МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
Национальный исследовательский Нижегородский государственный университет
им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Кафедра программной инженерии

Направление подготовки
02.04.02. Фундаментальная информатика и информационные технологии

Направленность образовательной программы
магистерская программа «Инженерия программного обеспечения»

Отчет по лабораторной работе

«Проблемы векторизации»

Квалификация (степень)
магистр

Форма обучения
очная

Выполнил: студент группы 381706-1м
Чекодаев О. А.

Н. Новгород
2018 г.

Оглавление

Введение

Векторизация — это процесс преобразования алгоритма из работы с одним значением за раз в работу с набором значений (вектором) за один раз.
Современные процессоры обеспечивают прямую поддержку векторных операций, когда одна инструкция применяется к нескольким данным (SIMD). Например, ЦП с 512-битным регистром может содержать 16 32-битных двойных одинарной точности и выполнять один расчет в 16 раз быстрее, чем выполнение одной инструкции за раз. Объединение этого с многопоточными и многоядерными процессорами приводит к увеличению производительности на несколько порядков.
При последовательном вычислении отдельные элементы вектора (массива) добавляются последовательно. Дополнительное регистровое пространство в современных процессорах не используется.
В векторизованном вычислении все элементы вектора (массива) могут быть добавлены за один шаг вычисления.

Проблемы векторизации

Как упоминалось выше, векторизация используется для использования SIMD-инструкций, которые могут выполнять идентичные операции с различными данными, упакованными в большие регистры.
Общая рекомендация, позволяющая компилятору автоматически векторизовать цикл, состоит в том, чтобы гарантировать отсутствие потоковых и анти-зависимых элементов данных в разных итерациях цикла.
Некоторые компиляторы, такие как компиляторы Intel C ++ / Fortran, способны автоматически векторизовать код. Если не удалось векторизовать цикл, компилятор Intel может сообщить, почему он не смог этого сделать. Отчеты могут быть использованы для изменения кода таким образом, чтобы он стал векторизованным (при условии, что это возможно).
Существует целый ряд проблем, которые могут повлиять на эффективность векторизации. Некоторые из наиболее распространенных включают:

- Loop-carried зависимости
- Вызов функций внутри цикла
- Неизвестное количество итераций цикла
- Внешние циклы
- Косвенный доступ к памяти

В данной работе будут рассмотрены причины возникновения этих проблем и возможные варианты их решения.

Инструментарий

Для демонстрации проблем векторизации была написана небольшая программа на языке C++ (полный код программы в приложении), включающая несколько функций, содержащих основные проблемы векторизации и возможные пути их решения.
Инструменты для диагностики проблем векторизации:

1. Компилятор Visual Studio с дерективой /Qvec-report:2 показывающей какие циклы были векторизованы, какие нет и причину почему нет.
2. Intel Advisor. Vectorization Advisor Analysis.

При должном знании теории можно обойтись только первым инструментом, однако Intel Advisor способен делать более глубокий анализ и предлагать возможные пути решения проблем.

Руководство к действию

Intel’s 6 Step Program for Vectorization

Самый простой способ реализовать векторизацию - начать с 6-шагового процесса Intel. Этот процесс использует инструменты Intel, чтобы обеспечить четкий путь для преобразования существующего кода в современное высокопроизводительное программное обеспечение, использующее многоядерные и многоядерные процессоры.

Шаг 1. Измерение производительности сборки базового кода

Отправной точкой является сборка базовой версии кода. Сборка релиза важна, потому что:

- Компилятор оптимизирует ваш код
- Вам нужно иметь базовые данные, чтобы измерить, как векторизация улучшает производительность
- В идеале вы должны установить цель для производительности, чтобы знать, когда вы закончите.

Шаг 2. Определите горячие точки

Такие инструменты, как профилировщик производительности Intel VTune ™ Amplifier XE, можно использовать для профилирования вашего приложения с целью поиска наиболее трудоемких областей кода или «горячих точек». Определение горячих точек помогает сосредоточить усилия на областях оптимизации, которые принесут наибольшую пользу.

Шаг 3. Определить кандидатов на векторизацию

Отчеты компилятора, такие как Intel's Compiler Optimization Report, могут подсказать, какие циклы подходят для векторизации. Циклы в горячих точках, которые не могут быть автоматически векторизованы, могут быть изменены с использованием различных методов, позволяющих их векторизовать.

Шаг 4. Анализ кода горячей точки для измерения прироста производительности.

Такие инструменты, как советник Intel, можно использовать для измерения потенциальных выгод от векторизации конкретного кода, чтобы сосредоточить усилия на максимальном выигрыше.

Шаг 5. Реализация рекомендаций по векторизации

Реализуйте рекомендации по векторизации кода, используя переупорядочение кода, подсказки компилятора или другие методы.

Шаг 6. Повторите

Процесс повторяется и должен повторяться до тех пор, пока не будет достигнута желаемая производительность.

В данной работе будут пропущены шаги 2 и 3 и вместо них будет просмотрен анализ компилятора Visual Studio, т.к. и он достаточно хорошо справляется с этими задачами.

Решение проблем векторизации на реальном примере

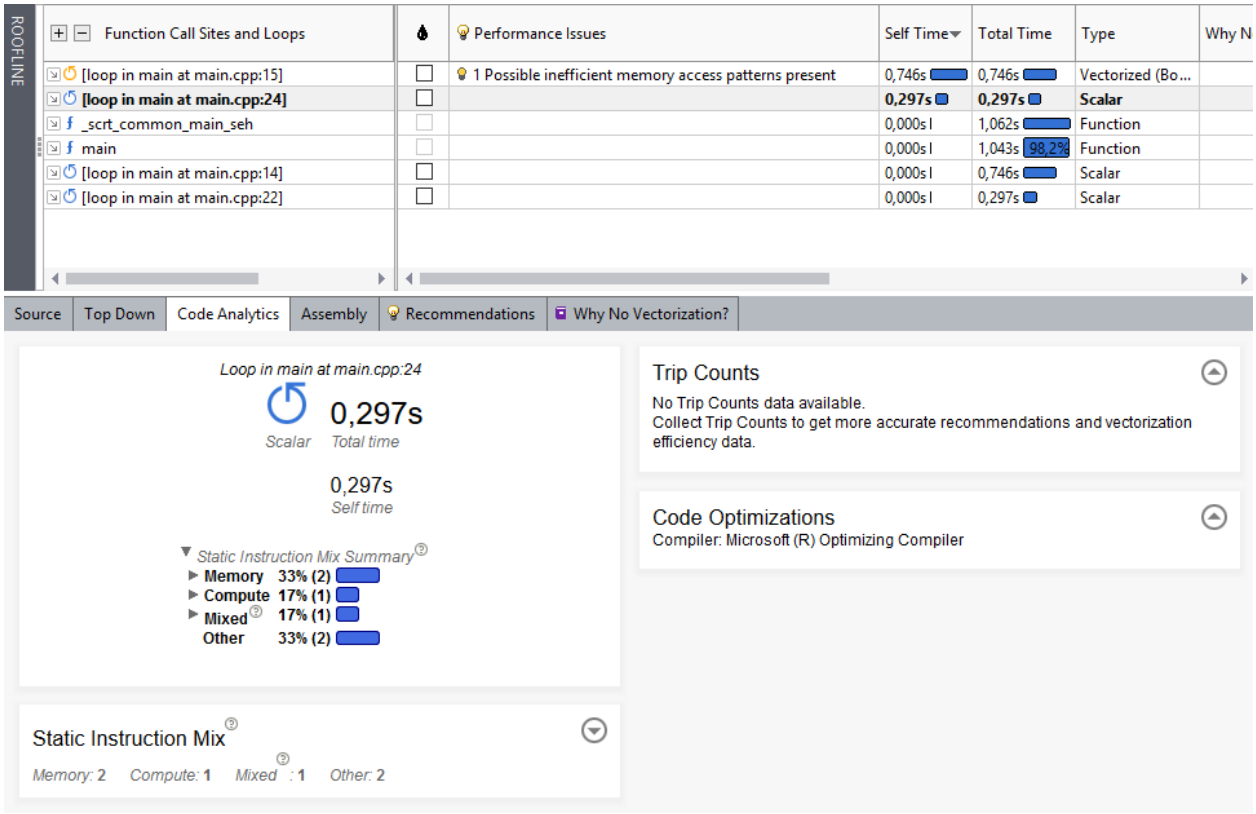
Рассмотрим первую проблему: *Loop-carried dependencies*.

Проанализирует следующий код и отчет компилятора:

```
21      //LoopDependencies
22      for (int j = 0; j < ROW; j++) {
23          for (int i = 0; i < COL - 1; i += 1) {
24              Data[j][i + 1] = Data[j][i] + Data2[j][i];
25
26          }
27      }
28      //
```

```
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(14) : info C5002: цикл не векторизирован по следующей причине: "1106"
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(23) : info C5002: цикл не векторизирован по следующей причине: "1200"
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(22) : info C5002: цикл не векторизирован по следующей причине: "1106"
```

При рассмотрении данной проблемы нас интересуют вложенные циклы. Цикл на строке 23 не векторизован по следующей причине: «Цикл содержит связанные с циклом зависимости данных, исключающие векторизацию. Различные итерации цикла взаимодействуют между собой таким образом, что векторизация цикла приведет к получению ошибочных результатов; автоматический векторизатор не может удостовериться в отсутствии таких зависимостей данных.»[2] Посмотрим на отчет Intel Advisor:



Теперь попробуем проверить эти циклы на наличие зависимостей. Для этого воспользуемся Intel Advisor Check Dependencies:

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint	Site Name	Recommendati...
[loop in main at main.cpp:24]	No information available	100% / 0% / 0%	All unit strides	26KB	loop_site_58	
[loop in main at main.cpp:35]	RAW:1	No information available	No information available	No information available	loop_site_40	

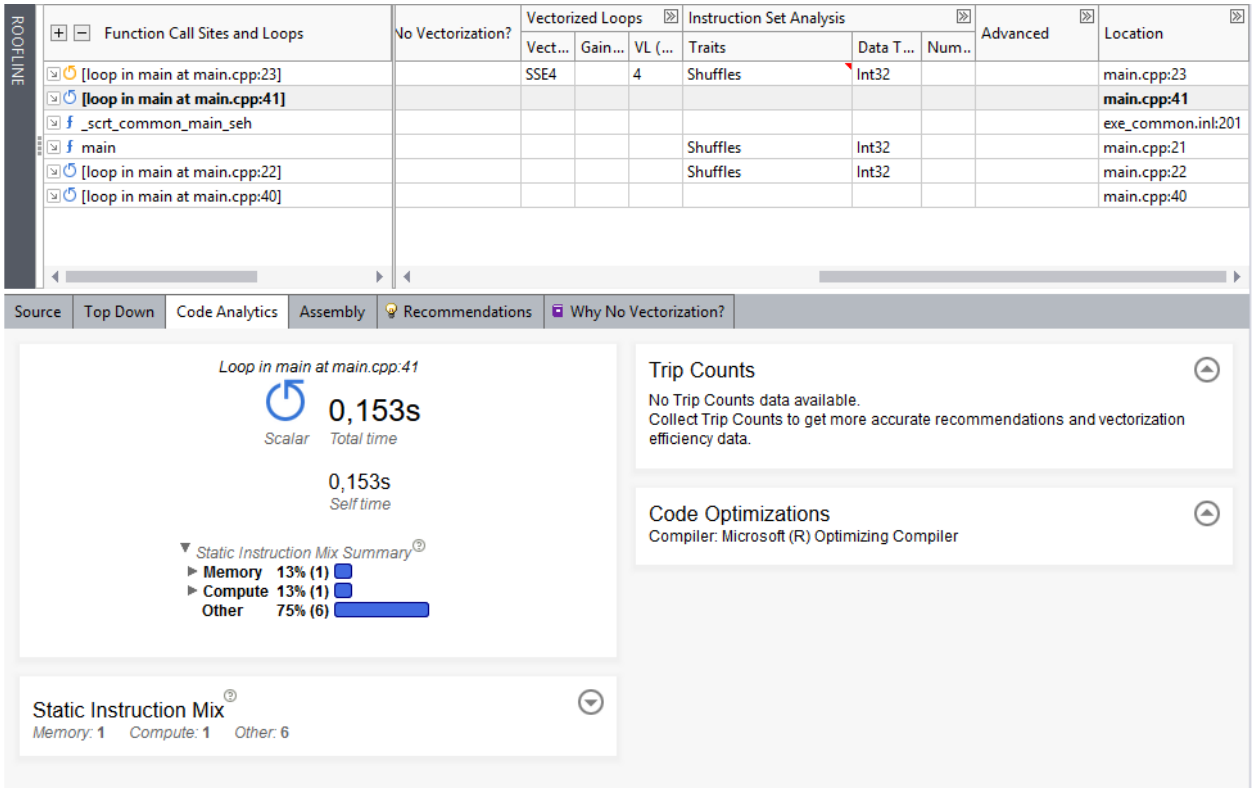
Можно заметить, что цикл действительно имеет зависимости: read after write dependencies. Циклы с такими зависимостями не векторизуются, т.к. в противном случае можно потерять данные. Для оптимизации программы необходимо избегать подобных зависимостей. Возможность этого определяется самой задачей. Для такого абстрактного цикла, приведенного в программе подобный анализ невозможен, т.к. единственное назначение цикла показать наличие проблемы.

Рассмотрим вторую проблему: вызов функций внутри цикла и неизвестное количество итераций цикла.
Рассмотрим следующий код и отчет компилятора:

```
int Calculations (int i) {
    if (i % 2 == 0) {
        return COL;
    } else {
        return COL / 2;
    }
}
```

```
39      //function calls
40      for (int j = 0; j < ROW; j++) {
41          for (int i = 0; i < Calculations (j); i++) {
42              Data[j][i] = i * j;
43          }
44      }
45      //
```

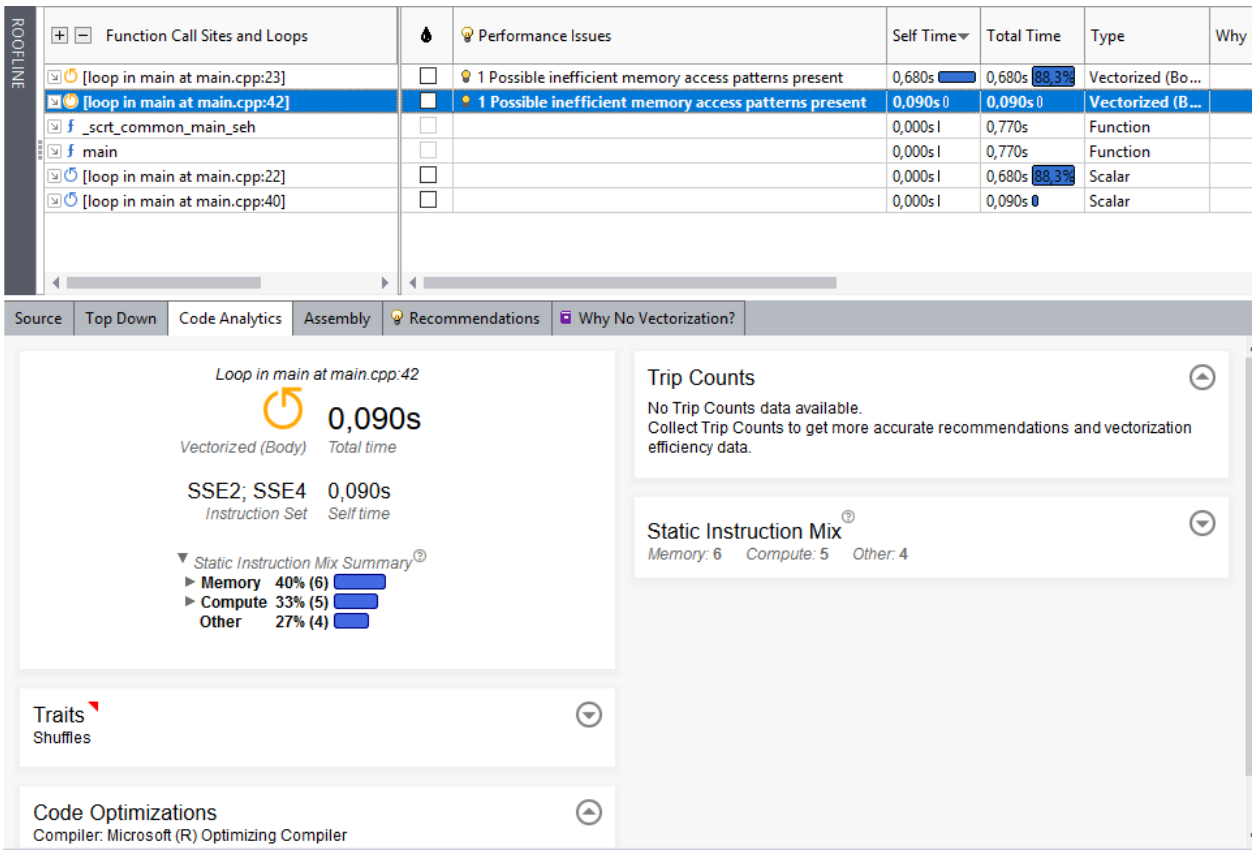
```
1>  --- Анализ функции: main
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(41) : info C5002: цикл не векторизирован по следующей причине: "500"
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(23) : info C5001: векторизированный цикл
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(22) : info C5002: цикл не векторизирован по следующей причине: "1106"
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(40) : info C5002: цикл не векторизирован по следующей причине: "1100"
```



Как видно из анализа цикл не векторизован. Это случилось из-за того, что компилятор не понимает какое количество итераций будет выполнено циклом. Произошло это из-за того, что вместо точного числа стоит функция, которая возвращает значение, зависимое от внешнего цикла. Самый простой способ решить эту проблему – вынести подсчет количества элементов из объявления цикла. Посмотрим на отчет после выполнения этих действий.

```
39      //function calls
40      for (int j = 0; j < ROW; j++) {
41          int n = Calculations(j);
42          for (int i = 0; i < n; i++) {
43              Data[j][i] = i * j;
44          }
45      }
```

```
1>  --- Анализ функции: main
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(23) : info C5001: векторизированный цикл
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(22) : info C5002: цикл не векторизирован по следующей причине: "1106"
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(42) : info C5001: векторизированный цикл
1>  f:\documents\visual studio 2015\projects\vectorisationproject\vectorisationproject\main.cpp(40) : info C5002: цикл не векторизирован по следующей причине: "1106"
```

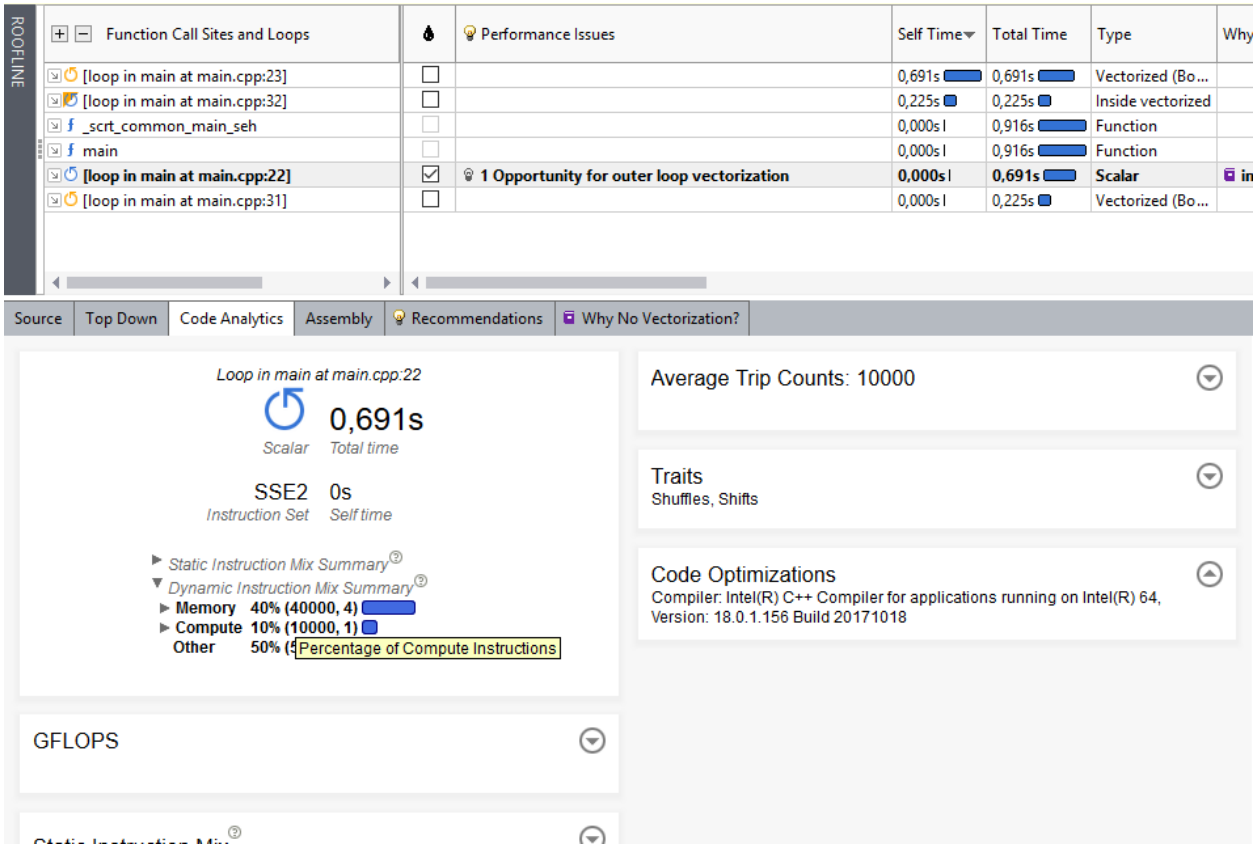


Из анализа можно наблюдать, что цикл успешно векторизован и время его выполнения сократилось почти в 2 раза.

Рассмотрим 3 проблему: внешние циклы
Рассмотрим следующий код и отчет компилятора:

```
22      for (int j = 0; j < ROW; j++) {
23          for (int i = 0; i < COL; i++) {
24              Data[j][i] = i * j;
25              Data2[j][i] = i * j;
26          }
27      }
28
```

Как видно из анализа внешний цикл не векторизировался.
Попробуем векторизовать внешний цикл у уже векторизованного внутреннего цикла.



Проведя все необходимые проверки, а именно Dependencies Check and Memory Access Patterns check, получаем следующую рекомендацию.

All Advisor-detectable issues: C++ | Fortran

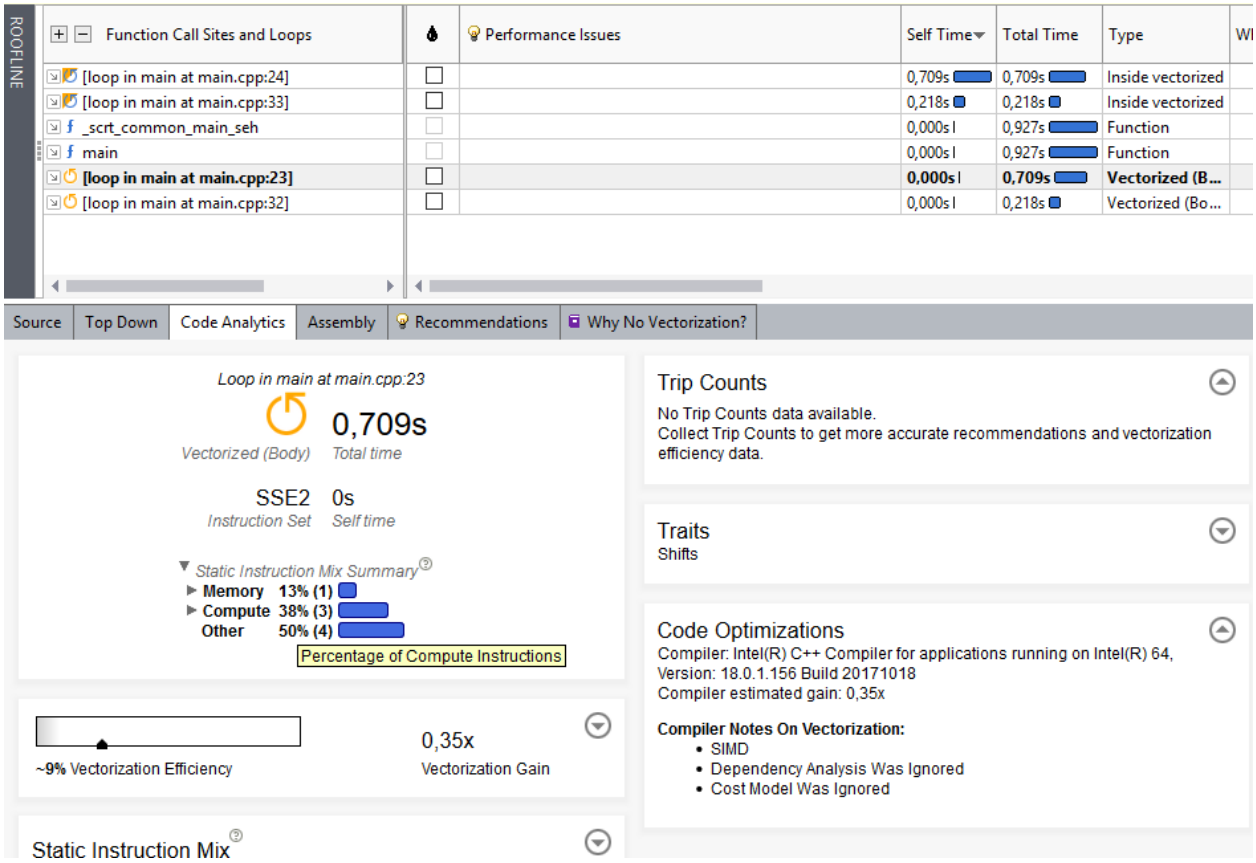
Recommendation: Consider outer loop vectorization

The compiler never targets loops other than innermost ones, so it vectorized the inner loop while did not vectorize the outer loop. However outer loop vectorization could be more profitable because of better Memory Access Pattern, higher Trip Counts or better Dependencies profile.
To enforce outer loop vectorization:

Target	Directive
Outer loop	#pragma omp simd
Inner loop	#pragma novector

Example:

```
#pragma omp simd
for(i=0; i<N; i++)
...
```



Цикл векторизован, однако это не принесло ожидаемого ускорения, т.к. основная активность происходит во внутреннем цикле и выгоднее векторизовать именно его.

Заключение

Были рассмотрены такие проблемы векторизации как Loop-carried зависимости, вызов функций внутри цикла, неизвестное количество итераций цикла, внешние циклы. Были описаны возможные варианты векторизации циклов, где это возможно. Была реализована программа-бенчмарк, демонстрирующая основные проблемы векторизации.

Литература

[1] HPC codes modernizationusing vector parallelism – part 2 (tools) Zakhar A. Matveev, PhD, Intel Russia, Intel Software and Services Group
[2] Visual Studio documentation Сообщения векторизатора и параллелизатора.
[3] Vectorization, Kirill Rogozhin, Intel, March 2017
[4] Vectorization of Performance Dies for the Latest AVX SIMD, Kevin O'Leary, Intel, Aug 2016,
[5] A Guide to Vectorization with Intel® C++ Compilers, Intel, Nov 2010,
[6] Vectorization Codebook, Intel, Sep 2015,

Приложения

Код программы

```
#include <Windows.h>

#define opt

#define ROW 10000
#define COL 10000

unsigned int Data[ROW][COL];
unsigned int Data2[ROW][COL];

int Calculations (int i) {
    if (i % 2 == 0) {
        return COL;
    } else {
        return COL / 2;
    }
}

int main()
{
#ifdef opt
#pragma simd
#endif
    for (int j = 0; j < ROW; j++) {
#ifdef opt
#pragma novector
#endif
        for (int i = 0; i < COL; i++) {
            Data[j][i] = i * j;
            Data2[j][i] = i * j;
        }
    }

    //LoopDependencies
#ifndef opt
    for (int j = 0; j < ROW; j++) {
        for (int i = 0; i < COL - 1; i += 1) {
            Data[j][i + 1] = Data[j][i] + Data2[j][i];
        }
    }
#endif
    //


    //function calls
    for (int j = 0; j < ROW; j++) {
#ifdef opt
        int n = Calculations(j);
        for (int i = 0; i < n; i++) {
            Data[j][i] = i * j;
        }
#elif
        for (int i = 0; i < Calculations(j); i++) {
            Data[j][i] = i * j;
        }
#endif
    }
    //

    return 0;
};
```