation enter
        #NO_APP
        vmovq        %rdi, %xmm0
        vpbroadcastq        %xmm0, %ymm0
        vmovdqa        %ymm0, 64(%rsp)                 # 32-byte Spill
        vmovq        %rsi, %xmm0
        vpbroadcastq        %xmm0, %ymm0
        vmovdqa        %ymm0, 32(%rsp)                 # 32-byte Spill
        vzeroupper
        callq        _ZNSt6chrono3_V212system_clock3nowEv
        movq        %rax, %r14
        movabsq        $25000000000, %rsi              # imm = 0x5D21DBA00
        #APP
        mfence
        rdtsc
        shlq        $32, %rdx
        orq        %rdx, %rax
        #NO_APP
        movq        %rax, %r15
        movq        __svml_i64div4@GOTPCREL(%rip), %rdi
        .p2align        4, 0x90
.LBB3_1:                                # =>This Inner Loop Header: Depth=1
        vmovdqa        64(%rsp), %ymm0                 # 32-byte Reload
        vmovaps        32(%rsp), %ymm1                 # 32-byte Reload
        callq        *%rdi
        vmovdqa        %ymm0, (%rsp)
        #APP
        #NO_APP
        decq        %rsi
        jne        .LBB3_1
# %bb.2:
        #APP
        mfence
        rdtsc
        shlq        $32, %rdx
        orq        %rdx, %rax
        #NO_APP
        movq        %rax, %rbx
```

```
        vzeroupper
        callq   _ZNSt6chrono3_V212system_clock3nowEv
        vmovq   %rbx, %xmm0
        vmovq   %rax, %xmm1
        vpunpcklqdq     %xmm1, %xmm0, %xmm0     # xmm0 = xmm0[0],xmm1[0]
        vmovq   %r14, %xmm1
        vmovq   %r15, %xmm2
        vpunpcklqdq     %xmm1, %xmm2, %xmm1     # xmm1 = xmm2[0],xmm1[0]
        vpsubq  %xmm1, %xmm0, %xmm0
        vpxor   %xmm1, %xmm1, %xmm1
        vpblendw        $3, %xmm0, %xmm1, %xmm1         # xmm1 =
        ↪ xmm0[0,1],xmm1[2,3,4,5,6,7]
        vpor    .LCPI3_0(%rip), %xmm1, %xmm1
        vpsrlq  $32, %xmm0, %xmm2
        vpor    .LCPI3_1(%rip), %xmm2, %xmm2
        vsubpd  .LCPI3_2(%rip), %xmm2, %xmm2
        vaddpd  %xmm2, %xmm1, %xmm1
        vpextrq $1, %xmm0, %rax
        vcvtsi2sd       %rax, %xmm3, %xmm0
        vmovddup        %xmm0, %xmm0                    # xmm0 = xmm0[0,0]
        vblendpd        $1, %xmm1, %xmm0, %xmm0         # xmm0 =
        ↪ xmm1[0],xmm0[1]
        vdivpd  .LCPI3_3(%rip), %xmm0, %xmm0
        #APP
        # independent_vector_operation exit
        #NO_APP
```

Векторное деление (зависимые операции)

```
        #APP
        # dependent_vector_operation enter
        #NO_APP
        vmovq   %rdi, %xmm0
        vpbroadcastq    %xmm0, %ymm0
        vmovdqa %ymm0, 32(%rsp)                 # 32-byte Spill
        vmovdqa %ymm0, (%rsp)
        vmovq   %rsi, %xmm0
        vpbroadcastq    %xmm0, %ymm0
        vmovdqa %ymm0, 64(%rsp)                 # 32-byte Spill
        vzeroupper
        callq   _ZNSt6chrono3_V212system_clock3nowEv
        vmovaps 32(%rsp), %ymm0                 # 32-byte Reload
        movq    %rax, %r14
        movabsq $25000000000, %rsi              # imm = 0x5D21DBA00
        #APP
        mfence
        rdtsc
        shlq    $32, %rdx
        orq     %rdx, %rax
        #NO_APP
        movq    %rax, %r15
        movq    __svml_i64div4@GOTPCREL(%rip), %rdi
```

```
        .p2align        4, 0x90
.LBB4_1:                                # =>This Inner Loop Header: Depth=1
        vmovaps         64(%rsp), %ymm1                 # 32-byte Reload
        callq           *%rdi
        decq            %rsi
        jne             .LBB4_1
# %bb.2:
        vmovaps         %ymm0, (%rsp)
        #APP
        #NO_APP
        #APP
        mfence
        rdtsc
        shlq            $32, %rdx
        orq             %rdx, %rax
        #NO_APP
        movq            %rax, %rbx
        vzeroupper
        callq           _ZNSt6chrono3_V212system_clock3nowEv
        vmovq           %rbx, %xmm0
        vmovq           %rax, %xmm1
        vpunpcklqdq             %xmm1, %xmm0, %xmm0     # xmm0 = xmm0[0],xmm1[0]
        vmovq           %r14, %xmm1
        vmovq           %r15, %xmm2
        vpunpcklqdq             %xmm1, %xmm2, %xmm1     # xmm1 = xmm2[0],xmm1[0]
        vpsubq          %xmm1, %xmm0, %xmm0
        vpxor           %xmm1, %xmm1, %xmm1
        vpblendw        $3, %xmm0, %xmm1, %xmm1         # xmm1 =
        ↪  xmm0[0,1],xmm1[2,3,4,5,6,7]
        vpor            .LCPI4_0(%rip), %xmm1, %xmm1
        vpsrlq          $32, %xmm0, %xmm2
        vpor            .LCPI4_1(%rip), %xmm2, %xmm2
        vsubpd          .LCPI4_2(%rip), %xmm2, %xmm2
        vaddpd          %xmm2, %xmm1, %xmm1
        vpextrq         $1, %xmm0, %rax
        vcvtsi2sd       %rax, %xmm3, %xmm0
        vmovddup        %xmm0, %xmm0                    # xmm0 = xmm0[0,0]
        vblendpd        $1, %xmm1, %xmm0, %xmm0         # xmm0 =
        ↪  xmm1[0],xmm0[1]
        vdivpd          .LCPI4_3(%rip), %xmm0, %xmm0
        #APP
        # dependent_vector_operation exit
        #NO_APP
```

4.2.3 GNU GCC

«Наивное» интегрирование

```
# Begin ASM
# # integrate_dummy enter
# End ASM
```

```
                                # LOE rbx rbp r12 r13 r14 r15
..B17.32:                       # Preds ..B17.33
                                # Execution count [1.00e+00]
        movq      (%r12), %rdx                                  #21.25
        movq      8(%r12), %rax                                 #21.25
        vxorpd    %xmm0, %xmm0, %xmm0                           #20.16
        vmovsd    %xmm0, 8(%rsp)                                #20.16[spill]
        cmpq      %rax, %rdx                                    #21.25
        je        ..B17.24      # Prob 50%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.2:                        # Preds ..B17.32
                                # Execution count [9.00e-01]
        subq      %rdx, %rax                                    #21.25
        addq      $7, %rax                                      #21.25
        shrq      $3, %rax                                      #21.25
        cmpq      $16, %rax                                     #21.25
        jb        ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.3:                        # Preds ..B17.2
                                # Execution count [9.00e-01]
        cmpq      $29, %rax                                     #21.25
        jb        ..B17.28      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B17.4:                        # Preds ..B17.3
                                # Execution count [9.00e-01]
        movq      %rdx, %rdi                                    #21.25
        andq      $31, %rdi                                     #21.25
        je        ..B17.11      # Prob 50%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.5:                        # Preds ..B17.4
                                # Execution count [9.00e-01]
        testq     $7, %rdi                                      #21.25
        jne       ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.6:                        # Preds ..B17.5
                                # Execution count [4.50e-01]
        negq      %rdi                                          #21.25
        addq      $32, %rdi                                     #21.25
        shrq      $3, %rdi                                      #21.25
        lea       16(%rdi), %rcx                                #21.25
        cmpq      %rcx, %rax                                    #21.25
        jb        ..B17.25      # Prob 10%                      #21.25
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B17.7:                        # Preds ..B17.6
                                # Execution count [9.00e-01]
        movq      %rax, %rcx                                    #21.25
        xorl      %esi, %esi                                    #21.25
        subq      %rdi, %rcx                                    #21.25
        andq      $15, %rcx                                     #21.25
        negq      %rcx                                          #21.25
        addq      %rax, %rcx                                    #21.25
        testq     %rdi, %rdi                                    #21.25
        je        ..B17.12      # Prob 9%                       #21.25
```

```
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm0
..B17.8:                        # Preds ..B17.7
                                # Execution count [9.00e-01]
        vmovapd   %xmm0, %xmm1                                  #
        vmovsd    (%rsp), %xmm2                                 #[spill]
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm1 xmm2
..B17.9:                        # Preds ..B17.9 ..B17.8
                                # Execution count [5.00e+00]
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #23.9
        incq      %rsi                                          #21.25
        vaddsd    %xmm1, %xmm0, %xmm1                           #23.9
        cmpq      %rdi, %rsi                                    #21.25
        jb        ..B17.9       # Prob 82%                      #21.25
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm1 xmm2
..B17.10:                       # Preds ..B17.9
                                # Execution count [9.00e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
        jmp       ..B17.12      # Prob 100%                     #
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
..B17.11:                       # Preds ..B17.4
                                # Execution count [4.05e-01]
        movq      %rax, %rcx                                    #21.25
        andq      $15, %rcx                                     #21.25
        negq      %rcx                                          #21.25
        addq      %rax, %rcx                                    #21.25
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
..B17.12:                       # Preds ..B17.11 ..B17.10 ..B17.7 ..B17.28
                                # Execution count [9.00e-01]
        vmovsd    8(%rsp), %xmm1                                #20.16[spill]
        vxorpd    %ymm3, %ymm3, %ymm3                           #20.16
        vxorpd    %xmm0, %xmm0, %xmm0                           #20.16
        vmovdqa   %ymm3, %ymm2                                  #20.16
        vmovsd    %xmm1, %xmm0, %xmm4                           #20.16
        vbroadcastsd (%rsp), %ymm0                              #17.8[spill]
        vmovdqa   %ymm2, %ymm1                                  #20.16
        vmovaps   %xmm4, %xmm4                                  #20.16
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                ↪  ymm1 ymm2 ymm3 ymm4
..B17.13:                       # Preds ..B17.13 ..B17.12
                                # Execution count [5.00e+00]
        vfmadd231pd (%rdx,%rdi,8), %ymm0, %ymm4                 #23.9
        vfmadd231pd 32(%rdx,%rdi,8), %ymm0, %ymm3               #23.9
        vfmadd231pd 64(%rdx,%rdi,8), %ymm0, %ymm2               #23.9
        vfmadd231pd 96(%rdx,%rdi,8), %ymm0, %ymm1               #23.9
        addq      $16, %rdi                                     #21.25
        cmpq      %rcx, %rdi                                    #21.25
        jb        ..B17.13      # Prob 82%                      #21.25
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                ↪  ymm1 ymm2 ymm3 ymm4
..B17.14:                       # Preds ..B17.13
```

```
                                # Execution count [9.00e-01]
        vaddpd    %ymm3, %ymm4, %ymm0                             #20.16
        vaddpd    %ymm1, %ymm2, %ymm1                             #20.16
        vaddpd    %ymm1, %ymm0, %ymm2                             #20.16
        vextractf128 $1, %ymm2, %xmm3                             #20.16
        vaddpd    %xmm3, %xmm2, %xmm4                             #20.16
        vunpckhpd %xmm4, %xmm4, %xmm5                             #20.16
        vaddsd    %xmm5, %xmm4, %xmm6                             #20.16
        vmovsd    %xmm6, 8(%rsp)                                  #20.16[spill]
                                # LOE rax rdx rcx rbx rbp r13 r14 r15
..B17.15:                       # Preds ..B17.14 ..B17.25
                                # Execution count [1.00e+00]
        lea       1(%rcx), %rsi                                   #21.25
        cmpq      %rax, %rsi                                      #21.25
        ja        ..B17.24      # Prob 50%                        #21.25
                                # LOE rax rdx rcx rbx rbp r13 r14 r15
..B17.16:                       # Preds ..B17.15
                                # Execution count [9.00e-01]
        subq      %rcx, %rax                                      #21.25
        cmpq      $4, %rax                                        #21.25
        jb        ..B17.27      # Prob 10%                        #21.25
                                # LOE rax rdx rcx rbx rbp r13 r14 r15
..B17.17:                       # Preds ..B17.16
                                # Execution count [9.00e-01]
        vmovsd    8(%rsp), %xmm1                                  #20.16[spill]
        movq      %rax, %rsi                                      #21.25
        vxorpd    %xmm0, %xmm0, %xmm0                             #20.16
        lea       (%rdx,%rcx,8), %rdi                             #21.25
        vmovsd    %xmm1, %xmm0, %xmm1                             #20.16
        andq      $-4, %rsi                                       #21.25
        vbroadcastsd (%rsp), %ymm0                                #17.8[spill]
        xorl      %r8d, %r8d                                      #21.25
        vmovaps   %xmm1, %xmm1                                    #20.16
                                # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                ↪  r15 ymm0 ymm1
..B17.18:                       # Preds ..B17.18 ..B17.17
                                # Execution count [5.00e+00]
        vmulpd    (%rdi,%r8,8), %ymm0, %ymm2                      #23.9
        addq      $4, %r8                                         #21.25
        vaddpd    %ymm1, %ymm2, %ymm1                             #23.9
        cmpq      %rsi, %r8                                       #21.25
        jb        ..B17.18      # Prob 82%                        #21.25
                                # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                ↪  r15 ymm0 ymm1
..B17.19:                       # Preds ..B17.18
                                # Execution count [9.00e-01]
        vextractf128 $1, %ymm1, %xmm0                             #20.16
        vaddpd    %xmm0, %xmm1, %xmm2                             #20.16
        vunpckhpd %xmm2, %xmm2, %xmm3                             #20.16
        vaddsd    %xmm3, %xmm2, %xmm4                             #20.16
        vmovsd    %xmm4, 8(%rsp)                                  #20.16[spill]
                                # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
..B17.20:                       # Preds ..B17.19 ..B17.27
```

```
                                # Execution count [1.00e+00]
        cmpq      %rax, %rsi                                    #21.25
        jae       ..B17.24      # Prob 9%                       #21.25
                                # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
..B17.21:                       # Preds ..B17.20
                                # Execution count [9.00e-01]
        vmovsd    8(%rsp), %xmm1                                #21.25[spill]
        lea       (%rdx,%rcx,8), %rdx                           #21.25
        vmovsd    (%rsp), %xmm2                                 #21.25[spill]
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B17.22:                       # Preds ..B17.22 ..B17.21
                                # Execution count [5.00e+00]
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #23.9
        incq      %rsi                                          #21.25
        vaddsd    %xmm1, %xmm0, %xmm1                           #23.9
        cmpq      %rax, %rsi                                    #21.25
        jb        ..B17.22      # Prob 82%                      #21.25
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B17.23:                       # Preds ..B17.22
                                # Execution count [9.00e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
                                # LOE rbx rbp r13 r14 r15
..B17.24:                       # Preds ..B17.23 ..B17.20 ..B17.15 ..B17.32
                                # Execution count [1.00e+00]
# Begin ASM
# # integrate_dummy exit
# End ASM
```

Векторизованное интегрирование

```
# Begin ASM
# # integrate_omp_simd enter
# End ASM
                                # LOE rbx rbp r12 r13 r14 r15
..B19.33:                       # Preds ..B19.34
                                # Execution count [1.00e+00]
        movq      (%r12), %rdx                                  #47.24
        movq      8(%r12), %rax                                 #47.24
        vxorpd    %xmm0, %xmm0, %xmm0                           #45.16
        subq      %rdx, %rax                                    #47.24
        vmovsd    %xmm0, 8(%rsp)                                #45.16[spill]
        sarq      $3, %rax                                      #47.24
        je        ..B19.25      # Prob 50%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.2:                        # Preds ..B19.33
                                # Execution count [5.00e-01]
        je        ..B19.25      # Prob 50%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.3:                        # Preds ..B19.2
                                # Execution count [4.50e-01]
        cmpq      $16, %rax                                     #47.5
        jb        ..B19.26      # Prob 10%                      #47.5
```

```
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.4:                        # Preds ..B19.3
                                # Execution count [4.50e-01]
        cmpq      $29, %rax                                     #47.5
        jb        ..B19.29      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp r13 r14 r15 xmm0
..B19.5:                        # Preds ..B19.4
                                # Execution count [4.50e-01]
        movq      %rdx, %rdi                                    #47.5
        andq      $31, %rdi                                     #47.5
        je        ..B19.12      # Prob 50%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.6:                        # Preds ..B19.5
                                # Execution count [4.50e-01]
        testq     $7, %rdi                                      #47.5
        jne       ..B19.26      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.7:                        # Preds ..B19.6
                                # Execution count [2.25e-01]
        negq      %rdi                                          #47.5
        addq      $32, %rdi                                     #47.5
        shrq      $3, %rdi                                      #47.5
        lea       16(%rdi), %rcx                                #47.5
        cmpq      %rcx, %rax                                    #47.5
        jb        ..B19.26      # Prob 10%                      #47.5
                                # LOE rax rdx rbx rbp rdi r13 r14 r15 xmm0
..B19.8:                        # Preds ..B19.7
                                # Execution count [4.50e-01]
        movq      %rax, %rcx                                    #47.5
        xorl      %esi, %esi                                    #47.5
        subq      %rdi, %rcx                                    #47.5
        andq      $15, %rcx                                     #47.5
        negq      %rcx                                          #47.5
        addq      %rax, %rcx                                    #47.5
        testq     %rdi, %rdi                                    #47.5
        je        ..B19.13      # Prob 9%                       #47.5
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm0
..B19.9:                        # Preds ..B19.8
                                # Execution count [4.50e-01]
        vmovapd   %xmm0, %xmm1                                  #
        vmovsd    (%rsp), %xmm2                                 #[spill]
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm1 xmm2
..B19.10:                       # Preds ..B19.10 ..B19.9
                                # Execution count [2.50e+00]
        vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #49.9
        incq      %rsi                                          #47.5
        vaddsd    %xmm1, %xmm0, %xmm1                           #49.9
        cmpq      %rdi, %rsi                                    #47.5
        jb        ..B19.10      # Prob 82%                      #47.5
                                # LOE rax rdx rcx rbx rbp rsi rdi r13 r14 r15
                                ↪  xmm1 xmm2
```

```
..B19.11:                       # Preds ..B19.10
                                # Execution count [4.50e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
        jmp       ..B19.13      # Prob 100%                     #
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
..B19.12:                       # Preds ..B19.5
                                # Execution count [2.03e-01]
        movq      %rax, %rcx                                    #47.5
        andq      $15, %rcx                                     #47.5
        negq      %rcx                                          #47.5
        addq      %rax, %rcx                                    #47.5
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15
..B19.13:                       # Preds ..B19.12 ..B19.11 ..B19.8 ..B19.29
                                # Execution count [4.50e-01]
        vmovsd    8(%rsp), %xmm1                                #45.16[spill]
        vxorpd    %ymm3, %ymm3, %ymm3                           #45.16
        vxorpd    %xmm0, %xmm0, %xmm0                           #45.16
        vmovdqa   %ymm3, %ymm2                                  #45.16
        vmovsd    %xmm1, %xmm0, %xmm4                           #45.16
        vbroadcastsd (%rsp), %ymm0                              #42.8[spill]
        vmovdqa   %ymm2, %ymm1                                  #45.16
        vmovaps   %xmm4, %xmm4                                  #45.16
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                ↪  ymm1 ymm2 ymm3 ymm4
..B19.14:                       # Preds ..B19.14 ..B19.13
                                # Execution count [2.50e+00]
        vfmadd231pd (%rdx,%rdi,8), %ymm0, %ymm4                 #49.9
        vfmadd231pd 32(%rdx,%rdi,8), %ymm0, %ymm3               #49.9
        vfmadd231pd 64(%rdx,%rdi,8), %ymm0, %ymm2               #49.9
        vfmadd231pd 96(%rdx,%rdi,8), %ymm0, %ymm1               #49.9
        addq      $16, %rdi                                     #47.5
        cmpq      %rcx, %rdi                                    #47.5
        jb        ..B19.14      # Prob 82%                      #47.5
                                # LOE rax rdx rcx rbx rbp rdi r13 r14 r15 ymm0
                                ↪  ymm1 ymm2 ymm3 ymm4
..B19.15:                       # Preds ..B19.14
                                # Execution count [4.50e-01]
        vaddpd    %ymm3, %ymm4, %ymm0                           #45.16
        vaddpd    %ymm1, %ymm2, %ymm1                           #45.16
        vaddpd    %ymm1, %ymm0, %ymm2                           #45.16
        vextractf128 $1, %ymm2, %xmm3                           #45.16
        vaddpd    %xmm3, %xmm2, %xmm4                           #45.16
        vunpckhpd %xmm4, %xmm4, %xmm5                           #45.16
        vaddsd    %xmm5, %xmm4, %xmm6                           #45.16
        vmovsd    %xmm6, 8(%rsp)                                #45.16[spill]
                                # LOE rax rdx rcx rbx rbp r13 r14 r15
..B19.16:                       # Preds ..B19.15 ..B19.26
                                # Execution count [5.00e-01]
        lea       1(%rcx), %rsi                                 #47.5
        cmpq      %rax, %rsi                                    #47.5
        ja        ..B19.25      # Prob 50%                      #47.5
                                # LOE rax rdx rcx rbx rbp r13 r14 r15
..B19.17:                       # Preds ..B19.16
```

```
124                                 # Execution count [4.50e-01]
125         subq      %rcx, %rax                                    #47.5
126         cmpq      $4, %rax                                      #47.5
127         jb        ..B19.28      # Prob 10%                      #47.5
128                                 # LOE rax rdx rcx rbx rbp r13 r14 r15
129 ..B19.18:                       # Preds ..B19.17
130                                 # Execution count [4.50e-01]
131         vmovsd    8(%rsp), %xmm1                                #45.16[spill]
132         movq      %rax, %rsi                                    #47.5
133         vxorpd    %xmm0, %xmm0, %xmm0                           #45.16
134         lea       (%rdx,%rcx,8), %rdi                           #49.26
135         vmovsd    %xmm1, %xmm0, %xmm1                           #45.16
136         andq      $-4, %rsi                                     #47.5
137         vbroadcastsd (%rsp), %ymm0                              #42.8[spill]
138         xorl      %r8d, %r8d                                    #47.5
139         vmovaps   %xmm1, %xmm1                                  #45.16
140                                 # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                    ↪  r15 ymm0 ymm1
141 ..B19.19:                       # Preds ..B19.19 ..B19.18
142                                 # Execution count [2.50e+00]
143         vmulpd    (%rdi,%r8,8), %ymm0, %ymm2                    #49.9
144         addq      $4, %r8                                       #47.5
145         vaddpd    %ymm1, %ymm2, %ymm1                           #49.9
146         cmpq      %rsi, %r8                                     #47.5
147         jb        ..B19.19      # Prob 82%                      #47.5
148                                 # LOE rax rdx rcx rbx rbp rsi rdi r8 r13 r14
                                    ↪  r15 ymm0 ymm1
149 ..B19.20:                       # Preds ..B19.19
150                                 # Execution count [4.50e-01]
151         vextractf128 $1, %ymm1, %xmm0                           #45.16
152         vaddpd    %xmm0, %xmm1, %xmm2                           #45.16
153         vunpckhpd %xmm2, %xmm2, %xmm3                           #45.16
154         vaddsd    %xmm3, %xmm2, %xmm4                           #45.16
155         vmovsd    %xmm4, 8(%rsp)                                #45.16[spill]
156                                 # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
157 ..B19.21:                       # Preds ..B19.20 ..B19.28
158                                 # Execution count [4.50e-01]
159         cmpq      %rax, %rsi                                    #47.5
160         jae       ..B19.25      # Prob 9%                       #47.5
161                                 # LOE rax rdx rcx rbx rbp rsi r13 r14 r15
162 ..B19.22:                       # Preds ..B19.21
163                                 # Execution count [4.50e-01]
164         vmovsd    8(%rsp), %xmm1                                #49.26[spill]
165         lea       (%rdx,%rcx,8), %rdx                           #49.26
166         vmovsd    (%rsp), %xmm2                                 #49.26[spill]
167                                 # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
168 ..B19.23:                       # Preds ..B19.23 ..B19.22
169                                 # Execution count [2.50e+00]
170         vmulsd    (%rdx,%rsi,8), %xmm2, %xmm0                   #49.9
171         incq      %rsi                                          #47.5
172         vaddsd    %xmm1, %xmm0, %xmm1                           #49.9
173         cmpq      %rax, %rsi                                    #47.5
174         jb        ..B19.23      # Prob 82%                      #47.5
```

```
                                # LOE rax rdx rbx rbp rsi r13 r14 r15 xmm1 xmm2
..B19.24:                       # Preds ..B19.23
                                # Execution count [4.50e-01]
        vmovsd    %xmm1, 8(%rsp)                                #[spill]
                                # LOE rbx rbp r13 r14 r15
..B19.25:                       # Preds ..B19.24 ..B19.21 ..B19.16 ..B19.2
    ↪  ..B19.33
                                #
                                # Execution count [1.00e+00]
# Begin ASM
# # integrate_omp_simd exit
# End ASM
```

Параллельное интегрирование

```
# Begin ASM
# # integrate_omp_parallel enter
# End ASM
                                # LOE
..B18.35:                       # Preds ..B18.36
                                # Execution count [1.00e+00]
        movl      $.2.486_2_kmpc_loc_struct_pack.18, %edi       #33.5
        movq      $0, 72(%rsp)                                  #32.16
        call      __kmpc_global_thread_num                      #33.5
                                # LOE eax
..B18.34:                       # Preds ..B18.35
                                # Execution count [1.00e+00]
        movl      %eax, 80(%rsp)                                #33.5
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
        xorl      %eax, %eax                                    #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.385:
        call      __kmpc_ok_to_fork                             #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.386:
                                # LOE eax
..B18.2:                        # Preds ..B18.34
                                # Execution count [1.00e+00]
        testl     %eax, %eax                                    #33.5
        je        ..B18.4       # Prob 50%                      #33.5
                                # LOE
..B18.3:                        # Preds ..B18.2
                                # Execution count [0.00e+00]
        addq      $-16, %rsp                                    #33.5
        .cfi_def_cfa_offset 128
        movl
        ↪  $L__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_33__par_region0_2.40,
        ↪  %edx #33.5
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
        lea       16(%rsp), %rax                                #33.5
        movl      $4, %esi                                      #33.5
        lea       72(%rax), %rcx                                #33.5
        lea       56(%rax), %r8                                 #33.5
        movq      %rax, (%rsp)                                  #33.5
```

```
        lea       64(%rax), %r9                                 #33.5
        xorl      %eax, %eax                                    #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.388:
        call      __kmpc_fork_call                              #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.389:
                                # LOE
..B18.37:                       # Preds ..B18.3
                                # Execution count [0.00e+00]
        addq      $16, %rsp                                     #33.5
        .cfi_def_cfa_offset 112
        jmp       ..B18.7       # Prob 100%                     #33.5
                                # LOE
..B18.4:                        # Preds ..B18.2
                                # Execution count [0.00e+00]
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
        xorl      %eax, %eax                                    #33.5
        movl      80(%rsp), %esi                                #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.391:
        call      __kmpc_serialized_parallel                    #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.392:
                                # LOE
..B18.5:                        # Preds ..B18.4
                                # Execution count [0.00e+00]
        movl
        ↪  $___kmpv_zero_Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_0, %esi
        ↪  #33.5
        lea       80(%rsp), %rdi                                #33.5
        lea       -8(%rdi), %rdx                                #33.5
        lea       -16(%rdx), %rcx                               #33.5
        lea       -8(%rdx), %r8                                 #33.5
        lea       (%rsp), %r9                                   #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.393:
        call
        ↪  L__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd_33__par_region0_2.40
        ↪  #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.394:
                                # LOE
..B18.6:                        # Preds ..B18.5
                                # Execution count [0.00e+00]
        movl      $.2.486_2_kmpc_loc_struct_pack.63, %edi       #33.5
        xorl      %eax, %eax                                    #33.5
        movl      80(%rsp), %esi                                #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.395:
        call      __kmpc_end_serialized_parallel                #33.5
..___tag_value__Z22integrate_omp_parallelRKSt6vectorIdSaIdEEd.396:
                                # LOE
..B18.7:                        # Preds ..B18.37 ..B18.6
                                # Execution count [1.00e+00]
# Begin ASM
# # integrate_omp_parallel exit
# End ASM
```