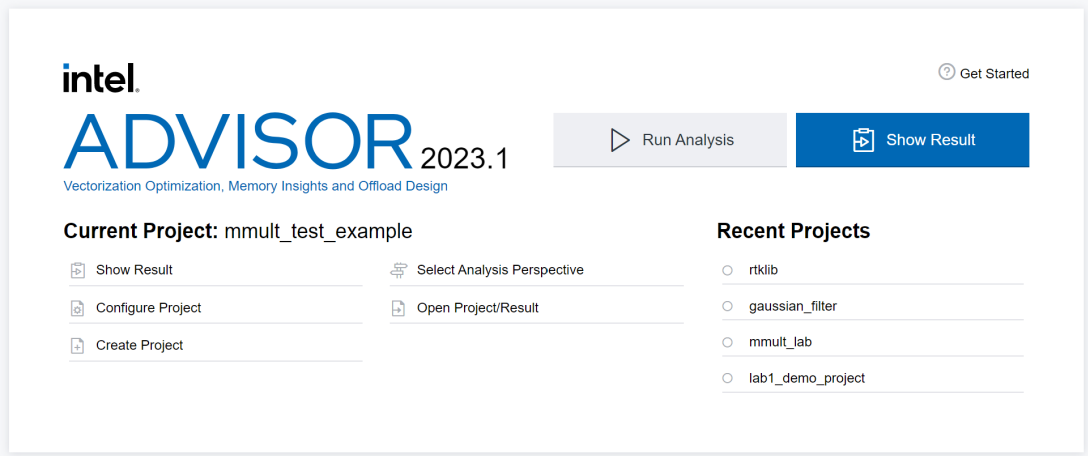
Lab2. Intel advisor

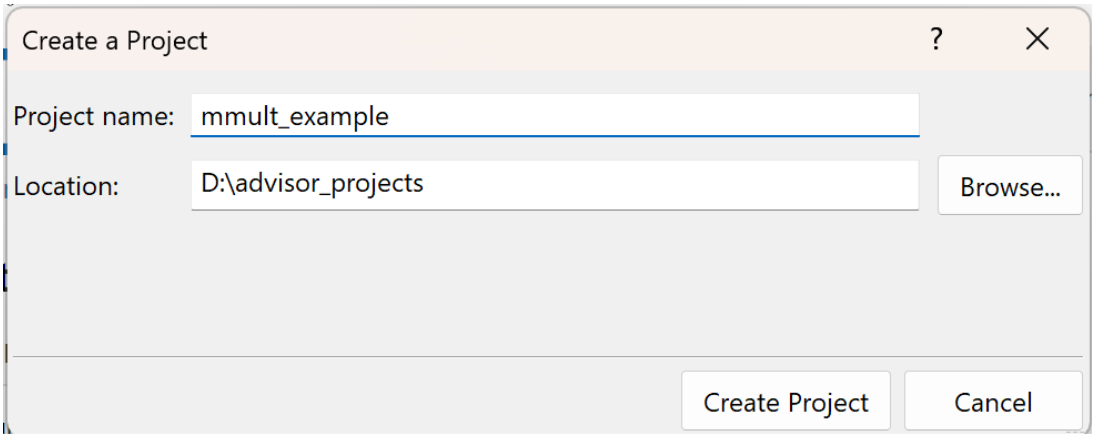
Этап с настройкой окружения для запуска Intel Advisor соответствует аналогичному шагу в предыдущей лабораторной работе. Запуск Intel Advisor с использованием пользовательского интерфейса (выполняется из консоли с настроенным окружением):

```
C:\Users\k.sandalov>advisor-gui
```

Для создания нового проекта - **Create project**:



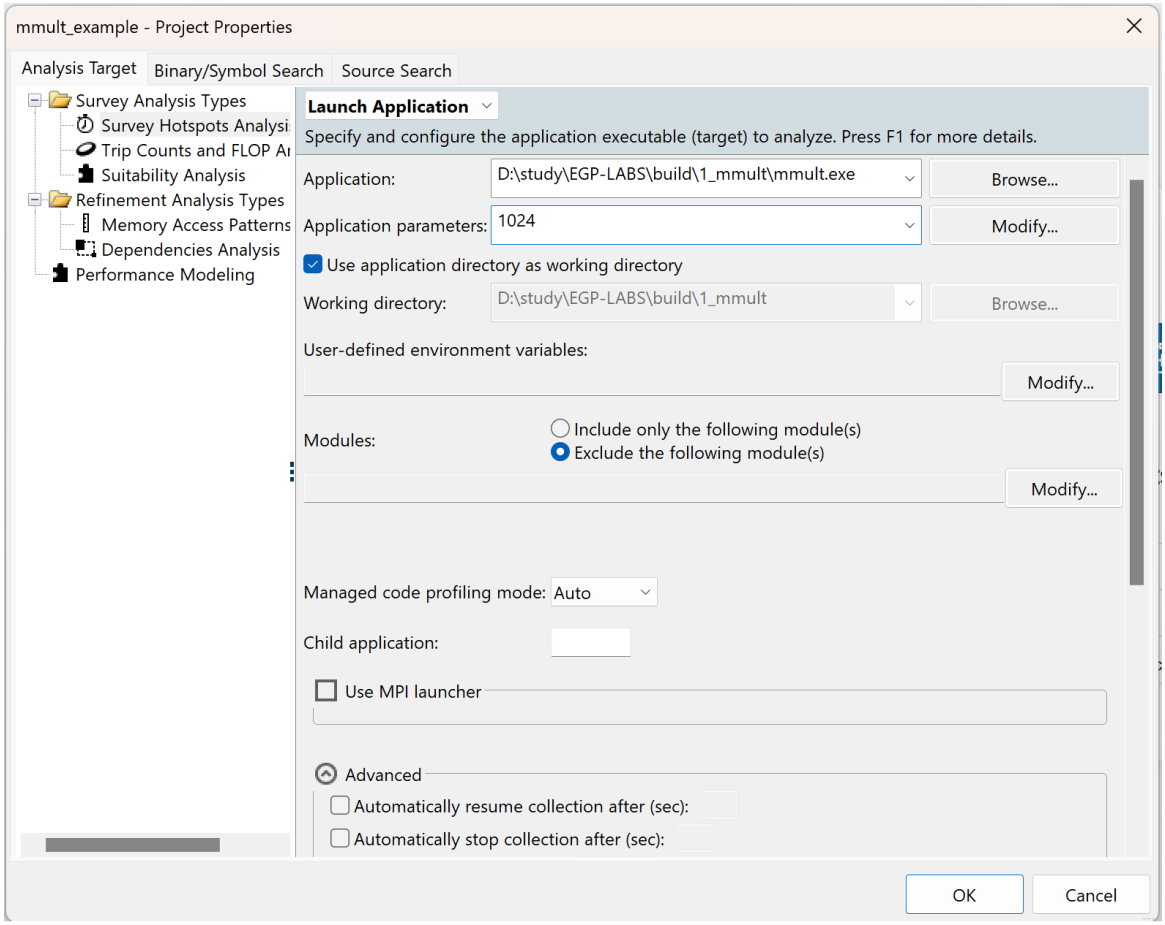
Выбор имени проекта и его расположение:



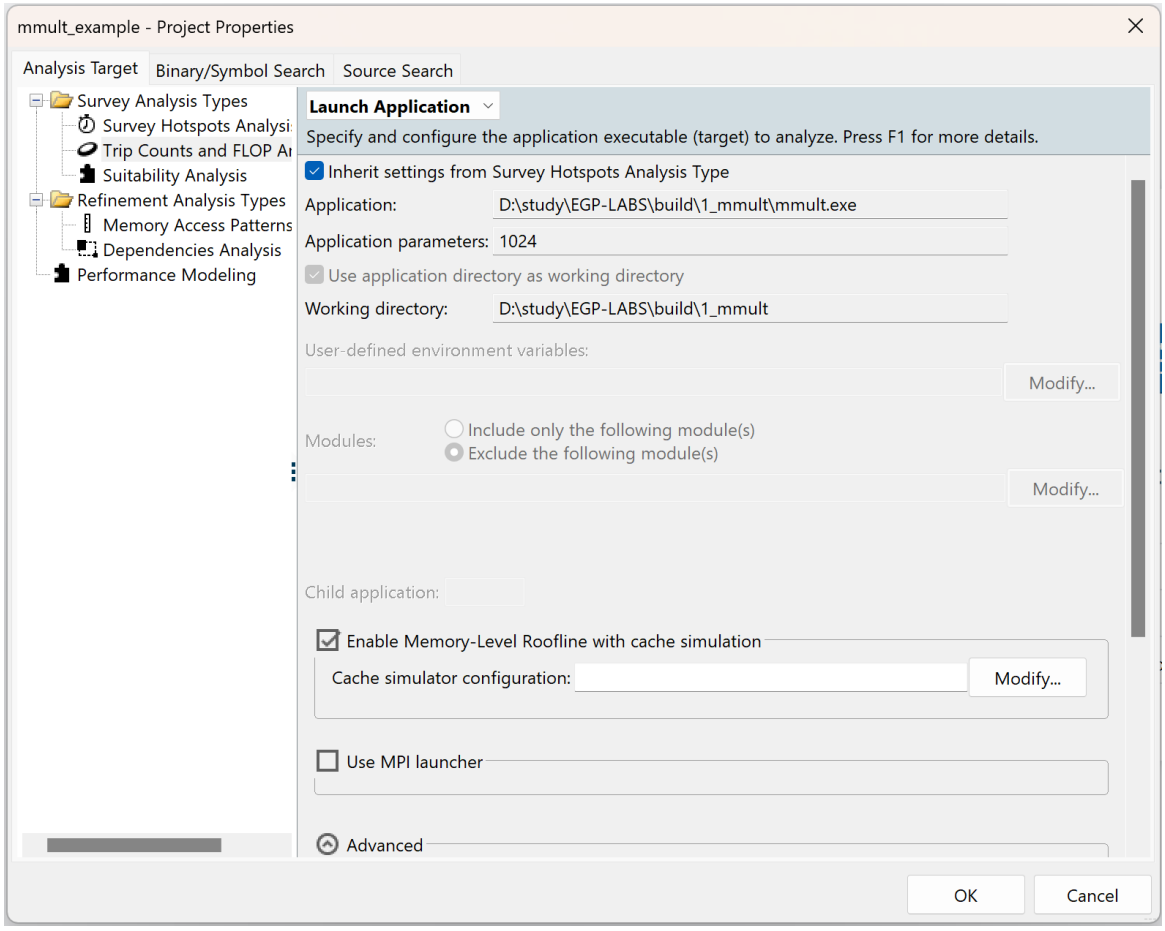
После этого, для начальной настройки проекта нужно на вкладке **Analysis Target** -> **Survey Hotspots Analysis** выбрать анализируемое приложение, указав полный путь до полученного .exe файла в окне **Application**. Соответственно, аргумены запускаемого приложения (выбран режим **Launch Application**), задаются в окне **Application parameters**.

В рамках лабораторной работы используем тестовое приложение mmult из этого репозитория. Сперва необходимо собрать его базовую версию со следующими опциями компиляции:

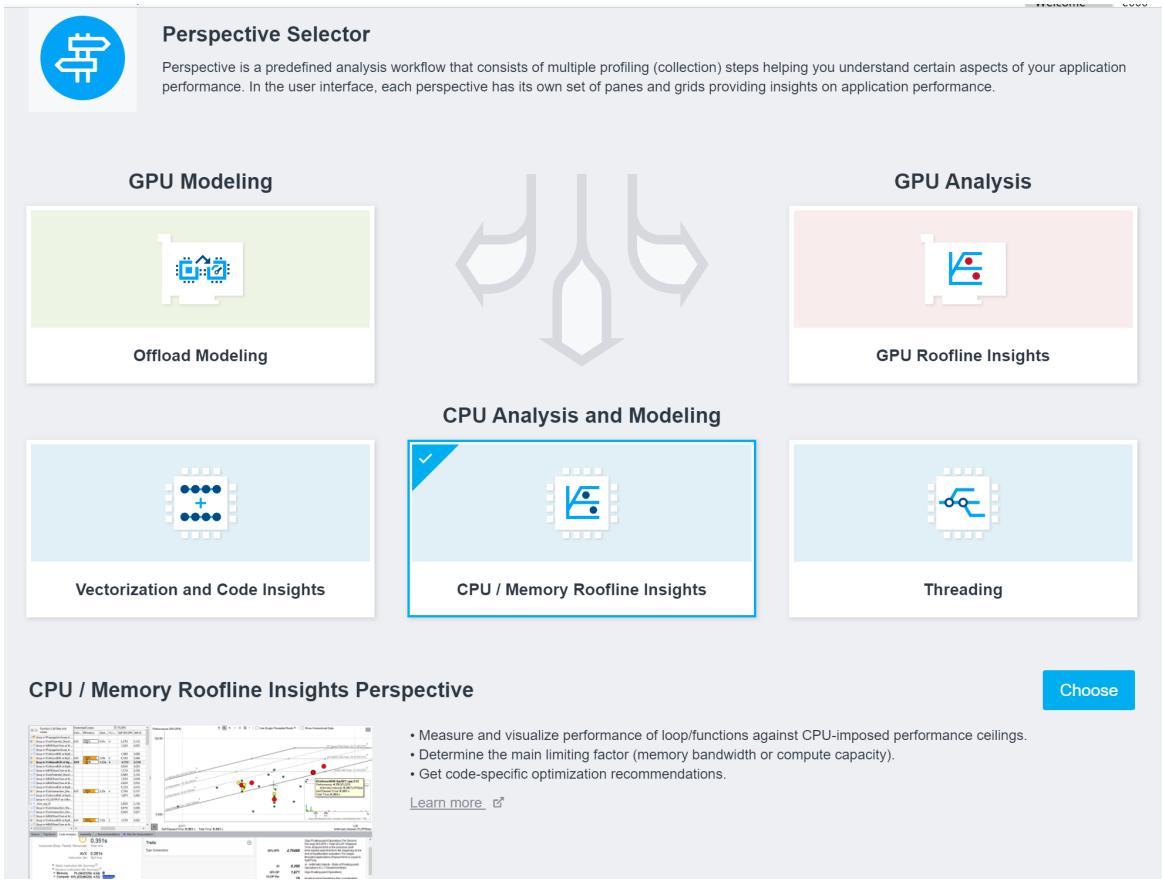
```
target_compile_options(mmult PRIVATE
/I$ENV{ONEAPI_ROOT}\\advisor\\latest\\include)
```



Дополнительно на вкладке **Analysis Target** -> **Trip Counts and FLOP Analysis** необходимо выбрать пункт **Enable Memory-Level Roofline with cache simulation**:



При создании нового проекта первым должно появиться окно с выбором определенного типа анализа или же, **Perspective selector**:



Необходимо небольшое отступление, чтобы в общих чертах описать цель и средства анализа, выполняемого в Intel Advisor.

В соответствии со своим названием, Intel Advisor способен проанализаровать приложение (исходный код, отлабочная информация + динамический анализ исполняемого приложения) и дать советы по его оптимизации.

Для анализа приложения в нем представлены различные flow, так называемые, перспективы, которые помогут раскрыть различные аспекты эффективности утилизации ресурсов вашей вычислительной платформы анализируемым приложением.

Соответственно, разные flow содержат различные наборы анализов из тех, что вообще представлены в инструменте:

- Survey

  Необходим для первичного анализа (самый легковесный) горячих участков приложения и эффективности векторизации. Далее можно уже применять какие-то предложения по повышению эффективности векторизации, и обратить свое внимание на то, какие у вас есть хотспоты и попытаться их оптимизировать

- Characterization

  Служит для дальнейшего определения возможностей оптимизации хотспотов. Он позволит собрать более подробную информацию о том, как именно выполняются найденные хотсопты. Если это какие-то циклы, то, можно определить количество раельно выполненных итераци, для функций же можно получить информацию о количестве их вызовов (это все Trip Counts). Тут же можно узнать, сколько было выполнено Floating Point / Int операций (FLOP) и уже на основании двух этих анализов - Trip Counts и FLOP построить очень полезный график Roofline. Он подскажет, чем реально ограничен тот или иной хостпот - подсистемой памяти, либо вычислительной пропускной способностью (это применение векторизации и параллелизма), и отталкиваясь от этого уже можно будет думать, а нужно ли как-то улучшать чтение/запись данных или уже пытаться добавлять парарллелизм, бороться с векторизацией.

- Memory Access Patterns

  Используется для определения эффективности паттернов работы с памятью, ведь оптимальнее всего обращатьс к памяти последовательно, либо же вообще попытаться избежать лишних операций чтения/записи в более дорогу память. Необходимо обозначить интересующие вас циклы, чтобы проанализировать, как в них происходит работа с памятью

- Dependencies

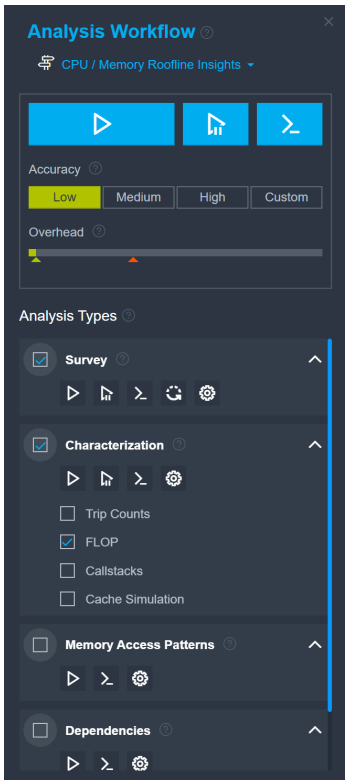
  Если выбхотите добавлять параллелизм, то необходимо помнить, что эффективная параллельная обработка данных предполагает определенную (лучше всего конечно высокую) степень независимости между данными, обрабатываемыми в разных потоках. Этот нанализ как раз таки позволяет определить, можно ли распараллелить тот или иной цикл по его итерациям эффективно, в зависимости от того, есть ли между его итерациями зависимости.

- Suitability

> Когда мы поняли, что на предыдущих анализах у нас все отлично, мы максимально оптимально работаем с памятью и независимы по итерациям, можно попытаться добавить параллелизм. Однако тут тоже не все так просто и полезно было бы оценить, как будет масштабироваться наша параллельная программа с ростом кол-ва выполняемой работы и ее распределеинем по отдельным создаваемым потокам. Для этого необходимо расставить аннотации в соответствии с рекомендациями.

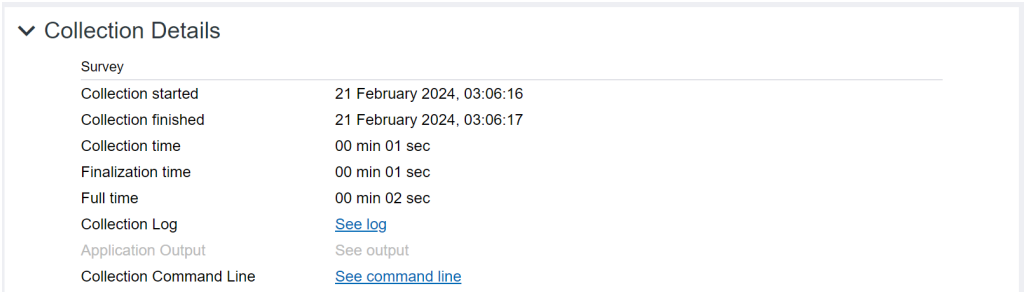
Возвращаясь к разбору Intel Advisor, рассмотрим перспективу **CPU / Memory Roofline Insights**.

В блоке **Analysis Workflow** можно переключиться между разными перспективами (были представлены на предыдущем изображении) а так же подобрать параметры для текущего типа анализа и запустить сбор данных + непосредственно сам анализ.



Для первого запуска должен быть выбран **Survey** + **Characterization (FLOP)**, сравнить время выполнения этих анализов между собой.

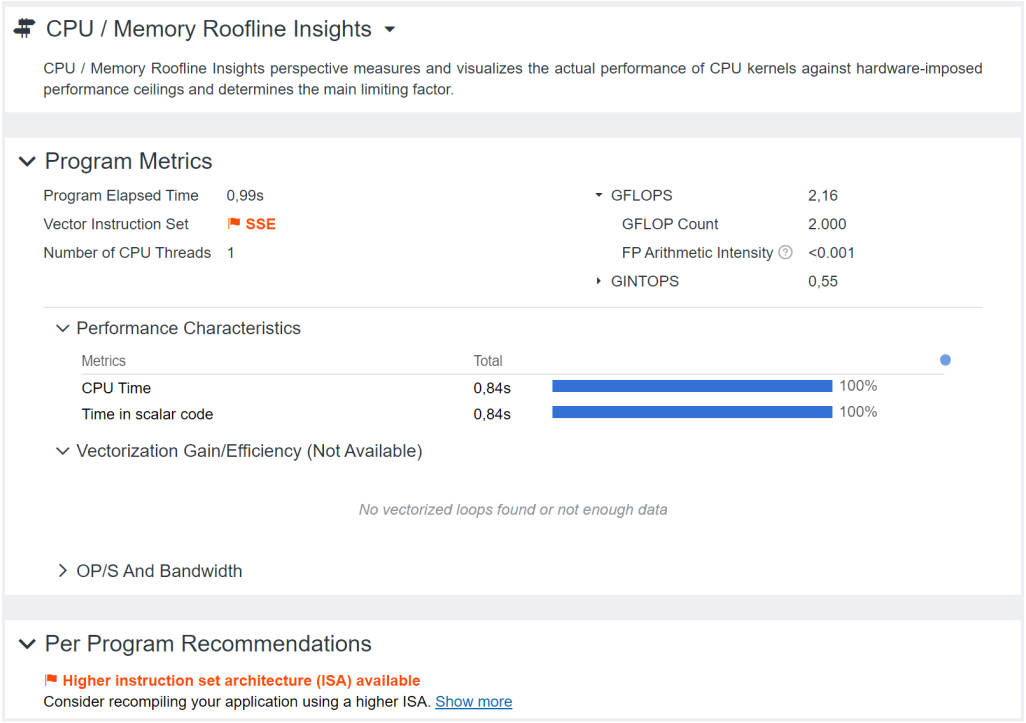
> Задание: Результаты сравнения добавить в отчет. Время выполнения анализа можно найти на вкладке **Summary** в блоке **Collection Details**:



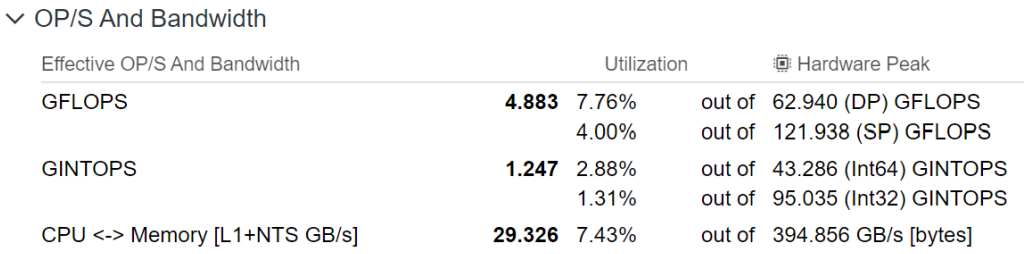
Summary:
На этой вкладке приведены общие сведения о производительности приложения:

- Количество выполняемых потоков
- Время работы программы от старта выполнения первого потока до завершения последнего
- Используемый набор векторных инструкций
- Доля векторизованных вычислений

- > GFLOPS = FLOPs / Seconds (аналогично и INTOPS)
- > ArithmeticIntensity(AI) = FLOPs / Bytes (отношение кол-ва вычислений к кол-ву запрошенных для этого байтов из памяти)



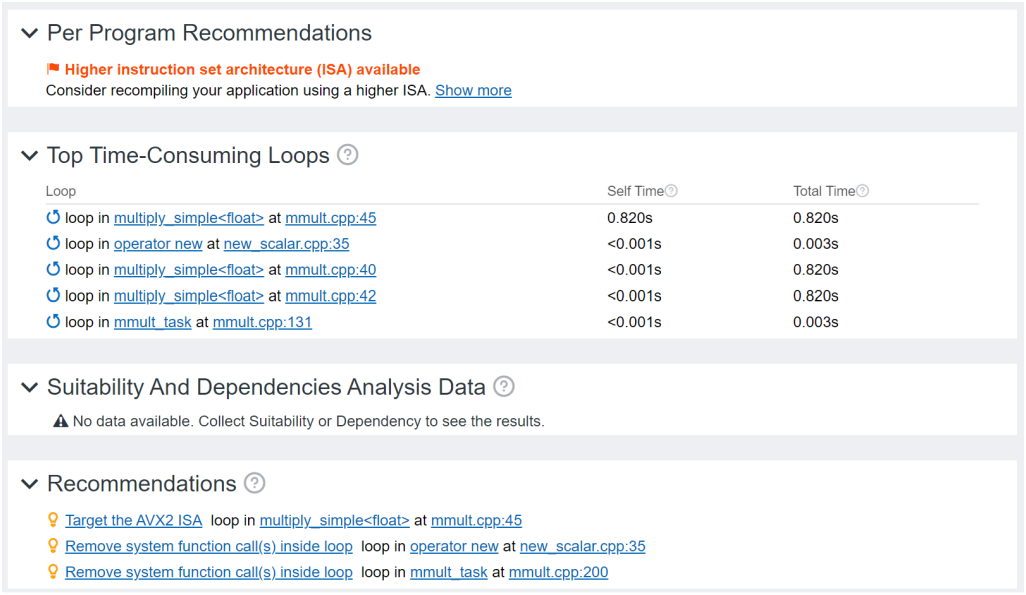
- Теоретические пиковые значения производительности для данной платформы и реально задействованная мощность:



Тут же можно найти рекомендации, применимые глобально ко всему приложению, например: Использовать более широкий набор инструкций.

> Задание: Зафиксировать время выполнения программы, факт использования векторных инструкций, текущие GFLOPS и AI для приложения. Пиковые значения пропускной способности вычислительной системы.

Топ самых тяжелых хотспотов и рекомендации для них. Подробнее про анализ хотспотов на вкладке **Survey and Roofline**:



Per Program Recommendations

Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA. Show more

Top Time-Consuming Loops

Loop	Self Time	Total Time
loop in multiply_simple<float> at mmult.cpp:45	0.820s	0.820s
loop in operator new at new_scalar.cpp:35	<0.001s	0.003s
loop in multiply_simple<float> at mmult.cpp:40	<0.001s	0.820s
loop in multiply_simple<float> at mmult.cpp:42	<0.001s	0.820s
loop in mmult_task at mmult.cpp:131	<0.001s	0.003s

Suitability And Dependencies Analysis Data

No data available. Collect Suitability or Dependency to see the results.

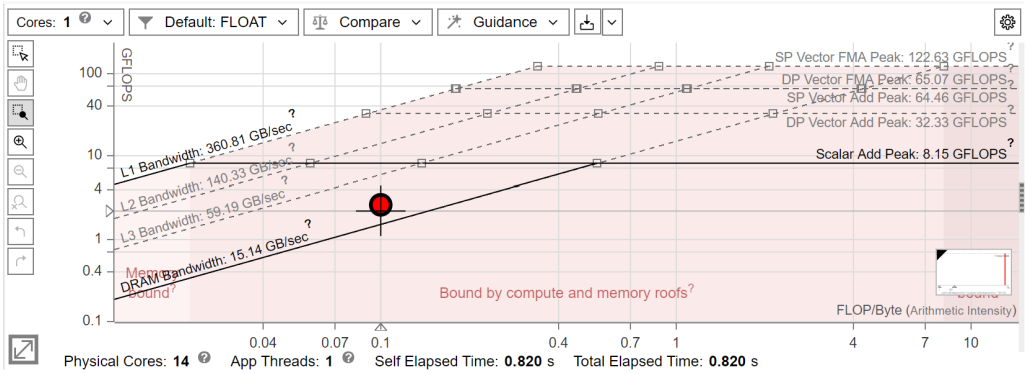
Recommendations

Target the AVX2 ISA loop in multiply_simple<float> at mmult.cpp:45

Remove system function call(s) inside loop loop in operator new at new_scalar.cpp:35

Remove system function call(s) inside loop loop in mmult_task at mmult.cpp:200

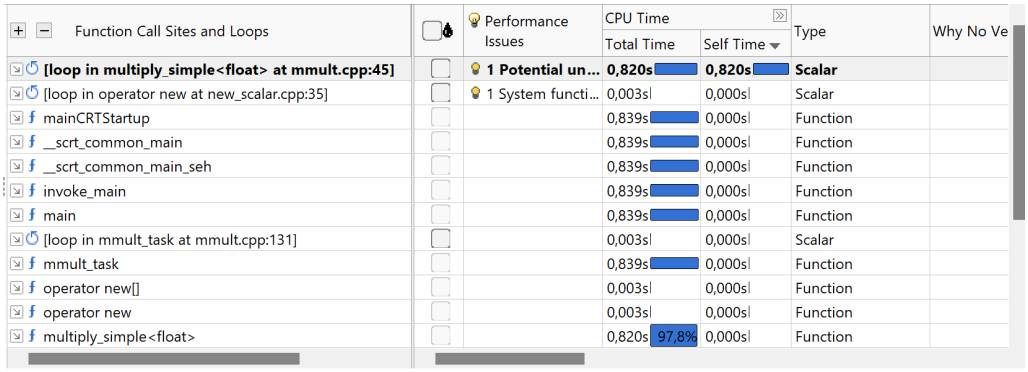
Roofline:



На графике руфлайна наглядно отображены ограничения пропускной способности (вычислительной - горизонатльные линии и подсистемы памяти - наклонные), а также то, насколько эффективно их использует анализируемое приложение. На графике отображаются точки, характеризующие производительность самых горячих участков тестового приложения. И в данном случае, это цикл на строке 45 в mmult.cpp. Для этой точки вычисляются значения GFLOPS/GINTOPS и AI, а так же, исходя из используемого набора инструкций, типа операций (FP или INT) и конфигурации подсистемы памяти ограничивающие ее крыши.

Соответственно, ограничением производительности будет служить `MIN(peakMemBandwidth x AI, peakGFLOPS)`

Survey:



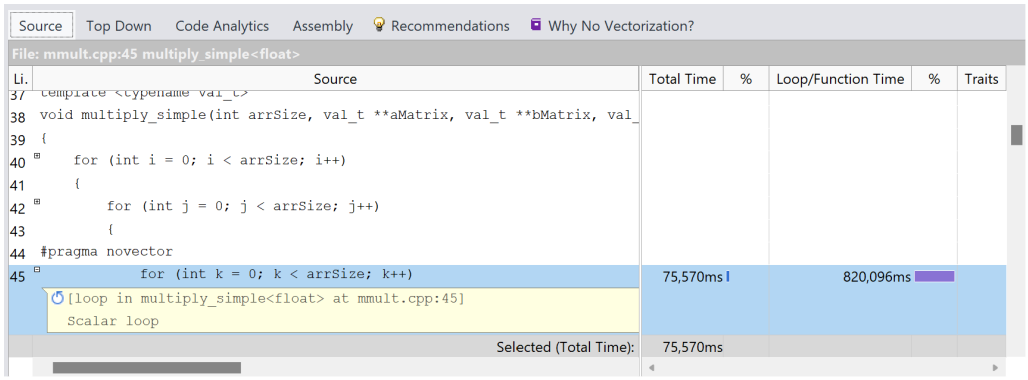
Function Call Sites and Loops	Performance Issues	CPU Time: Total Time	CPU Time: Self Time	Type	Why No Ve
[loop in multiply_simple<float> at mmult.cpp:45]	1 Potential un...	0,820s	0,820s	Scalar	
[loop in operator new at new_scalar.cpp:35]	1 System functi...	0,003s	0,000s	Scalar	
mainCRTStartup		0,839s	0,000s	Function	
__scrt_common_main		0,839s	0,000s	Function	
__scrt_common_main_seh		0,839s	0,000s	Function	
invoke_main		0,839s	0,000s	Function	
main		0,839s	0,000s	Function	
[loop in mmult_task at mmult.cpp:131]		0,003s	0,000s	Scalar	
mmult_task		0,839s	0,000s	Function	
operator new[]		0,003s	0,000s	Function	
operator new		0,003s	0,000s	Function	
multiply_simple<float>		0,820s 97,8%	0,000s	Function	

Bottom-up представление списка хотспотов приложения в виде таблицы с performance метриками.

> Задание: Определить главный хотспот. Затем выделить ограничивающие его крыши, зафиксировать в отчете.

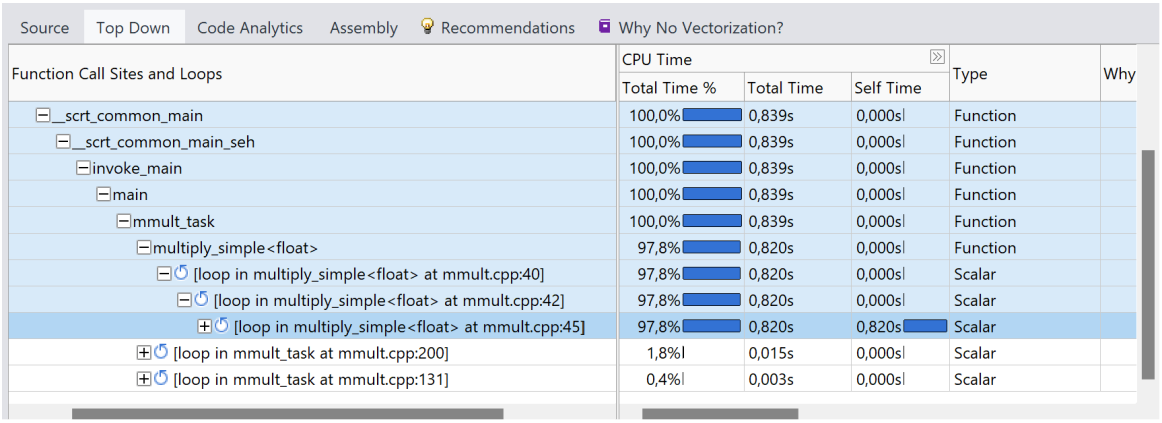
Ниже вкладки:

Source:



Выделено непосредственно место в исходном коде, соответствующее выбранному хотспоту.

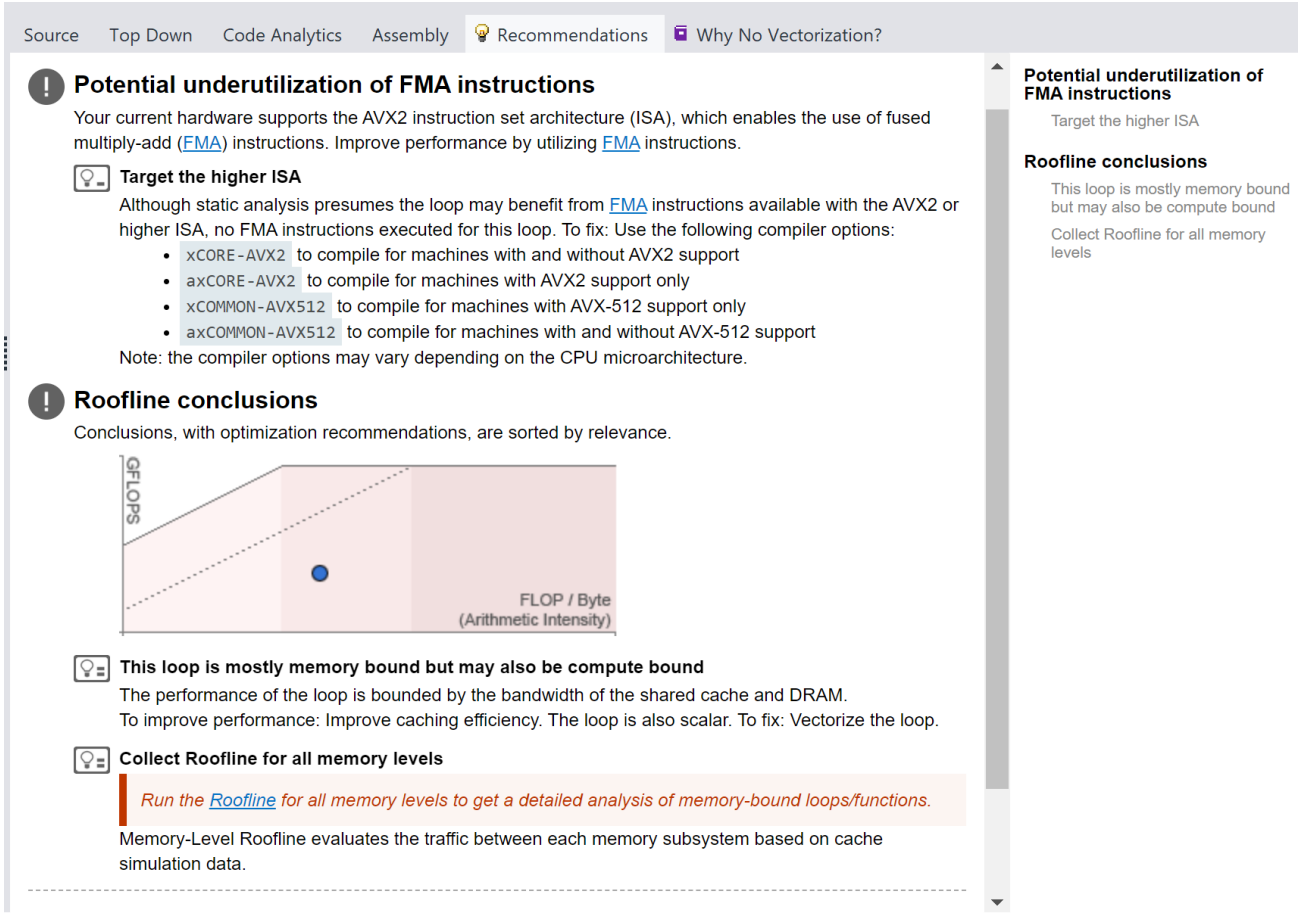
Top Down



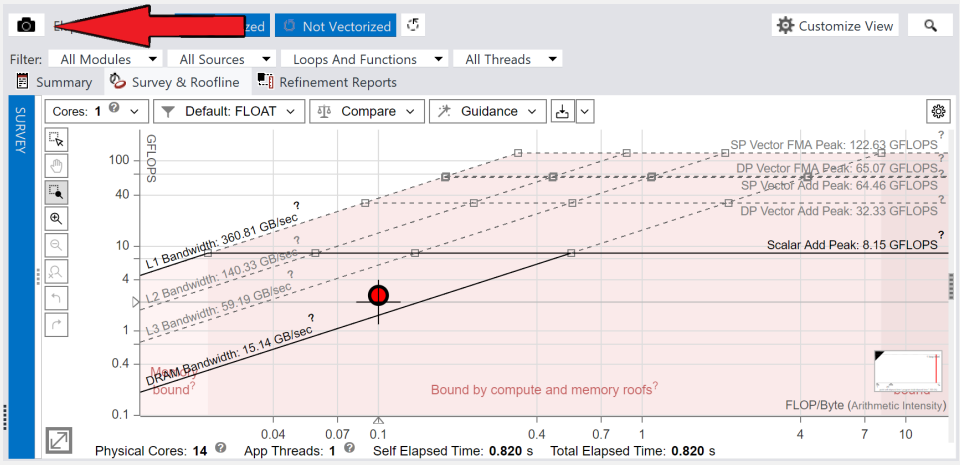
Дерево вызовов, для выбранного хотспота. Соответственно, для одного хотспота может быть выделено сразу несколько стеков вызовов. И тут уже можно посмотреть долю каждого в суммарном времени для хотспота.

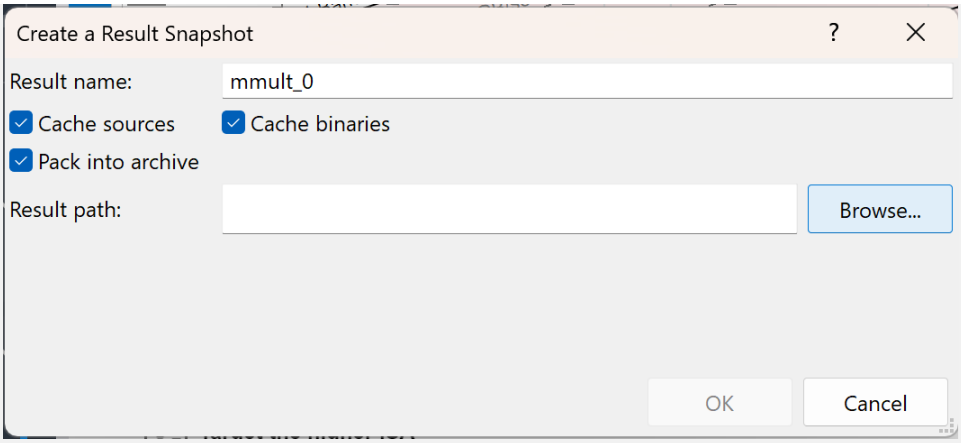
Code Analytics - суммарная информация по хостпоту + по миксу инструкций для него (соотношение разных типов инструкций - memory vs compute)

Recommendations



То, на что стоит обратить большее внимание. Довольно важной частью данного инструмента является система предоставления рекомендаций по анализируемому коду, советы, на что стоит обратить внимание, и даже, возможно, как можно исправить те или иные проблемы с производительностью приложения.

> Далее перед любыми модификациями приложения необходимо делать снэпшоты:
>
> 
>



Снимки профиля, которые далее можно будет сравнивать между собой.

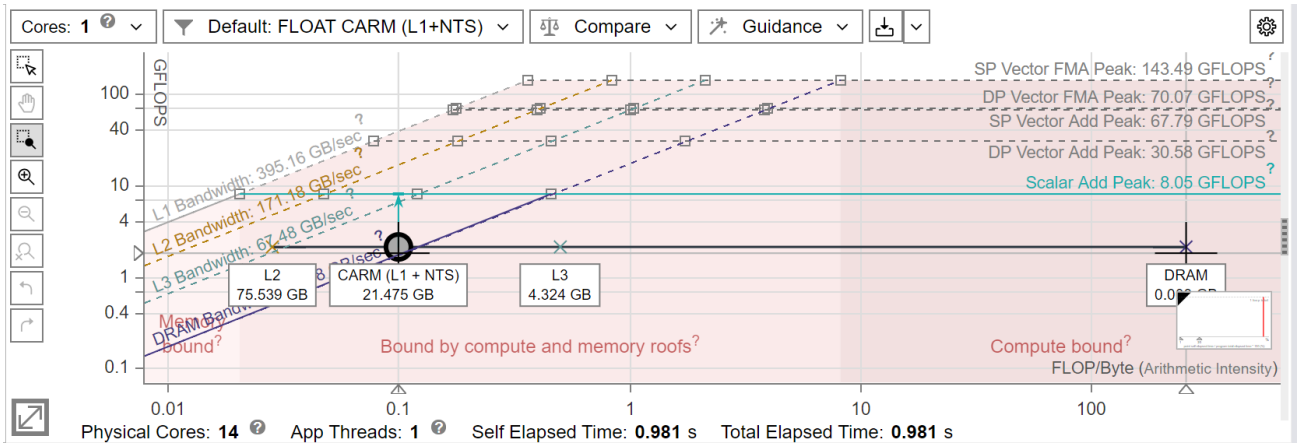
Итак, в данном примере нам предлагается две ракомендации:

- Использовать более широкий набор инструкций + FMA
- Собрать Roofline для всех уровней памяти.

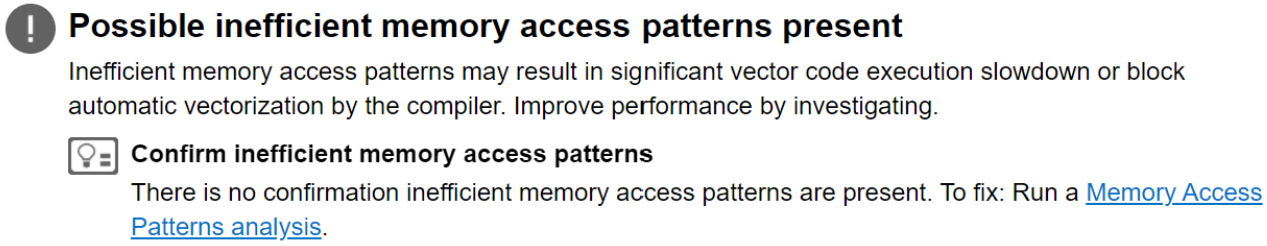
Можете попытаться, в соответствии с рекомендациями, поправить в CMakeLists.txt для примера 1_mmult используемое векторное расширение, результат зафиксировать.

В рамках этого разбора же попробуем собрать Memory Level Roofline.

Для этого необходимо в панели **Analysis Workflow**, в **Characterization** блоке выбрать пункт **Cache Simulation** и пересобрать отчет.

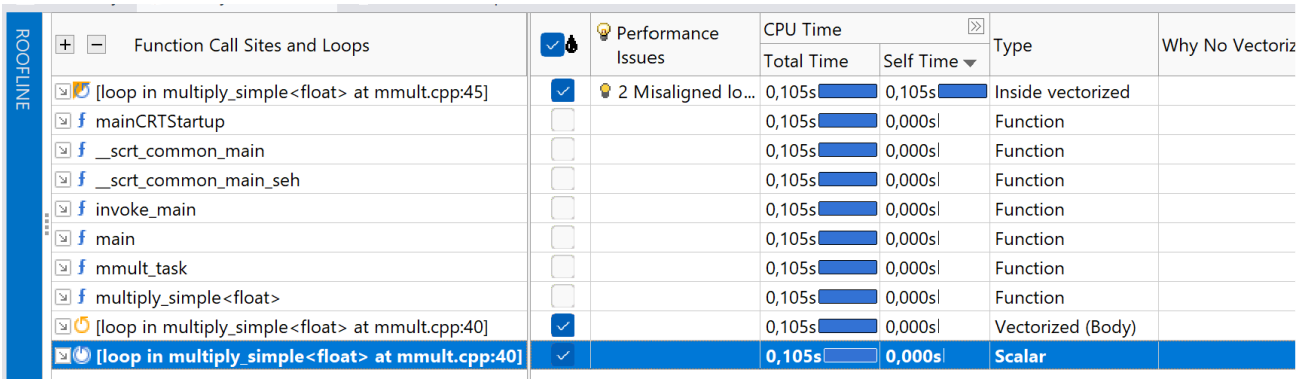


И теперь по двойному клику на точку на графике руфлайна раскроется раскладка для данного хотспота по утилизации различных уровней подсистемы памяти.



А в рекомендациях появится новая опция-предложение: запустить MAP анализ, который позволит определить паттерны доступа к памяти в рамках хотспота.

Для этого, во-первых, необходимо выделить интересующие нас хотспоты:

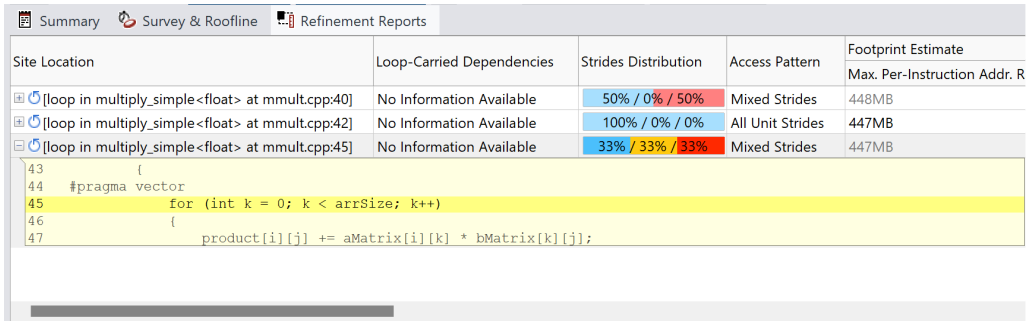


Function Call Sites and Loops	Performance Issues	CPU Time		Type	Why No Vectoriz
		Total Time	Self Time		
[loop in multiply_simple<float> at mmult.cpp:45]	2 Misaligned lo...	0,105s	0,105s	Inside vectorized	
mainCRTStartup		0,105s	0,000s	Function	
__scrt_common_main		0,105s	0,000s	Function	
__scrt_common_main_seh		0,105s	0,000s	Function	
invoke_main		0,105s	0,000s	Function	
main		0,105s	0,000s	Function	
mmult_task		0,105s	0,000s	Function	
multiply_simple<float>		0,105s	0,000s	Function	
[loop in multiply_simple<float> at mmult.cpp:40]		0,105s	0,000s	Vectorized (Body)	
[loop in multiply_simple<float> at mmult.cpp:40]		**0,105s**	**0,000s**	**Scalar**	

И выбрать в панели **Analysis Workflow** новый анализ - **Memory Access Patterns** и запустить его.

Третья вкладка **Refinement Reports**:

Тут можно найти отчет по проведенному MAP анализу:

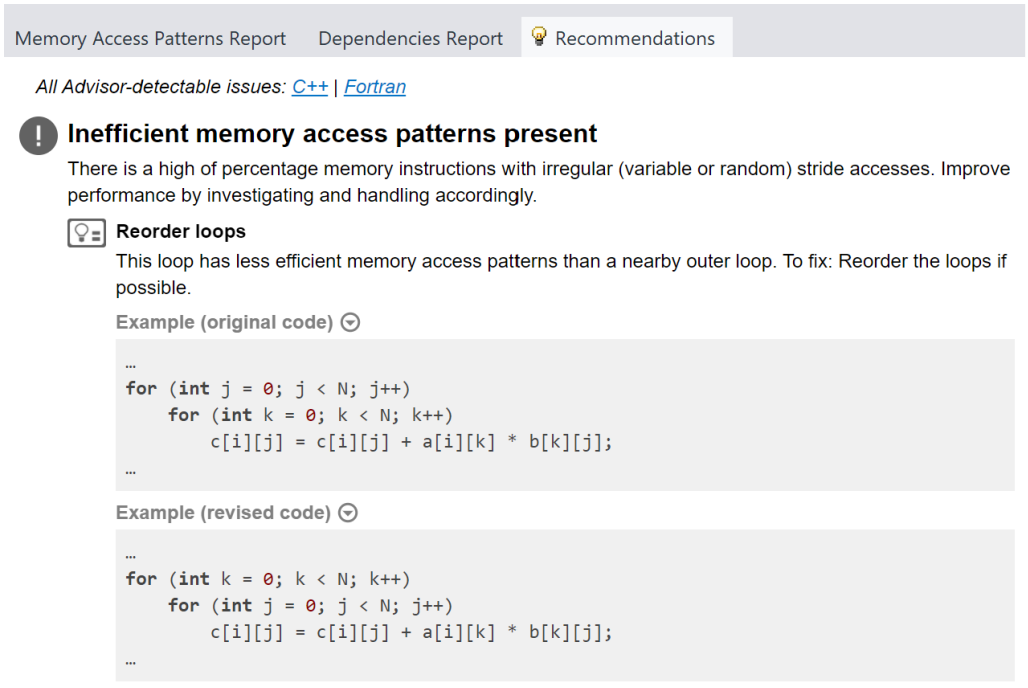


Summary | Survey & Roofline | Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate
				Max. Per-Instruction Addr. R
[loop in multiply_simple<float> at mmult.cpp:40]	No Information Available	50% / 0% / 50%	Mixed Strides	448MB
[loop in multiply_simple<float> at mmult.cpp:42]	No Information Available	100% / 0% / 0%	All Unit Strides	447MB
[loop in multiply_simple<float> at mmult.cpp:45]	No Information Available	33% / 33% / 33%	Mixed Strides	447MB

```
43          {
44  #pragma vector
45              for (int k = 0; k < arrSize; k++)
46              {
47                  product[i][j] += aMatrix[i][k] * bMatrix[k][j];
```

Где, собственно, и написано, что в нашей программе, в хотспоте, обнаружен неэффективный паттерн доступа к памяти. А ниже приведены рокомендации, как от этого можно избавиться:



Memory Access Patterns Report | Dependencies Report | Recommendations

All Advisor-detectable issues: C++ | Fortran

Inefficient memory access patterns present

There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating and handling accordingly.

Reorder loops

This loop has less efficient memory access patterns than a nearby outer loop. To fix: Reorder the loops if possible.

Example (original code)

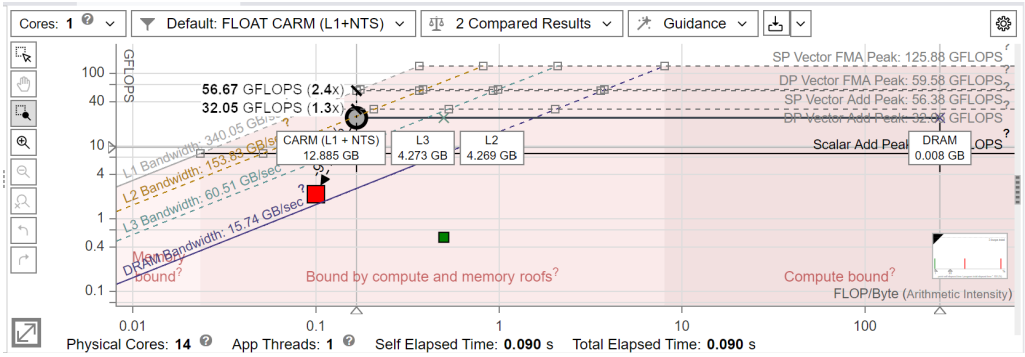
```
…
for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
…
```

Example (revised code)

```
…
for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
…
```

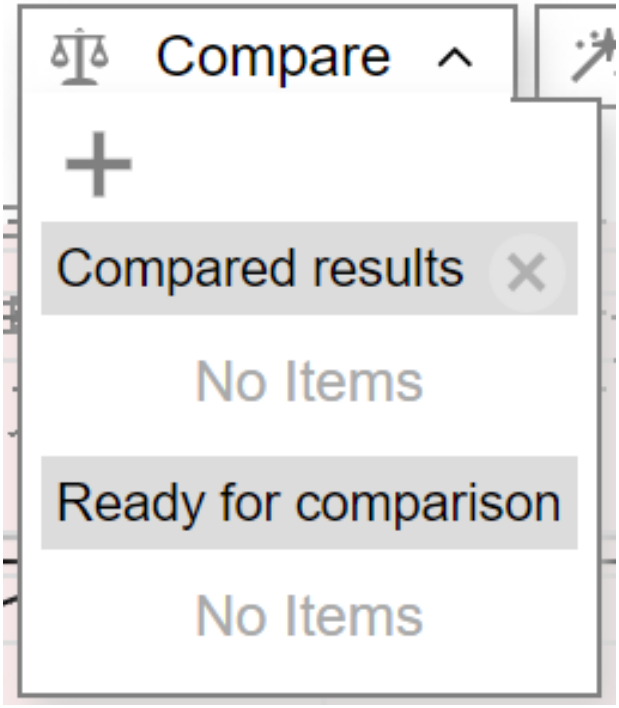
> Сохраняем снэпшот и применяем исправления в коде. Снэпшот приложить к отчету. Можно собрать все в архив и сохранить для дальнейшей отправки преподавателю.

Memory Level Roofline для новой версии приложения:

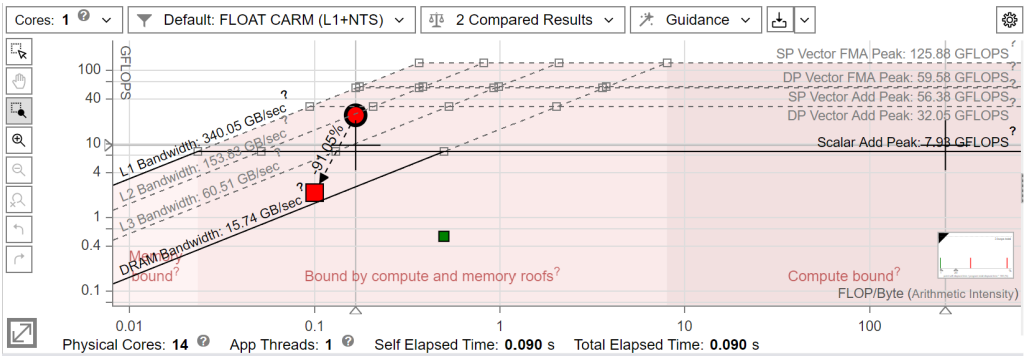


> Доп. вопрос: о чем на рисунке выше говорит подобное расположение точек для разных уровней памяти на графике руффлайна.

А чтобы можно было сравнить две версии отчета соотв. разным версиям приложения:



Добавляем сохраненный ранее снэпшот и получаем следующую картинку:



Наглядно видно прирост производительности.

Для самостоятельного разбора остается два анализа: Dependencies и Suitability

Они помогут понять, что вообще возможно распараллелить, и насколько это будет эффективно масштабироваться.