

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ





Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Использование инструментария для профилировки приложений: Intel VTune Profiler и Intel Advisor

Волокитин В.Д.
Кафедра МОСТ, ННГУ

Содержание

- ❑ Цель работы
- ❑ Возможности инструментария
 - Intel VTune Profiler
 - Intel Advisor
- ❑ Тестовая задача
- ❑ Оптимизация приложения шаг за шагом

ЦЕЛЬ РАБОТЫ

Цель работы

- ❑ Научиться использовать инструментарий для обнаружения проблем производительности в процессе пошаговой оптимизации кода

Задачи:

- ❑ Рассмотреть основные возможности Intel VTune Profiler
- ❑ Рассмотреть основные возможности Intel Advisor
- ❑ На примере тестовой задачи из области вычислительной физики выполнить оптимизацию кода с использованием инструментария

INTEL VTUNE PROFILER

Intel VTune Profiler

Основные возможности

- ❑ Основные возможности Intel VTune Amplifier (VTune Profiler):
 - Время работы программы:
 - **Hotspots** – простой анализ, измеряющий время работы каждой отдельной функции и некоторые дополнительные (настраиваемые) характеристики.
 - Задействование ресурсов вычислительной системы:
 - **Microarchitecture Exploration** – анализ, позволяющий понять узкое место при исполнении на конкретном процессоре.
 - **Memory Access** – анализ производительности подсистемы памяти
 - Параллелизм программы:
 - **Threading** – анализ параллелизма в программе
 - **HPC Performance Characterization** – сбор метрик производительности.
- ❑ Все режимы анализа, кроме Hotspots и Threading требуют прав администратора.

INTEL ADVISOR

❑ Основные возможности Intel Advisor:

– **Survey** анализ показывает:

- где наиболее выгодно векторизовать и распараллелить код
- причины замедления или отсутствия векторизации
- проблемы с производительностью кода в целом

– **Trip Counts and FLOP** анализ:

- динамически определяет количество вызовов функций и циклов
- считает количество операций с плавающей запятой и целыми числами, а также движение данных в подсистеме памяти

– **Roofline** анализ:

- позволяет визуализировать производительность приложения в зависимости от аппаратной архитектуры, используя оба предыдущих анализа
- позволяет определить ограничивающие факторы текущей реализации кода.

- **Dependencies** анализ: позволяет найти или исключить зависимость по данным в разных циклах
 - при отсутствии зависимости можно принудительно векторизовать цикл
 - при наличии зависимости можно пробовать изменить структуру данных
- **Memory Access Patterns (MAP)** анализ:
 - позволяет определить нерегулярный доступ к памяти
 - основная цель – сократить накладные расходы векторизации
- **Suitability** анализ:
 - позволяет определить вероятный выигрыш в производительности от параллелизма
 - позволяет определить потенциальные эффекты параллельных накладных расходов
 - позволяет определить влияние изменения количества итераций и продолжительности итерации на производительность выбранного цикла

ТЕСТОВАЯ ЗАДАЧА

Тестовая задача и метод решения

- Найти оператор интегрирования уравнения Шредингера:

$$\dot{A} = -iHA$$

где A, H – комплексные эрмитовы матрицы,
начальное условие уравнения $A(0) = E$ – единичная матрица.
Необходимо найти $A(T)$.

- Метод Рунге-Кутты 4-го порядка для данной задачи:

$$k_1 = -iHA(t)$$

$$k_2 = -iH\left(A(t) + \frac{h}{2}k_1\right)$$

$$k_3 = -iH\left(A(t) + \frac{h}{2}k_2\right)$$

$$k_4 = -iH\left(A(t) + hk_3\right)$$

$$A(t + h) = A(t) + \frac{h}{6}(k_1 + k_2 + k_3 + k_4)$$

Тестовая задача

Код программы (1)

// см. приложенный код

```
#include <complex>
#include <random>

typedef complex<double> dcomplex;
int main() {
    dcomplex *iH, *y;
    unsigned int size = 1000;
    unsigned int steps = 1;
    double h = 0.001 / (double)steps;
    chrono::time_point<chrono::system_clock> start, end;
    start = chrono::system_clock::now();

    iH = new dcomplex[size * size];
    y = new dcomplex[size * size];
    generationHamilton(iH, size);
    generationY0(y, size);
    RK4(iH, y, h, steps, size);
    end = chrono::system_clock::now();

    int elapsed_seconds =
        chrono::duration_cast<chrono::milliseconds>
        (end - start).count();
    cout <<"Total time: "<<elapsed_seconds/1000.0<<" sec"<<endl;
    return 0;
}
```

```
void generationHamilton(dcomplex * iH, unsigned int size) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> uniform_dis(1, 10.001);
    for(unsigned int i = 0; i < size; i++) {
        iH[i*size + i] = dcomplex(0.0, -uniform_dis(gen));
        for(unsigned int j = i + 1; j < size; j++) {
            double re = -uniform_dis(gen);
            double im = -uniform_dis(gen);
            iH[i*size + j] = dcomplex(re, im);
            iH[j*size + i] = dcomplex(-re, im);
        }
    }
}

void generationY0(dcomplex * y, unsigned int size) {
    memset(y, 0, size * size * sizeof(dcomplex));
    for(unsigned int i = 0; i < size; i++) {
        y[i*size + i] = dcomplex(1.0, 0.0);
    }
}
```

Тестовая задача

Код программы (2)

// см. приложенный код

```
void RK4(dcomplex * iH, dcomplex * y0, double h, unsigned int
steps, unsigned int size) {
    dcomplex * y1 = new dcomplex[size*size];
    dcomplex * y2 = new dcomplex[size*size];
    dcomplex * y3 = new dcomplex[size*size];
    dcomplex * y4 = new dcomplex[size*size];
    dcomplex * yTmpRes = new dcomplex[size*size];
    dcomplex * yRKstep = new dcomplex[size*size];
    for (int i = 0; i < steps; i++) {
        stepRK4(y0, y1, y2, y3, y4, yTmpRes, yRKstep, iH, h, size);
    }
    delete(y1);
    delete(y2);
    delete(y3);
    delete(y4);
    delete(yTmpRes);
    delete(yRKstep);
}
```

```
void stepRK4(dcomplex * y, dcomplex * y1, dcomplex * y2,
dcomplex * y3, dcomplex * y4, dcomplex * yTmpRes, dcomplex *
yRKstep, dcomplex * iH, double h, unsigned int size_system) {
    int size_system2 = size_system * size_system;
    Function(iH, y, yTmpRes, size_system);
    for(int i = 0; i < size_system2; i++) {
        y1[i] = h * yTmpRes[i];
        yRKstep[i] = (y[i] + y1[i]) / 2.0;
    }
    Function(iH, yRKstep, yTmpRes, size_system);
    for(int i = 0; i < size_system2; i++) {
        y2[i] = h * yTmpRes[i];
        yRKstep[i] = (y[i] + y2[i]) / 2.0;
    }
    Function(iH, yRKstep, yTmpRes, size_system);
    for(int i = 0; i < size_system2; i++) {
        y3[i] = h * yTmpRes[i];
        yRKstep[i] = (y[i] + y3[i]);
    }
    Function(iH, yRKstep, yTmpRes, size_system);
    for(int i = 0; i < size_system2; i++) {
        y4[i] = h * yTmpRes[i];
        y[i] += (y1[i] + 2.0 * y2[i] + 2.0 * y3[i] + y4[i])/6.0;
    }
}
```

Тестовая задача

Код программы (3)

// см. приложенный код

```
void Function(dcomplex * iH, dcomplex * y, dcomplex * res, unsigned int size_system) {
    matrix_mult(iH, y, res, size_system);
}

void matrix_mult(dcomplex * a, dcomplex * b, dcomplex * res, unsigned int size) {
    memset(res, 0, size*size * sizeof(dcomplex));

    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                res[i*size + j] += a[i*size + k] * b[k*size + j];
            }
        }
    }
}
```

АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ И ОПТИМИЗАЦИЯ ПРИЛОЖЕНИЯ

Intel VTune Profiler

Запуск профилировщика (1)

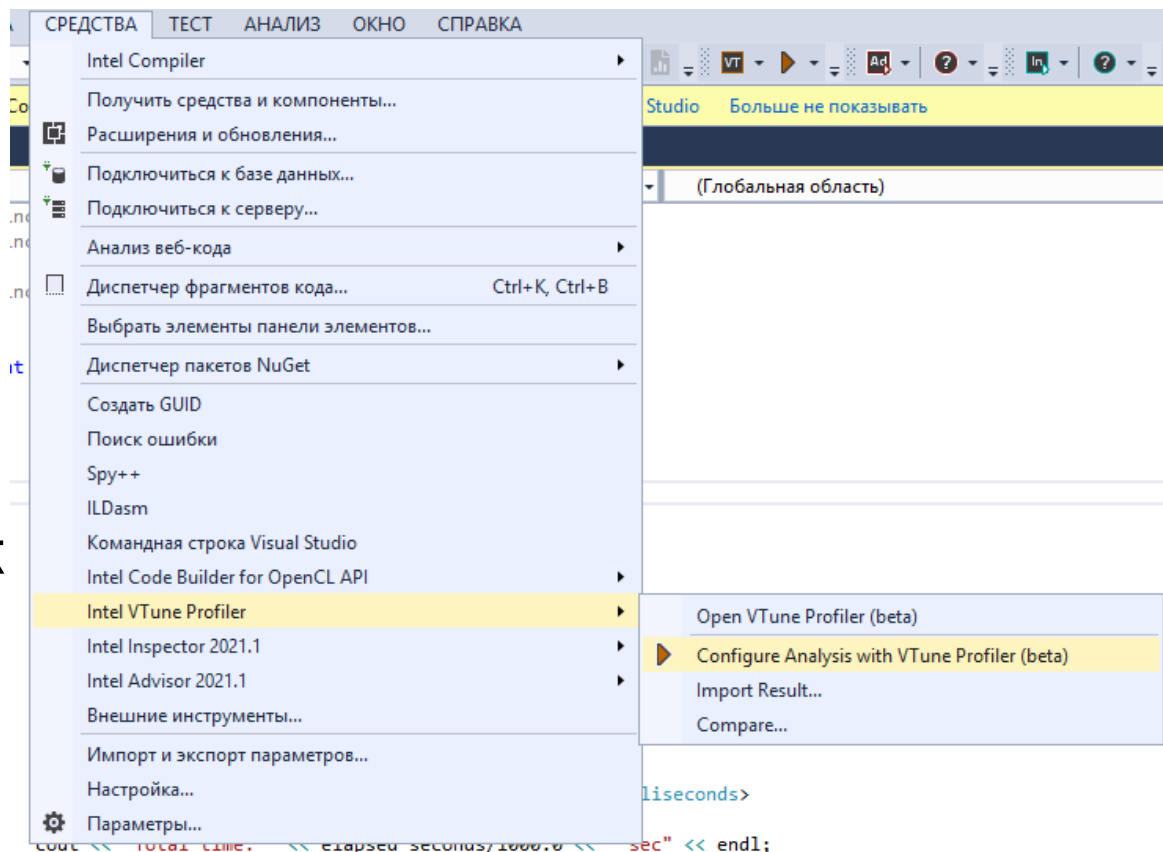
- ❑ Интеграция в Visual Studio:

Средства -> Intel VTune Profiler -> Configure Analysis

- ❑ Intel VTune Profiler может быть запущен как отдельное приложение

- ❑ Приложение должно быть скомпилировано с оптимизацией (Release)

- ❑ Для лучшего анализа приложение должно содержать отладочную информацию (Debug Info)



Intel VTune Profiler

Запуск профилировщика (2)

Выбор компьютера для запуска
(доступен запуск на другой машине)

Выбор типа анализа

Configure Analysis

WHERE

Local Host

WHAT

Launch Application

Specify and configure your analysis target: an application or a script to execute.

☒ Inherit settings from IDE project:

Application:
C:\Users\volokitin.v\source\repos\HPC2019\w64\Release\HPC2019.exe

Application parameters:

☐ Use application directory as working directory

Working directory:
C:\Users\volokitin.v\source\repos\HPC2019\HPC2019\

Advanced ▶

HOW

Hotspots

Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. [Learn more](#)

☒ User-Mode Sampling ?
☐ Hardware Event-Based Sampling ?

☒ Show additional performance insights

Overhead

Details

▶

▶

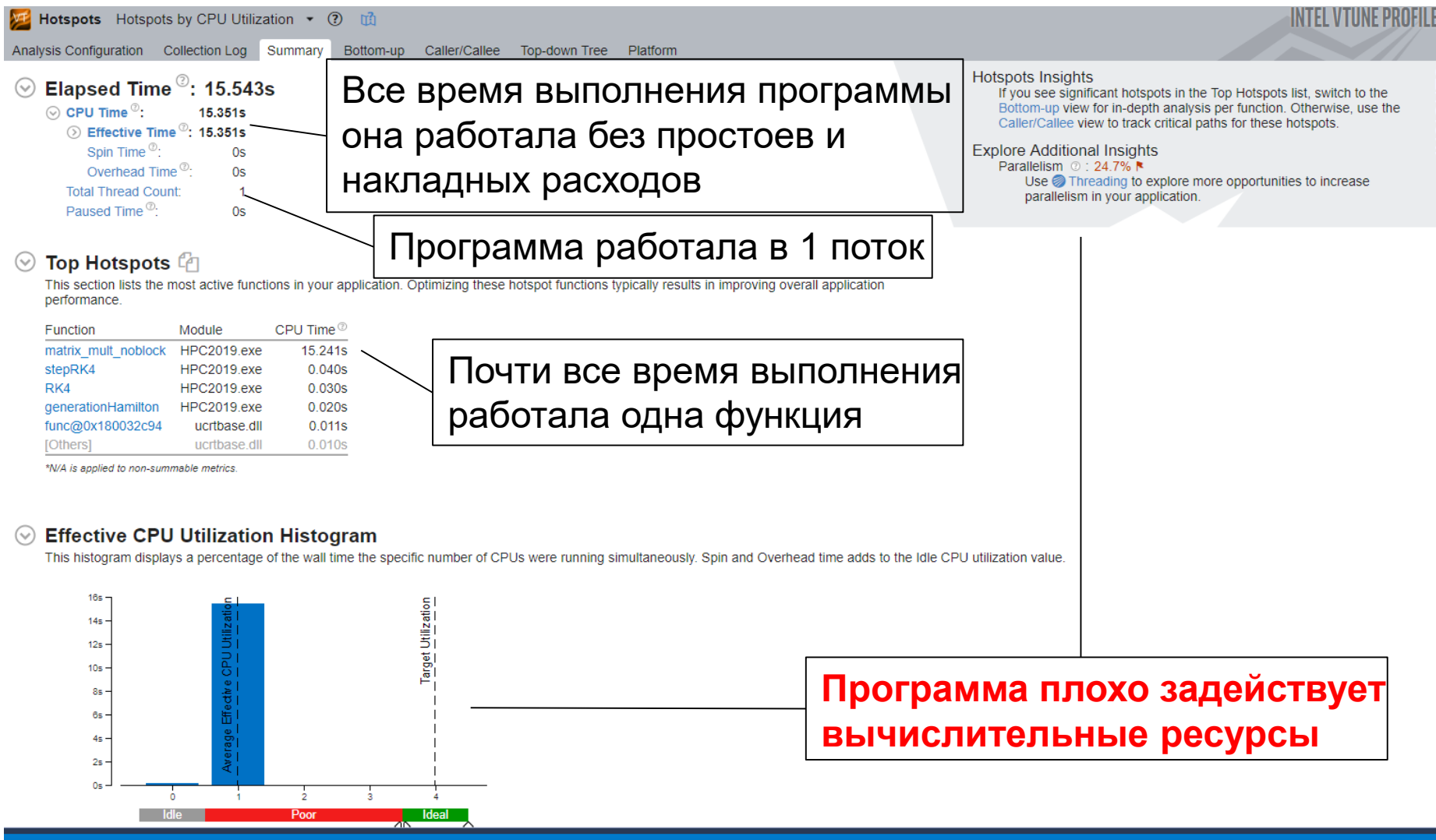
Настройки запуска приложения

Детальное описание анализа

Запуск настроенного анализа

Intel VTune Profiler

Результаты профилировки (1)



Оптимизация

Проведение оптимизации кода (1)

❑ Использование Intel Compiler:

- Свойства проекта -> Общие -> Platform Toolset -> Intel C++ Compiler 19.0

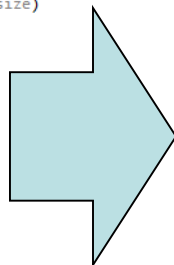
❑ Включение поддержки OpenMP:

- Свойства проекта -> C/C++ -> Language(Intel C++) -> OpenMP Support -> /Qopenmp

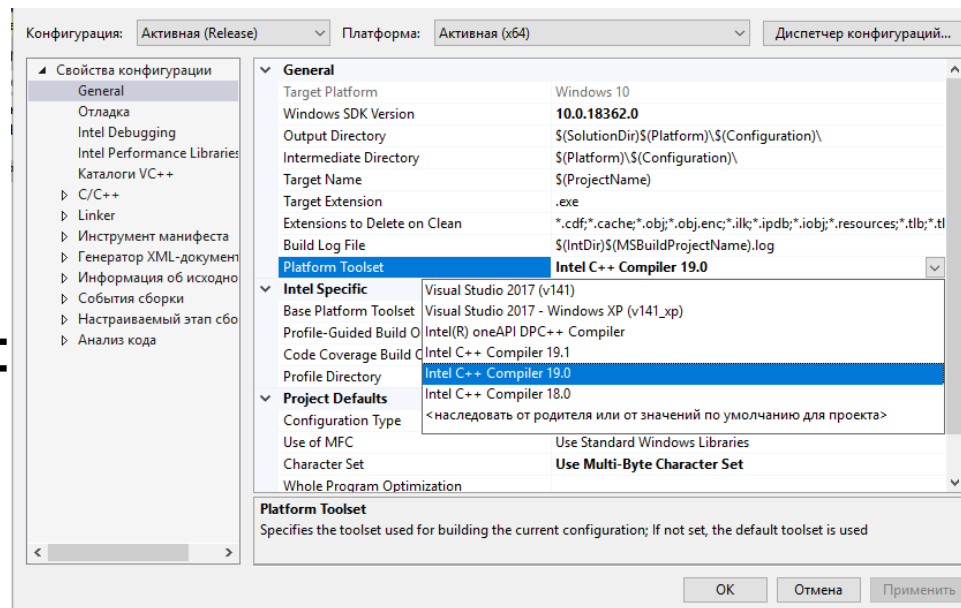
❑ Распараллеливание цикла:

```
void matrix_mult_noblock(dcomplex * a, dcomplex * b, dcomplex * res, unsigned int size)
{
    memset(res, 0, size*size * sizeof(dcomplex));

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                res[i*size + j] += a[i*size + k] * b[k*size + j];
            }
        }
    }
}
```



```
void matrix_mult_noblock(dcomplex * a, dcomplex * b, dcomplex * res, unsigned int size)
{
    memset(res, 0, size*size * sizeof(dcomplex));
    #pragma omp parallel for
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                res[i*size + j] += a[i*size + k] * b[k*size + j];
            }
        }
    }
}
```



Intel VTune Profiler

Результаты профилировки (2)

Hotspots Hotspots by CPU Utilization ?

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform

Elapsed Time: 8.482s

CPU Time: 30.997s
Effective Time: 30.051s
Spin Time: 0.737s
Overhead Time: 0.209s
Total Thread Count: 4
Paused Time: 0s

Появилось время простоя
Появились накладные расходы

Программа работала в 4 потока

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee view to track critical paths for these hotspots.

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
matrix_mult\$omp\$parallel_for@13	HPC2019.exe	30.006s
__kmp_fork_barrier	libiomp5md.dll	0.594s
__kmp_get_global_thread_id_reg	libiomp5md.dll	0.209s
__kmp_join_call	libiomp5md.dll	0.143s
RK4	HPC2019.exe	0.020s
[Others]		0.025s

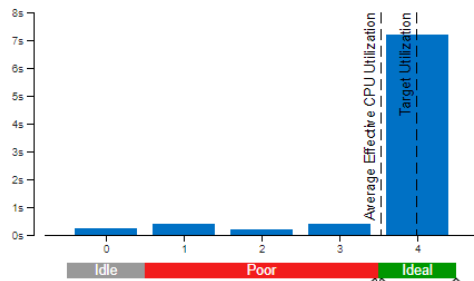
*N/A is applied to non-summable metrics.

Почти все время выполнения работала одна функция

Общее ускорение в 1.8 раз
(15.5 секунд против 8.5 секунд)

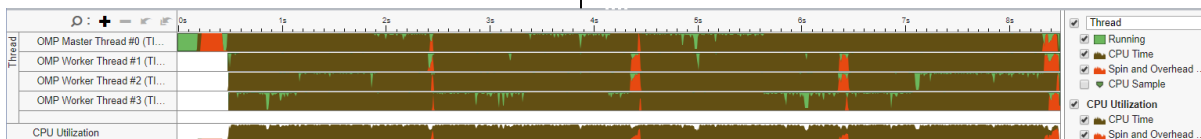
Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Программа эффективно использует ресурсы ЦП

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform



Intel Advisor

Запуск Intel Advisor

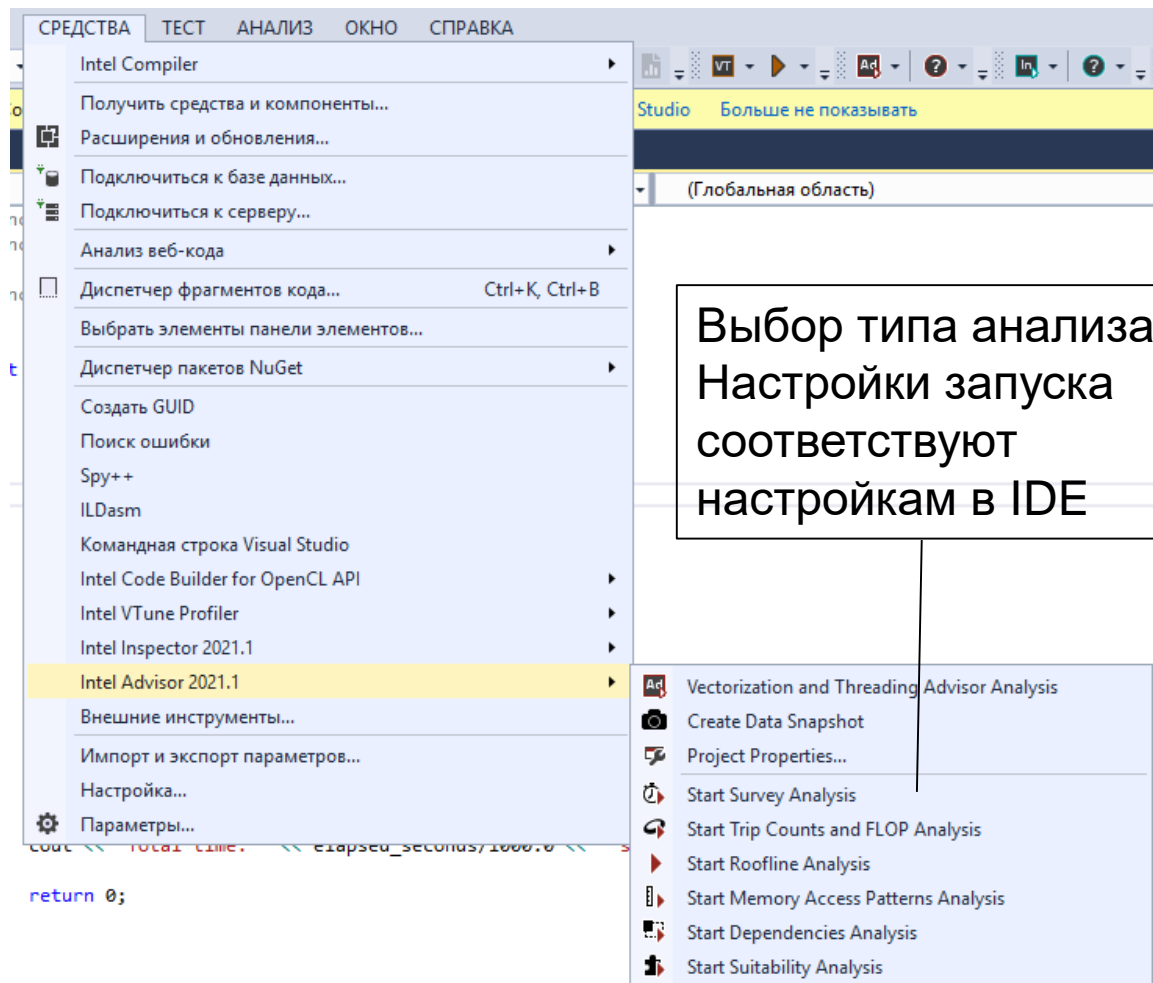
- ❑ Интеграция в Visual Studio:

Средства -> Intel Advisor 2021.1

- ❑ Intel Advisor может быть запущен как отдельное приложение

- ❑ Приложение должно быть скомпилирована с оптимизацией (Release)

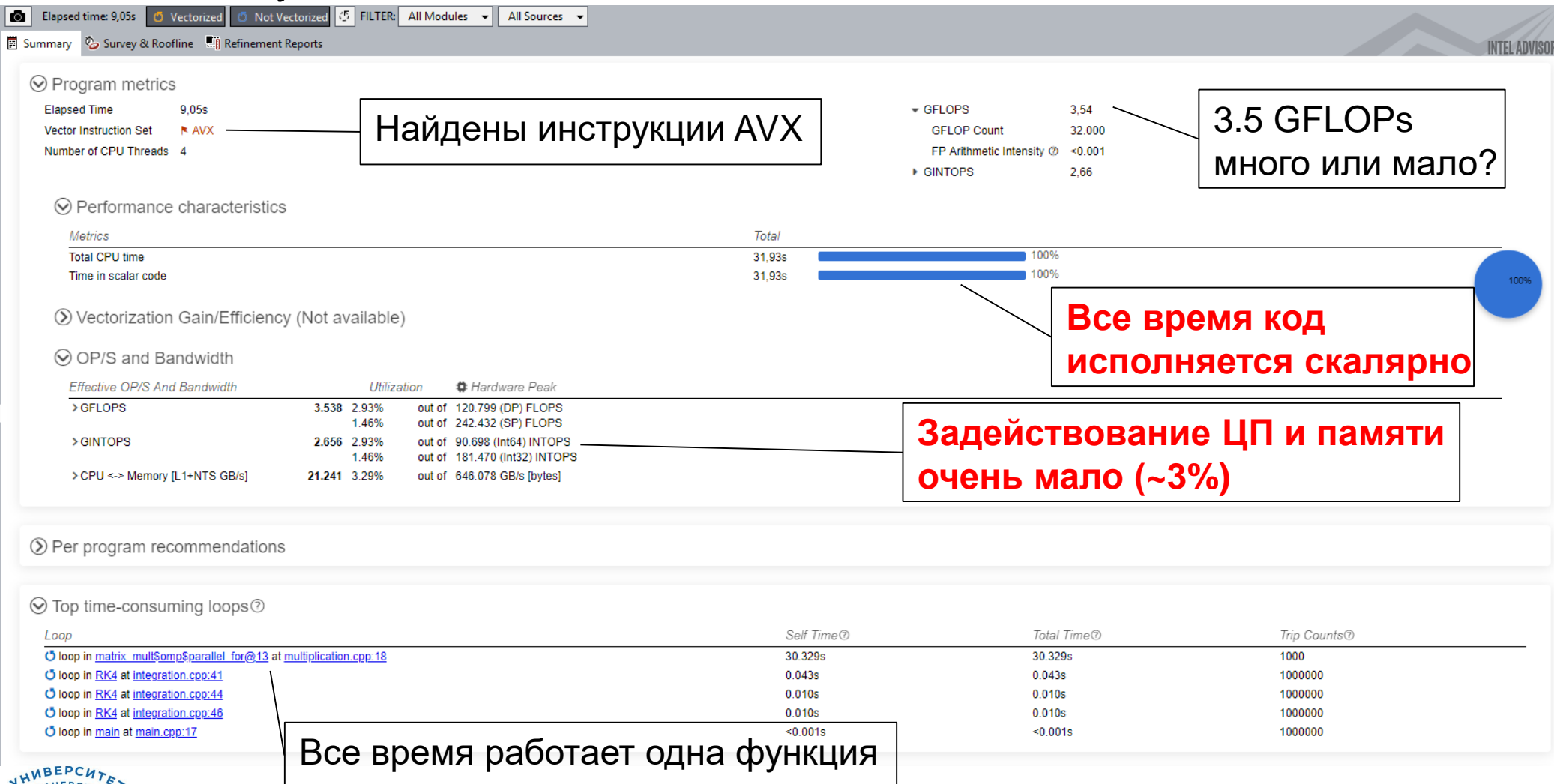
- ❑ Для лучшего анализа приложение должно содержать отладочную информацию (Debug Info)



Intel Advisor

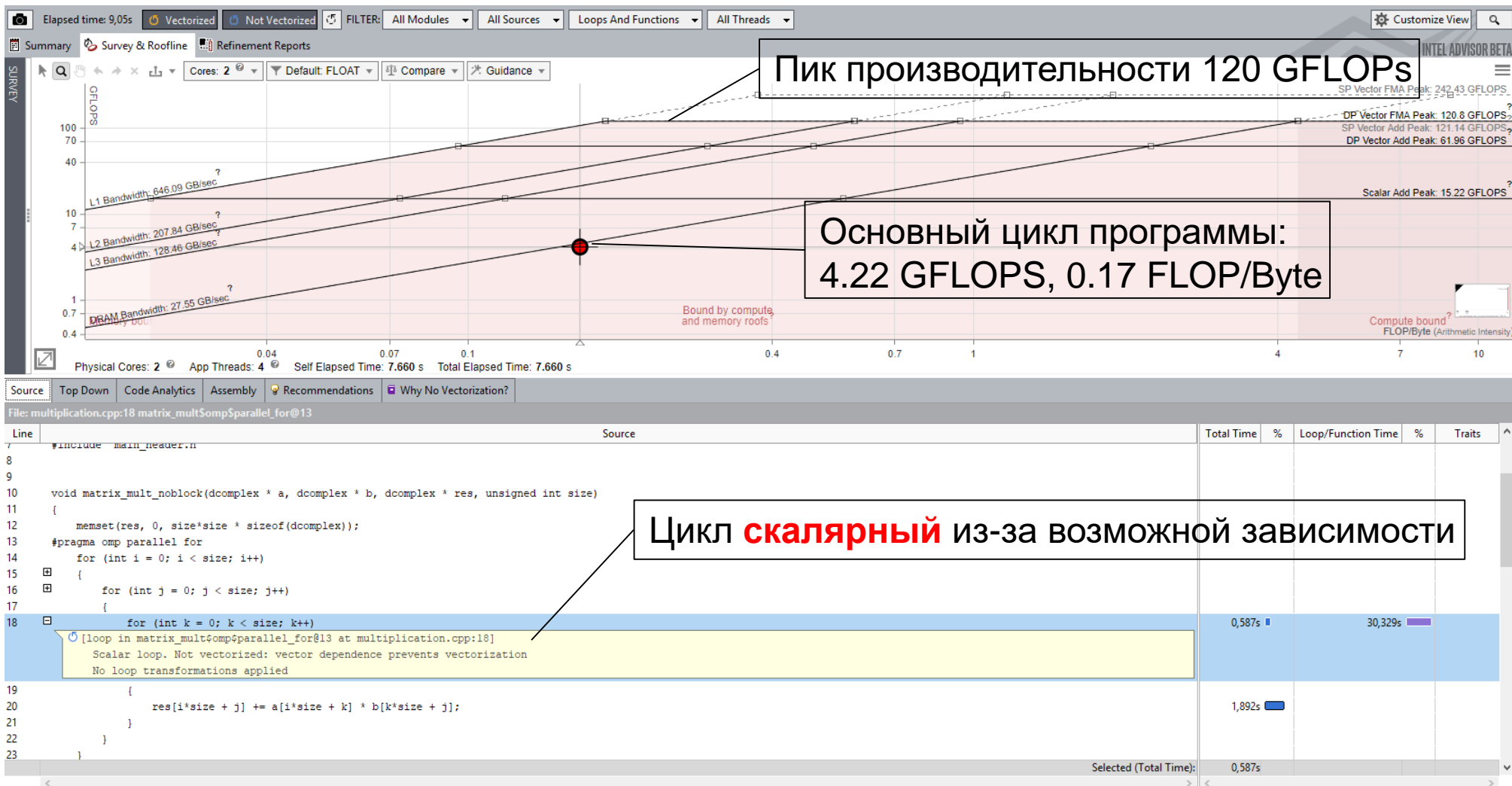
Результаты Roofline анализа (1)

Summary



Intel Advisor

Результаты Roofline анализа (2)



Оптимизация

Векторизация

❑ Директивы компилятора:

– **#pragma ivdep + #pragma vector always**

- Компилятор все равно анализирует зависимости
- `ivdep` – недоказанные зависимости необходимо проигнорировать
- `vector always` – векторизовать, даже если векторизация не эффективна

– **#pragma simd**

- Векторизовать цикл в любом случае, даже при наличии доказанной зависимости по данным

❑ OpenMP 4:

– **#pragma omp simd**

- Аналогично `#pragma ivdep + #pragma vector always`

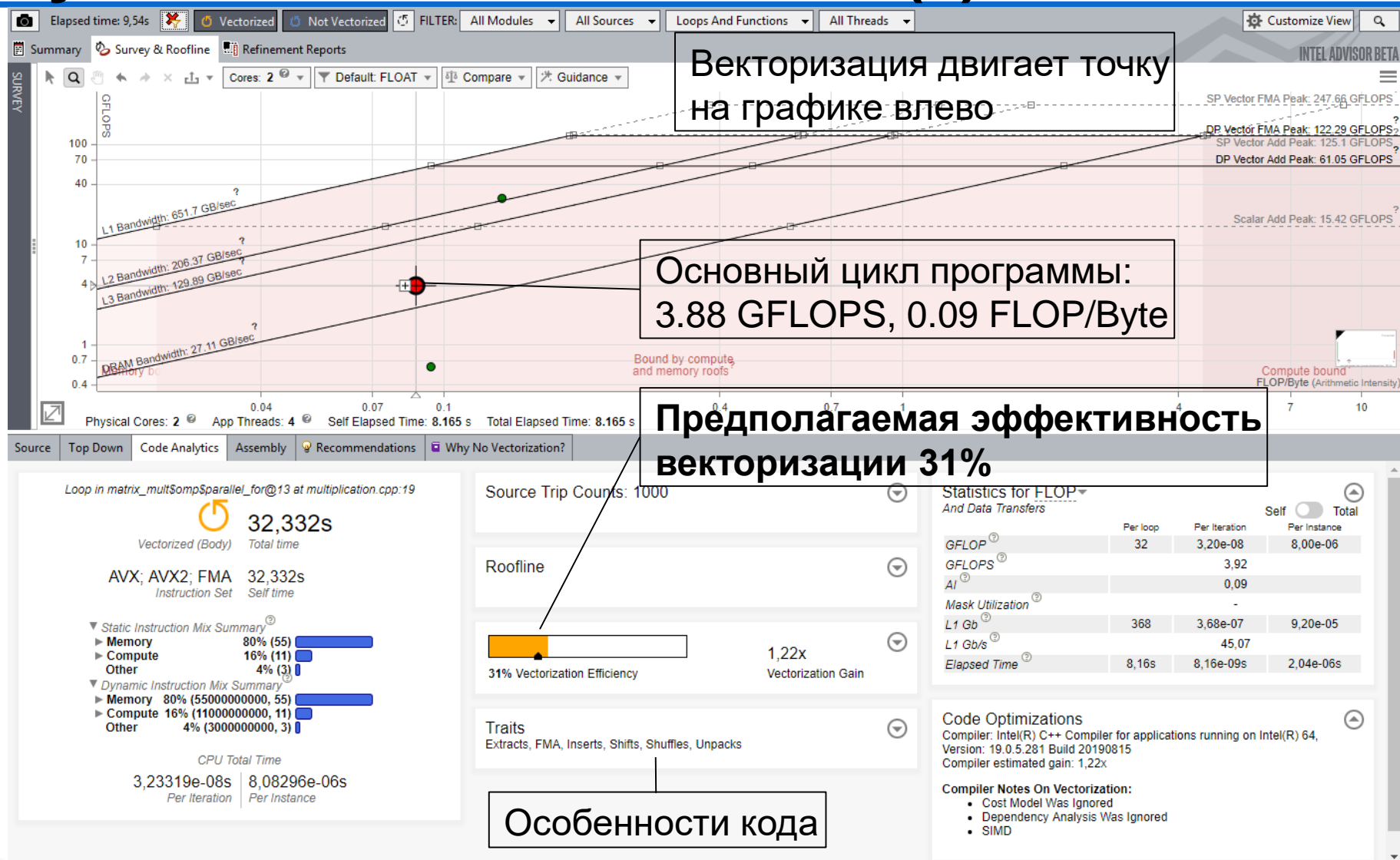
❑ C99:

– Ключевое слово **restrict (double* restrict a)**

- Массивы, объявленные через указатель на первый элемент, с ключевым словом `restrict`, не пересекаются

Intel Advisor

Результаты Roofline анализа (3)



Intel Advisor

Результаты MAP анализа

Elapsed time: 9,54s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate			Site Name	Performance Issues
				Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memo...		
[loop in matrix_multSomp\$parallel_for@13 at multiplicatio ...	No Information Available	90% / 10% / 0%	Mixed Strides	14MB	14MB	0B	loop_...	1 Inefficient memory access patterns present

```

17 {
18 #pragma omp simd
19 for (int k = 0; k < size; k++)
20 {
21     res[i*size + j] += a[i*size + k] * b[k*size + j];

```

90% - обращений в ту же память и последовательных обращений в память

10% - обращений в память с фиксированным шагом

0% - нерегулярных обращений

90%: percentage of memory instructions with unit stride or stride 0 accesses

Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration

Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration

10%: percentage of memory instructions with fixed or constant non-unit stride accesses

Constant stride (stride N) = Instruction accesses memory that consistently changes by N elements from iteration to iteration

Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration

0%: percentage of memory instructions with irregular (variable or random) stride accesses

Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration

Typically observed for indirect indexed array accesses, for example, a[index[i]]

- gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

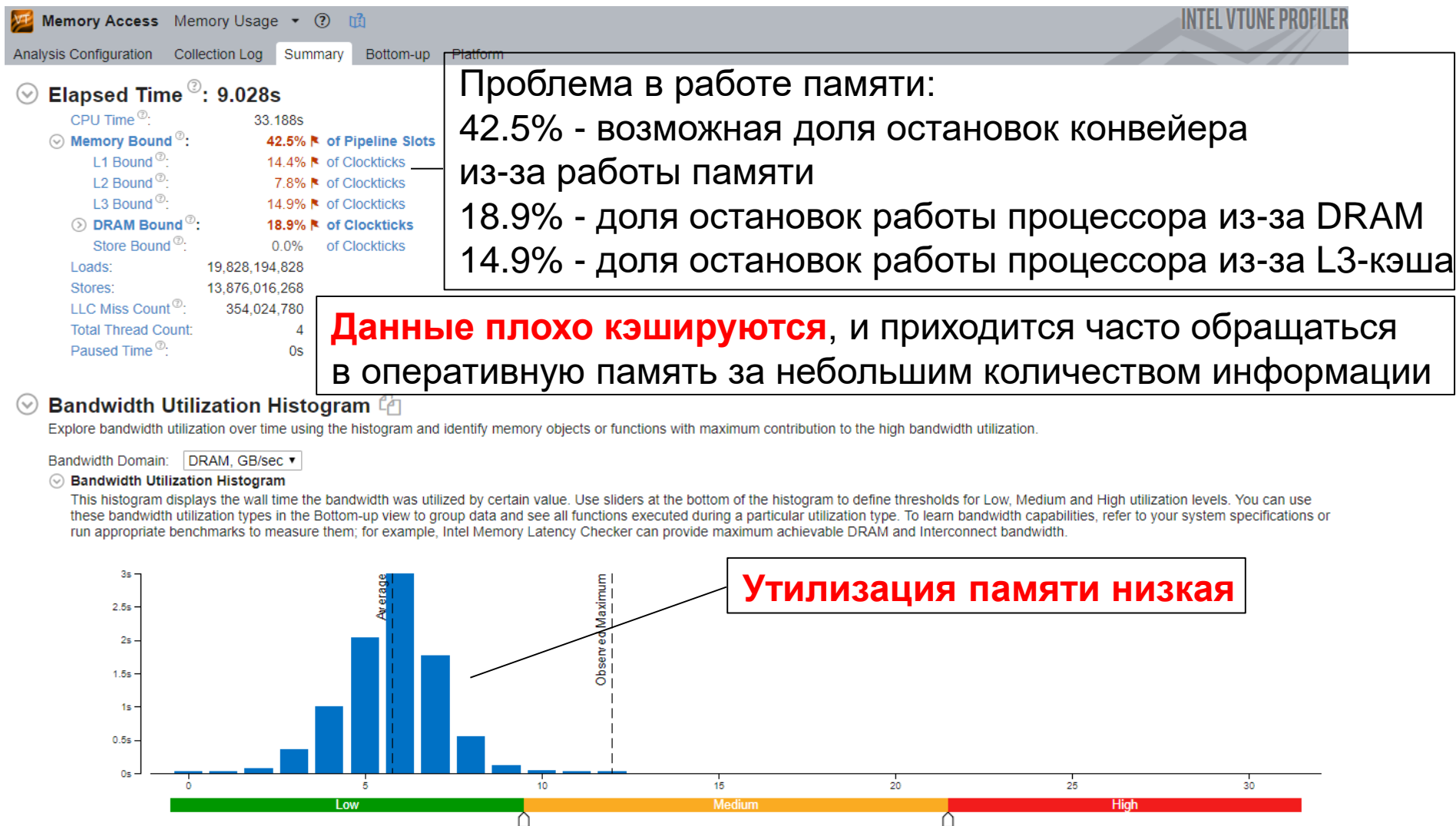
- scatter (irregular) accesses, detected for v(p)scatter* instructions on AVX2 Instruction Set Architecture

Memory Access Patterns Report			Dependencies Report		Recommendations					
ID	Icon	Stride	Type	Source	Nested Function	Variable references	Max. Per-Instruction Addr. Range	Mo.	Site Name	Access Type
P1		1; 8	Unit stride	complex:1292		block 0x180272e5040 a	15KB		hpc. loop_site_23	Read
P2		4000	Constant stride	complex:638		block 0x18028252040 a	14MB		hpc. loop_site_23	Read
P3			Parallel site information	multiplication.cpp:19					hpc. loop_site_23	
P5		0	Uniform stride	complex:1292			16B		hpc. loop_site_23	Write
P6		0	Uniform stride	complex:617			8B		hpc. loop_site_23	Read
P7		0	Uniform stride	complex:624		block 0x1802d20b040 a	8B		hpc. loop_site_23	Write
P8		0	Uniform stride	complex:625		block 0x1802d20b040 a	8B		hpc. loop_site_23	Write
P9		0	Uniform stride	complex:641			8B		hpc. loop_site_23	Read
P10		0	Uniform stride	complex:642			8B		hpc. loop_site_23	Read
P11		0	Uniform stride	complex:643			8B		hpc. loop_site_23	Write
P12		0	Uniform stride	complex:645			8B		hpc. loop_site_23	Write

Максимальный шаг по памяти 14 MB

Intel VTune Profiler

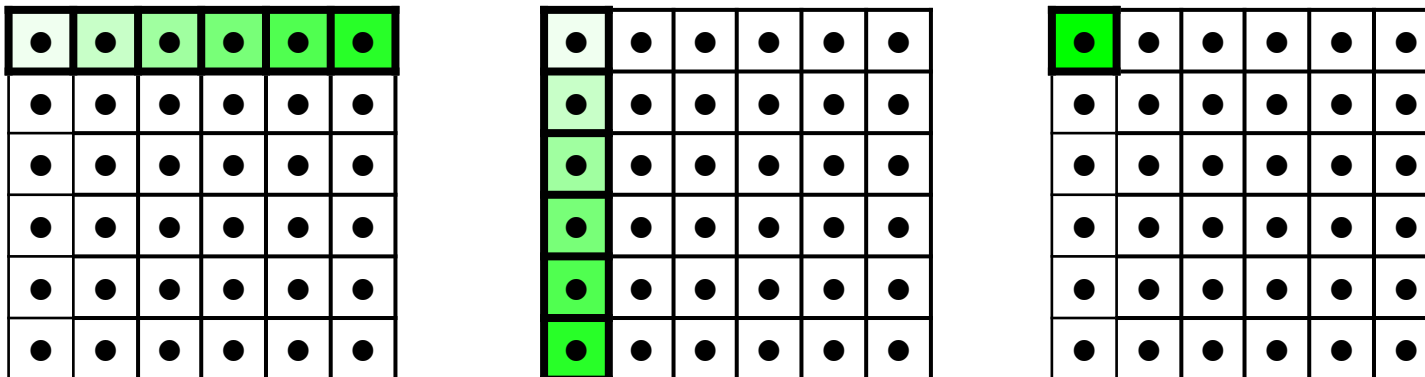
Результаты Memory Access анализа



Оптимизация

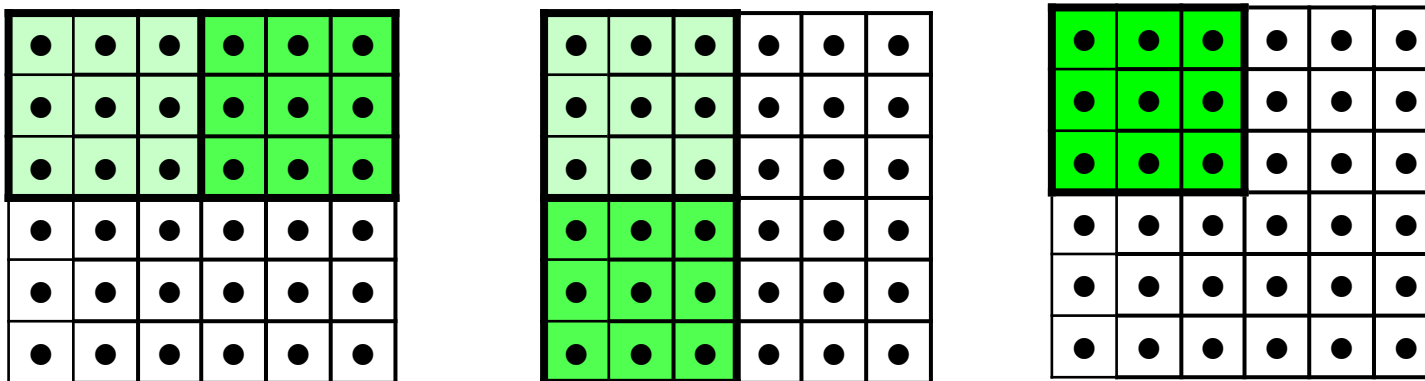
Проведение оптимизации кода (2)

- Стандартная схема вычислений матричного умножения:



Кэш обновляется каждую итерацию при большом размере матриц

- Блочная схема вычислений матричного умножения:



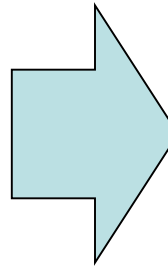
Кэш обновляется каждый раз при смене блоков и загружает новые блоки вместе

Оптимизация

Проведение оптимизации кода (3)

❑ Блочная схема вычисления для кэша L3

```
void matrix_mult_noblock(dcomplex * a, dcomplex * b, dcomplex * res, unsigned int size)
{
    memset(res, 0, size*size * sizeof(dcomplex));
    #pragma omp parallel for
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            #pragma omp simd
            for (int k = 0; k < size; k++)
            {
                res[i*size + j] += a[i*size + k] * b[k*size + j];
            }
        }
    }
}
```



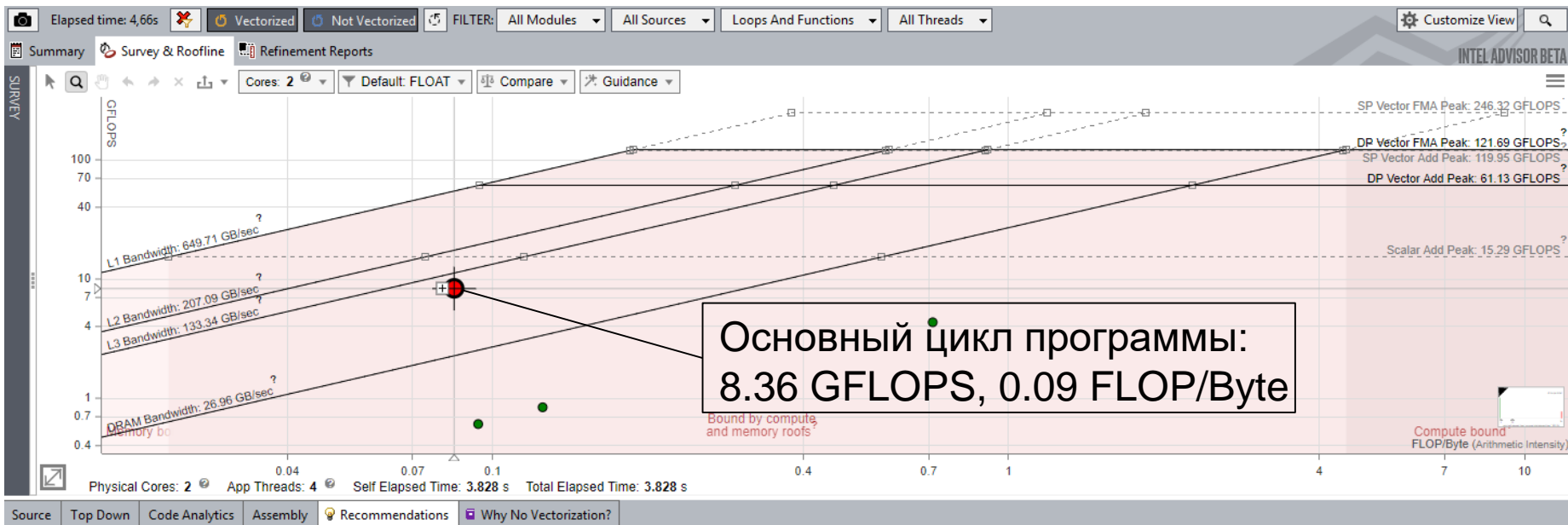
```
void matrix_mult_block(dcomplex * a, dcomplex * b, dcomplex * res, unsigned int size)
{
    int size_block = 64;
    int isize = size;
    memset(res, 0, size*size * sizeof(dcomplex));

    #pragma omp parallel for
    for (int ib = 0; ib < isize; ib += size_block)
    {
        for (int jb = 0; jb < isize; jb += size_block)
        {
            for (int kb = 0; kb < isize; kb += size_block)
            {
                int iEnd = min(isize, ib + size_block);
                int jEnd = min(isize, jb + size_block);
                int kEnd = min(isize, kb + size_block);
                for (int i = ib; i < iEnd; i++)
                {
                    for (int j = jb; j < jEnd; j++)
                    {
                        #pragma omp simd
                        for (int k = kb; k < kEnd; k++)
                        {
                            res[i*size + j] += a[i*size + k] * b[k*size + j];
                        }
                    }
                }
            }
        }
    }
}
```

Размер блока можно определить, исходя из параметров кэша

Intel Advisor

Результаты Roofline анализа (4)



All Advisor-detectable issues: [C++](#) | [Fortran](#)

! Inefficient memory access patterns present

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

🔍 Use SoA instead of AoS

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize data using SoA instead of AoS.

🔍 Use Intel SDLT

The cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines.

Example (original code) ☹

```
...  
struct kValues {  
    float Kx;  
    float Ky;  
    float Kz;  
    float PhiMag;  
};  
std::vector<kValues> dataset(count);  
...
```

Неэффективный доступ к памяти,
необходимо использовать другую структуру.
Связанно с хранением комплексных чисел

⬆ Inefficient memory access patterns present

Use SoA instead of AoS

Use Intel SDLT

Roofline conclusions

This loop is mostly memory bound

Intel Advisor

Результаты MAP анализа (2)

Elapsed time: 4,66s
 Vectorized
 Not Vectorized

FILTER: All Modules All Sources

Summary

Survey & Roofline

Refinement Reports

INTEL ADVISOR BETA

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate			Site Name	Performance
				Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint		
<div> [loop in matrix_multSomp\$parallel_for@34 at multiplication.cpp...] </div> <div> <pre> 47 { 48 #pragma omp simd 49 for (int k = kb; k < kEnd; k++) 50 { 51 res[i*isize + j] += a[i*isize + k] * b[k*isize + j]; </pre> </div>	No Information Available	83% / 17% / 0%	Mixed Strides	813KB	860KB	0B	loop_site_7	1 In

Изменение доли обращений между последовательным доступом и доступом с фиксированным шагом из-за появления смены блоков между итерациями

Максимальный шаг по памяти 813 KB

Memory Access Patterns Report Dependencies Report Recommendations										
ID	Icon	Stride	Type	Source	Nested Function	Variable references	Max. Per-Instruction Addr. Range	Modules	Site Name	Access Type
P1		4; 8	Constant stride	complex:1292		block 0x231458db040 allocated at main.cpp	848B	hpc2019.exe	loop_site_7	Read
P2		4000	Constant stride	complex:638		block 0x23146847040 allocated at main.cpp	813KB	hpc2019.exe	loop_site_7	Read
P3			Parallel site information	multiplication.cpp:49				hpc2019.exe	loop_site_7	
P5		0	Uniform stride	complex:1292			16B	hpc2019.exe	loop_site_7	Write
P6		0	Uniform stride	complex:617			8B	hpc2019.exe	loop_site_7	Read
P7		0	Uniform stride	complex:624		block 0x2314b7f6040 allocated at integratio	8B	hpc2019.exe	loop_site_7	Write
P8		0	Uniform stride	complex:625		block 0x2314b7f6040 allocated at integratio	8B	hpc2019.exe	loop_site_7	Write
P9		0	Uniform stride	complex:638			8B	hpc2019.exe	loop_site_7	Read
P10		0	Uniform stride	complex:641			8B	hpc2019.exe	loop_site_7	Read
P11		0	Uniform stride	complex:642			8B	hpc2019.exe	loop_site_7	Read
P12		0	Uniform stride	complex:643			8B	hpc2019.exe	loop_site_7	Write
P13		0	Uniform stride	complex:645			8B	hpc2019.exe	loop_site_7	Write
P14		0	Uniform stride	multiplication.cpp:49			4B	hpc2019.exe	loop_site_7	Read

Intel VTune Profiler

Результаты Memory Access анализа (2)

Memory Access Memory Usage ?
Analysis Configuration Collection Log Summary Bottom-up Platform

INTEL VTUNE PROFILER

Elapsed Time: 4.060s

CPU Time	15.048s	
Memory Bound	7.1%	of Pipeline Slots
L1 Bound	13.3%	of Clockticks
L2 Bound	2.7%	of Clockticks
L3 Bound	0.4%	of Clockticks
DRAM Bound	1.0%	of Clockticks
Store Bound	0.1%	of Clockticks
Loads	21,111,033,312	
Stores	13,528,005,828	
LLC Miss Count	3,600,252	
Total Thread Count	4	
Paused Time	0s	

Общее уменьшение проблем работы с памятью
Почти отсутствуют неэффективные обращения
в DRAM и кэш L3

Программа хорошо кэшируется и эффективно
работает с памятью на уровне DRAM и L3.
Ускорение по сравнению с первоначальной
версией ~3.9 раза

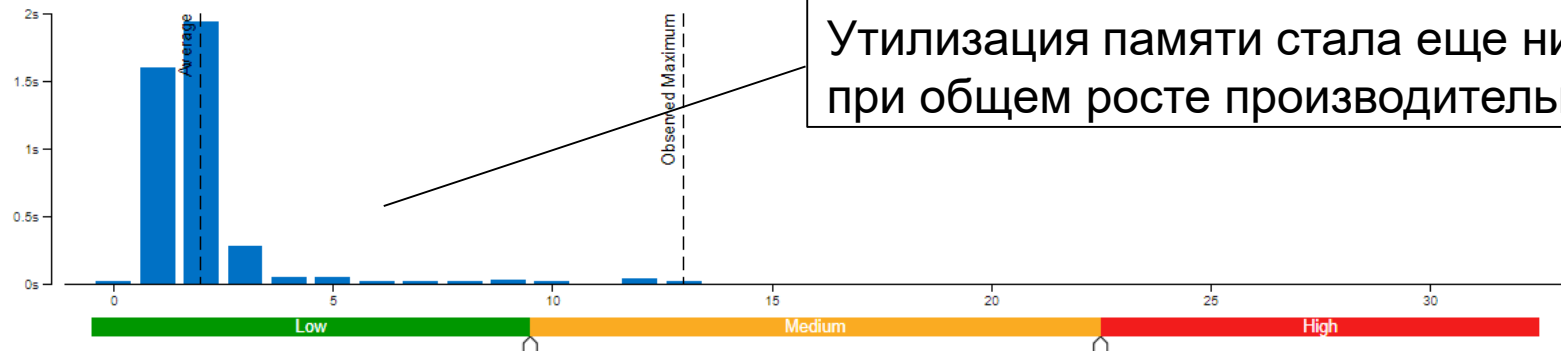
Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain: DRAM, GB/sec

Bandwidth Utilization Histogram

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.



Утилизация памяти стала еще ниже,
при общем росте производительности

Литература

1. Intel VTune Profiler: <https://software.intel.com/en-us/vtune>
2. Intel Advisor: <https://software.intel.com/en-us/advisor>
3. Williams S., Waterman A., Patterson D. Roofline: an insightful visual performance model for multicore architectures //Communications of the ACM. – 2009. – Т. 52. – №. 4. – С. 65-76.

Контакты

Нижегородский государственный университет

<http://www.unn.ru>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>

Волокитин В.Д.

valentin.d.volokitin@gmail.com