Эффективное программирование современных микропроцессоров и мультипроцессоров

Задание 1

Выполнил:

Пирожков Андрей

19210

Преподаватель:

Константин Викторович

Задание

1. Разработать программу в соответствии с одним из вариантов. Проверить правильность её работы.
2. Выполнить базовую оптимизацию программы (вручную и с помощью компилятора) с целью минимизации времени её работы. На каждом этапе оптимизации измерять время работы программы. Отмечать в отчёте успешные и неуспешные попытки оптимизации. Для оценки времени работы программы использовать одни и те же параметры: NX = NY ≈ 8000 – 10000, NT ≈ 100 – 120.
3. Для наиболее быстро работающего варианта программы с помощью средств профилирования выполнить следующие действия:
   1. Построить граф вызовов программы (картинку), определить «горячие точки» программы с точностью до функций.
   2. Построить аннотированный листинг программы, определить «горячие точки» программы с точностью до строк исходного кода и машинных команд.
   3. Собирая информацию о соответствующих событиях, получить с помощью профилирования следующие характеристики исполнения программы в целом:
      1. среднее число тактов на микрооперацию (или микроопераций на такт),
      2. процент кэш-промахов для кэшей первого и последнего уровней,
      3. процент неправильно предсказанных переходов.
   4. Сделать предположение о том, что является основной причиной временных затрат (вычислительные операции, обращения в память, выполнение команд перехода, …).
4. На roofline-модели отметить точку, соответствующую программе. Требуемые характеристики оценить исходя из анализа кода в «горячей точке» программы.

Все тесты программы выполнялись на устройстве Asus ux481fl на процессоре Intel Core i7 10510U

Результаты тестирования программ

* Самый упрощённый код (lab1\_-2.c).

Здесь осуществилась автоматическая векторизация программы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gcc | -O0 | -O1 | -O2 | -O3 | -Ofast |
| -g | 132.99 | 19.91 | 19.67 | 19.21 | 13.14 |
| -g  -march=native | 130.64 | 22.66 | 19.49 | 16.46 | 10.51 |

* Пересчитаны все индексы (lab1\_-1.c).

Теперь программа не может автоматически векторизовать программу

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gcc | -O0 | -O1 | -O2 | -O3 | -Ofast |
| -g | 137.21 | 33.25 | 32.63 | 33.53 | 32.25 |
| -g  -march=native | 136.74 | 32.86 | 27.3 | 28.61 | 29.91 |

* Пересчитаны константы (lab1.c).

В этом случае вынесены константы за общий цикл для избегания их пересчета

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gcc | -O0 | -O1 | -O2 | -O3 | -Ofast |
| -g | 133.81 | 33.24 | 32.62 | 33.64 | 32.23 |
| -g  -march=native | 133.81 | 32.81 | 27.28 | 28.88 | 30.15 |

* Константы считаются единожды (lab1\_1.c)

Тут только один раз за программу высчитываются константы и используются в программе

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gcc | -O0 | -O1 | -O2 | -O3 | -Ofast |
| -g | 133.88 | 33.06 | 32.71 | 33.58 | 32.21 |
| -g  -march=native | 133.66 | 33.36 | 27.3 | 28.56 | 30.05 |

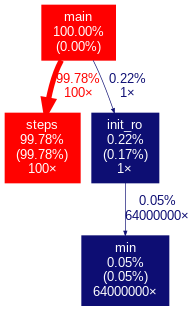
* Массив ro вынесен как константный массив (lab1\_1\_ro.c)

Теперь массив с ro высчитывается только один раз за программу

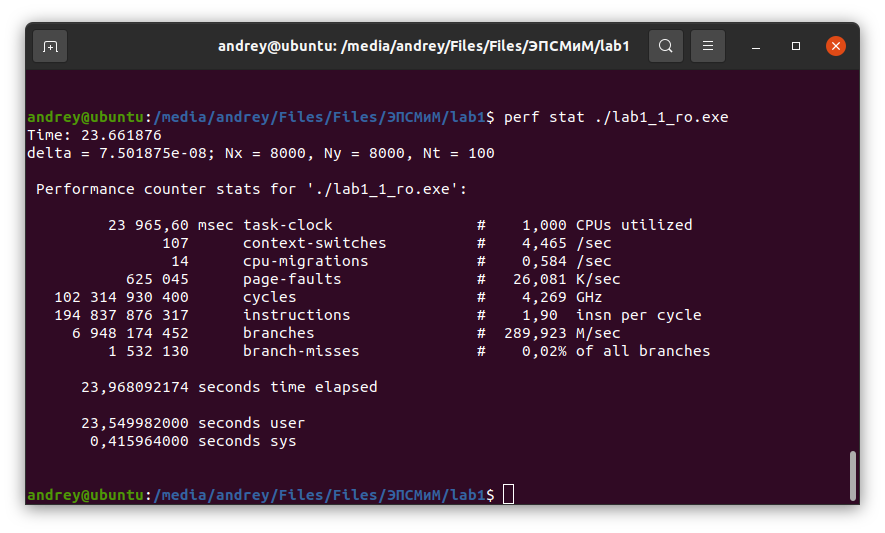
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gcc | -O0 | -O1 | -O2 | -O3 | -Ofast |
| -g | 129.29 | 26.66 | 26.64 | 27.43 | 26.51 |
| -g  -march=native | 128.56 | 26.57 | 23.71 | 24.18 | 23.44 |

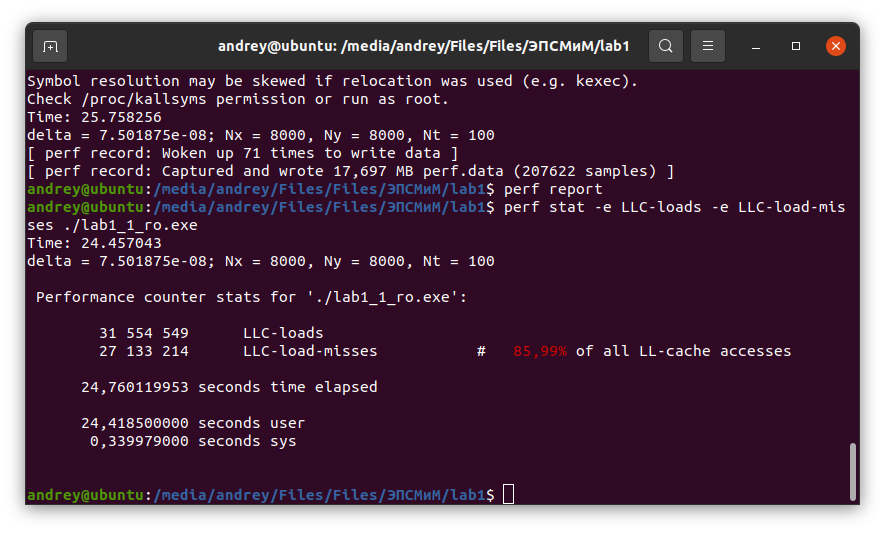
Результаты профилирования программы

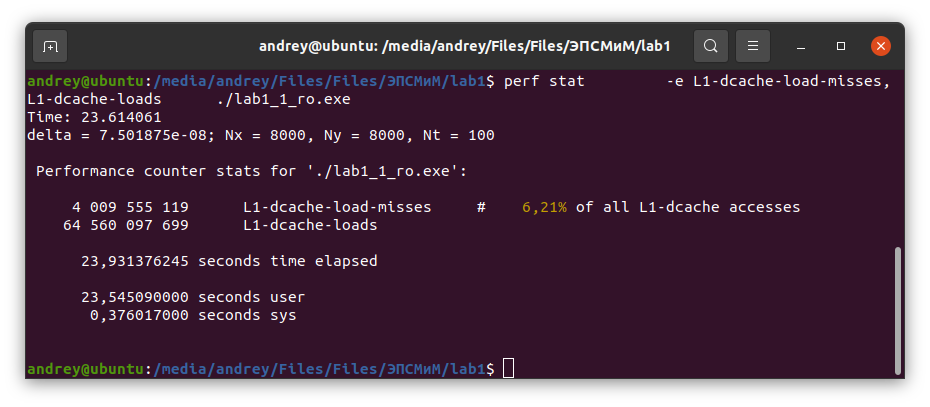
Граф вызовов:



Характеристика исполнения программы:

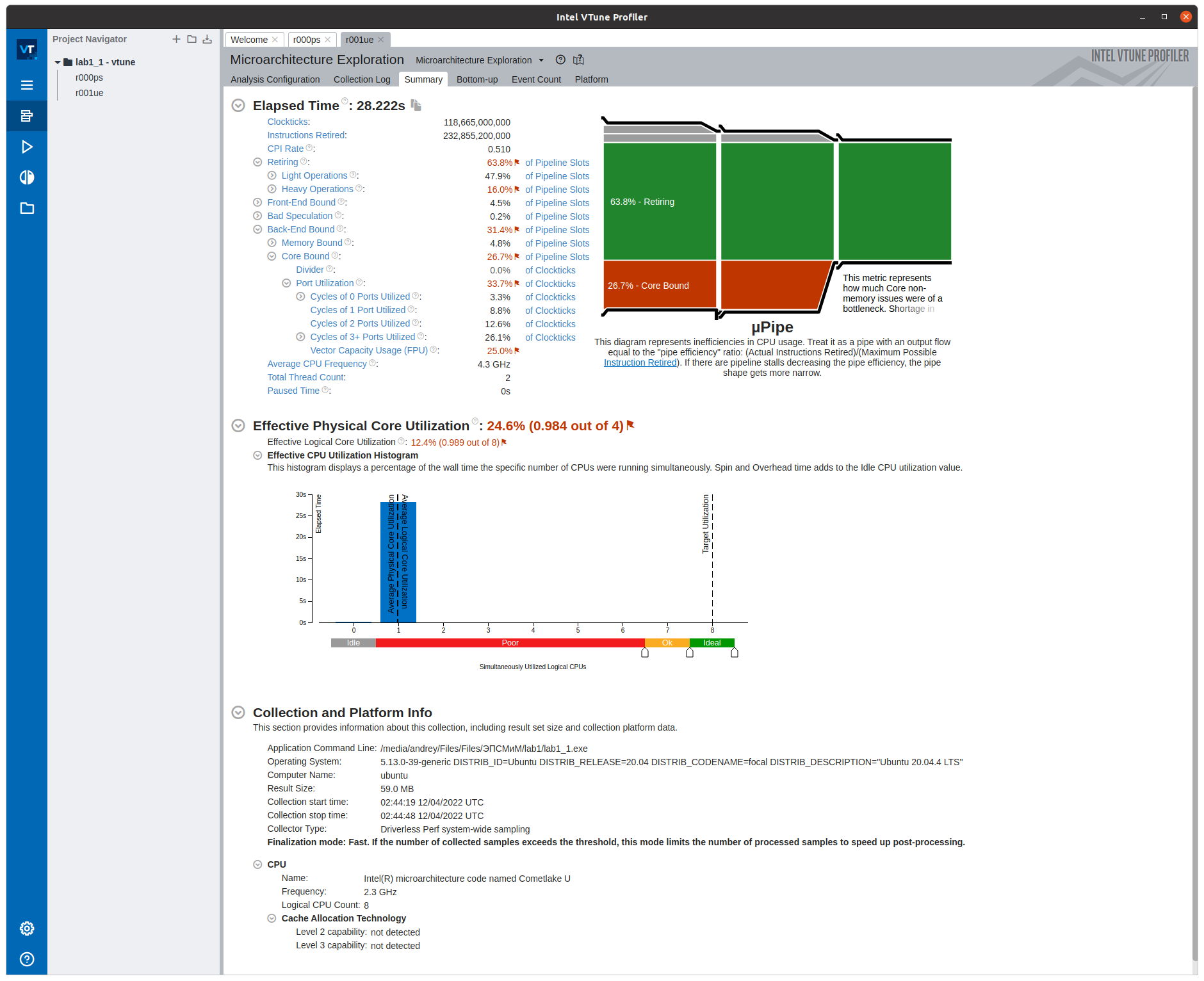


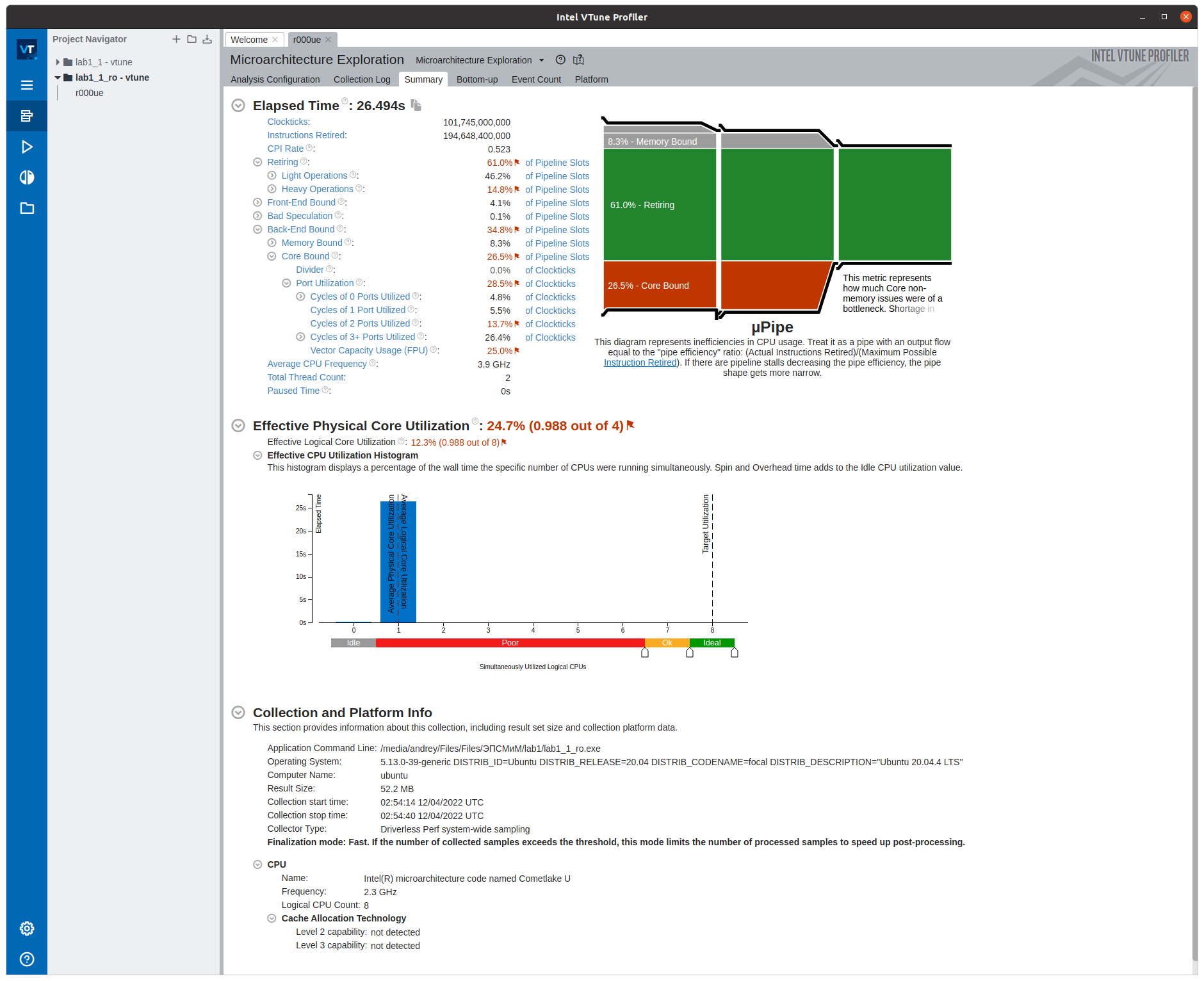




Скриншоты из Vtune:

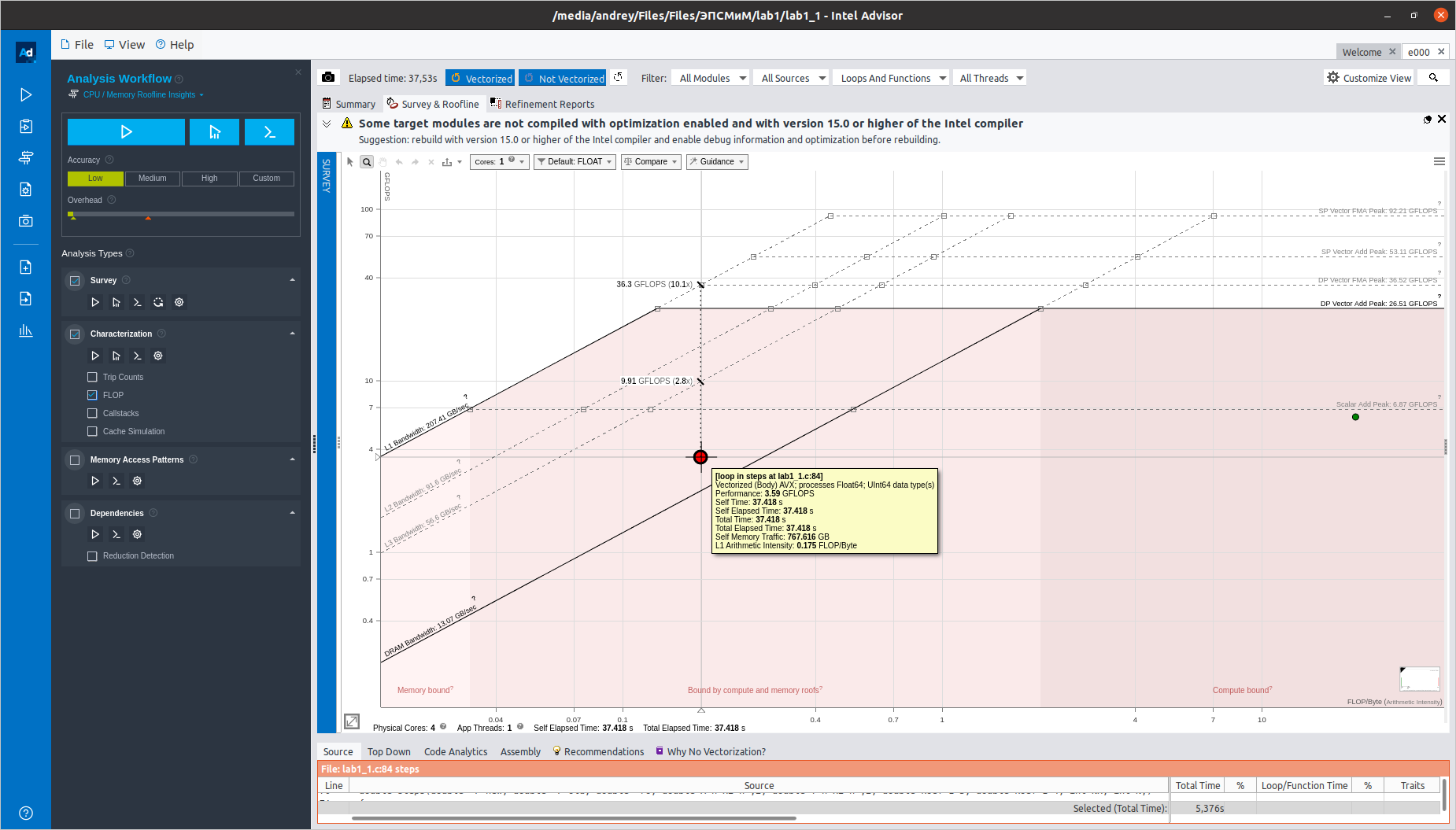
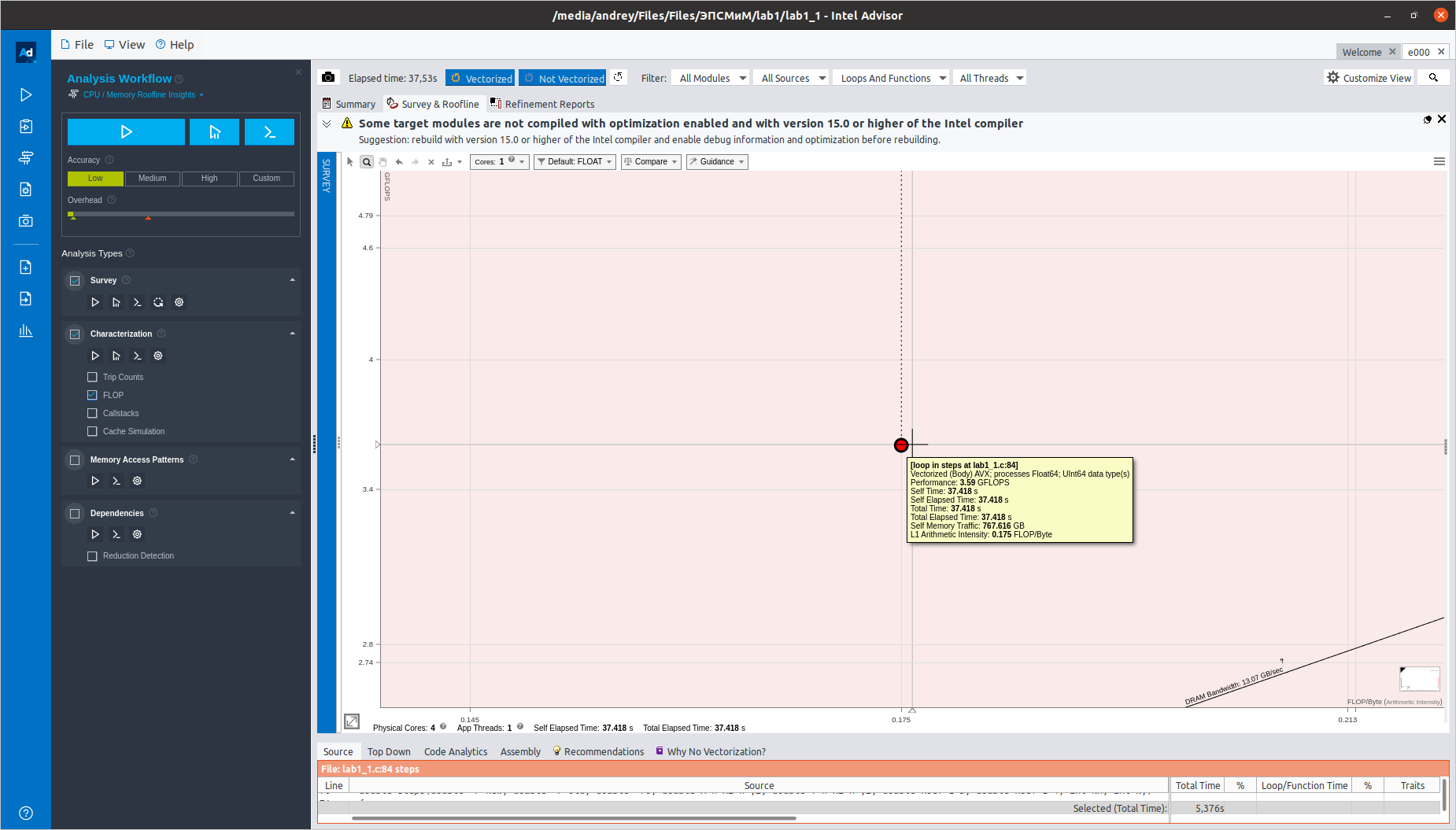
Сравнение двух программ: на 1-ом скриншоте – lab1\_1.c, на 2-ом скриншоте lab1\_1\_ro.c



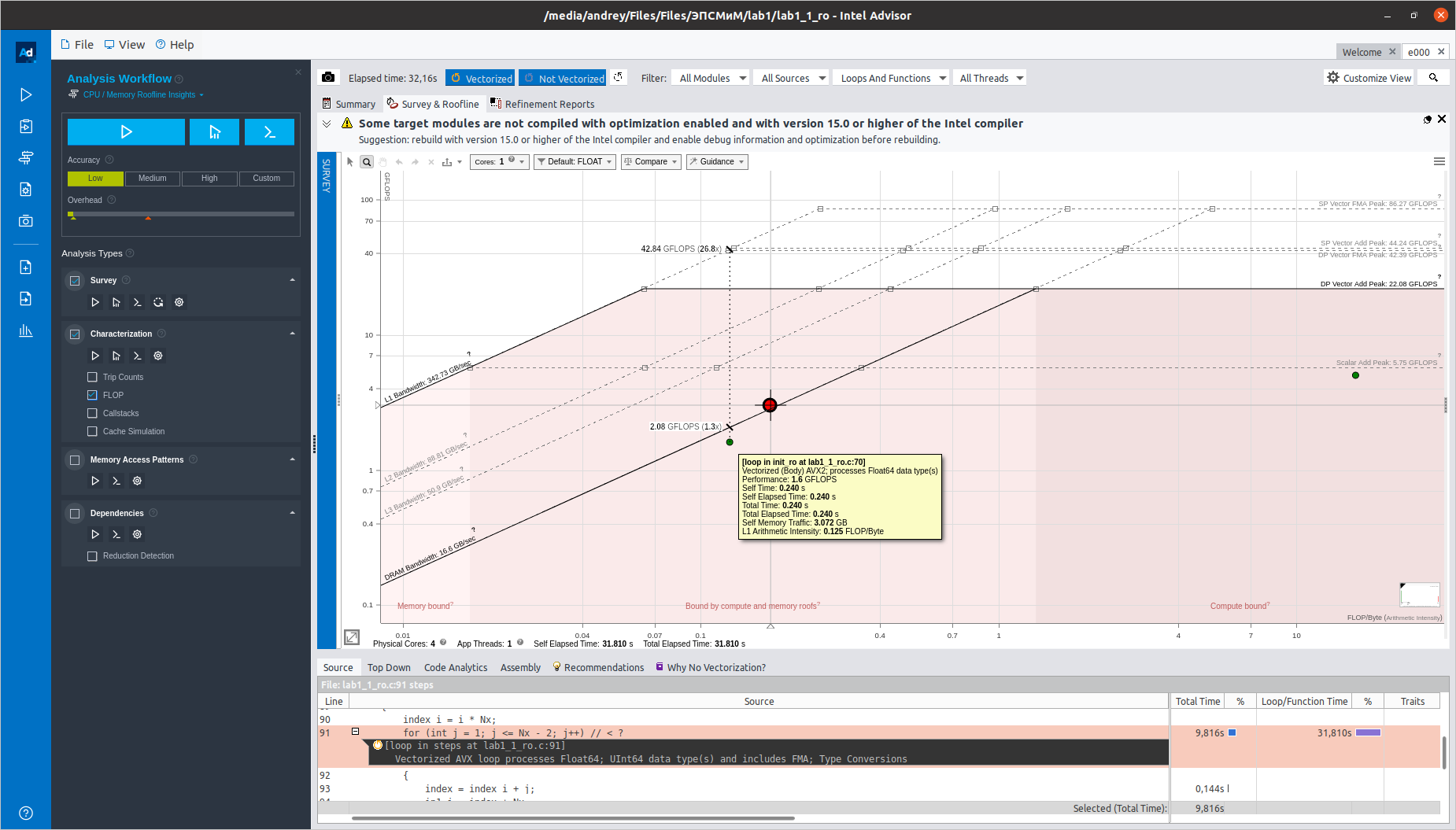
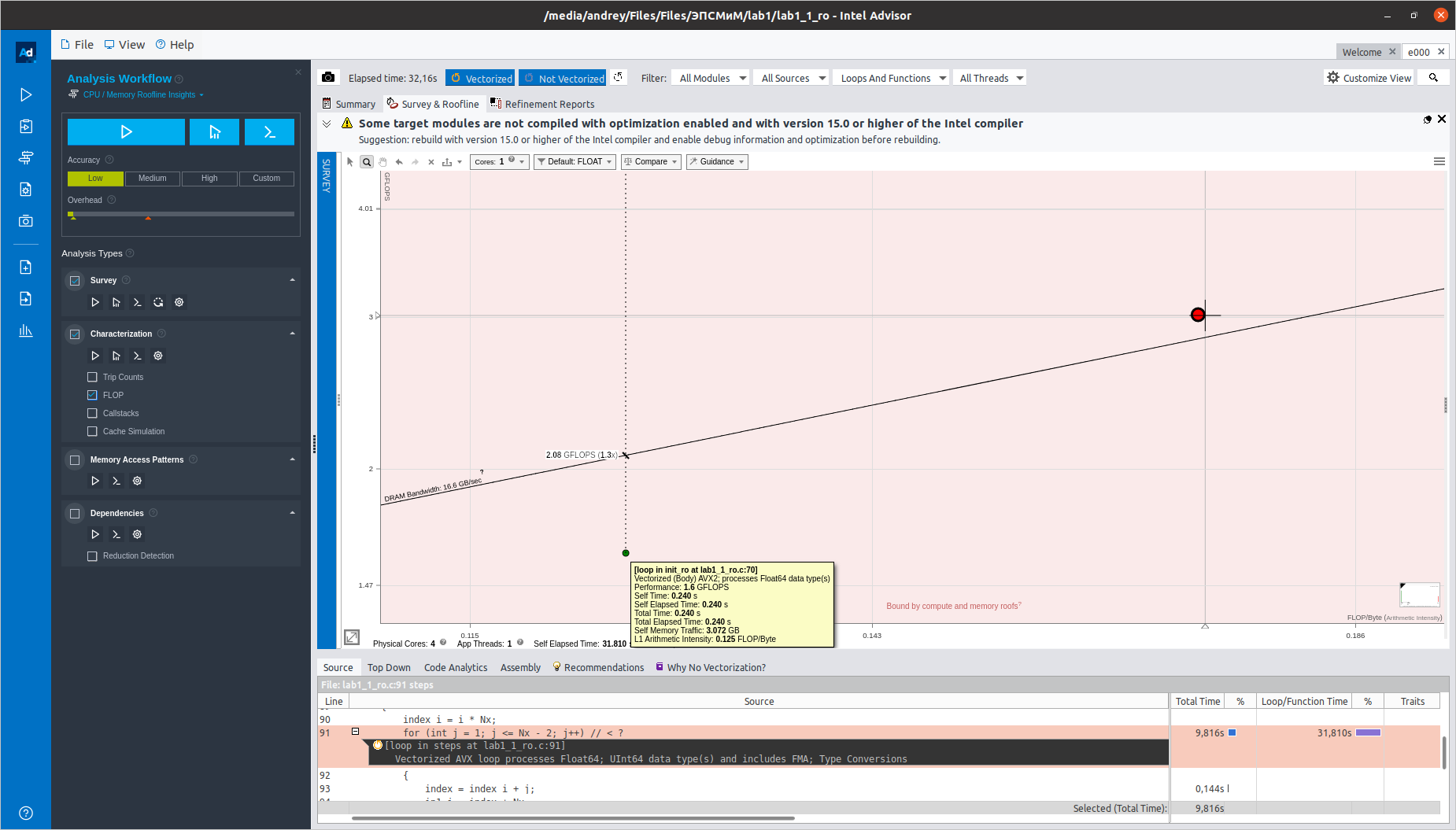
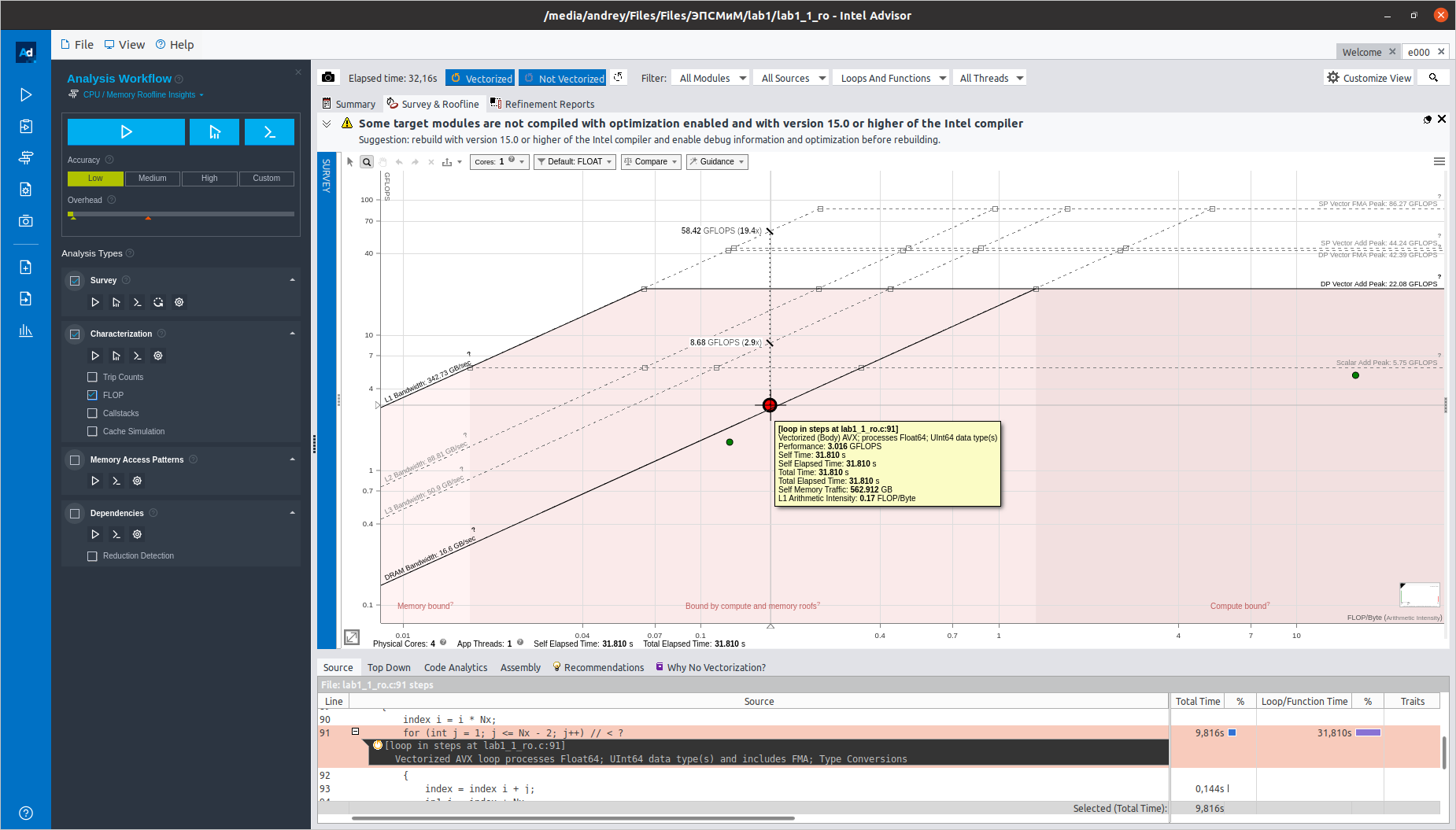
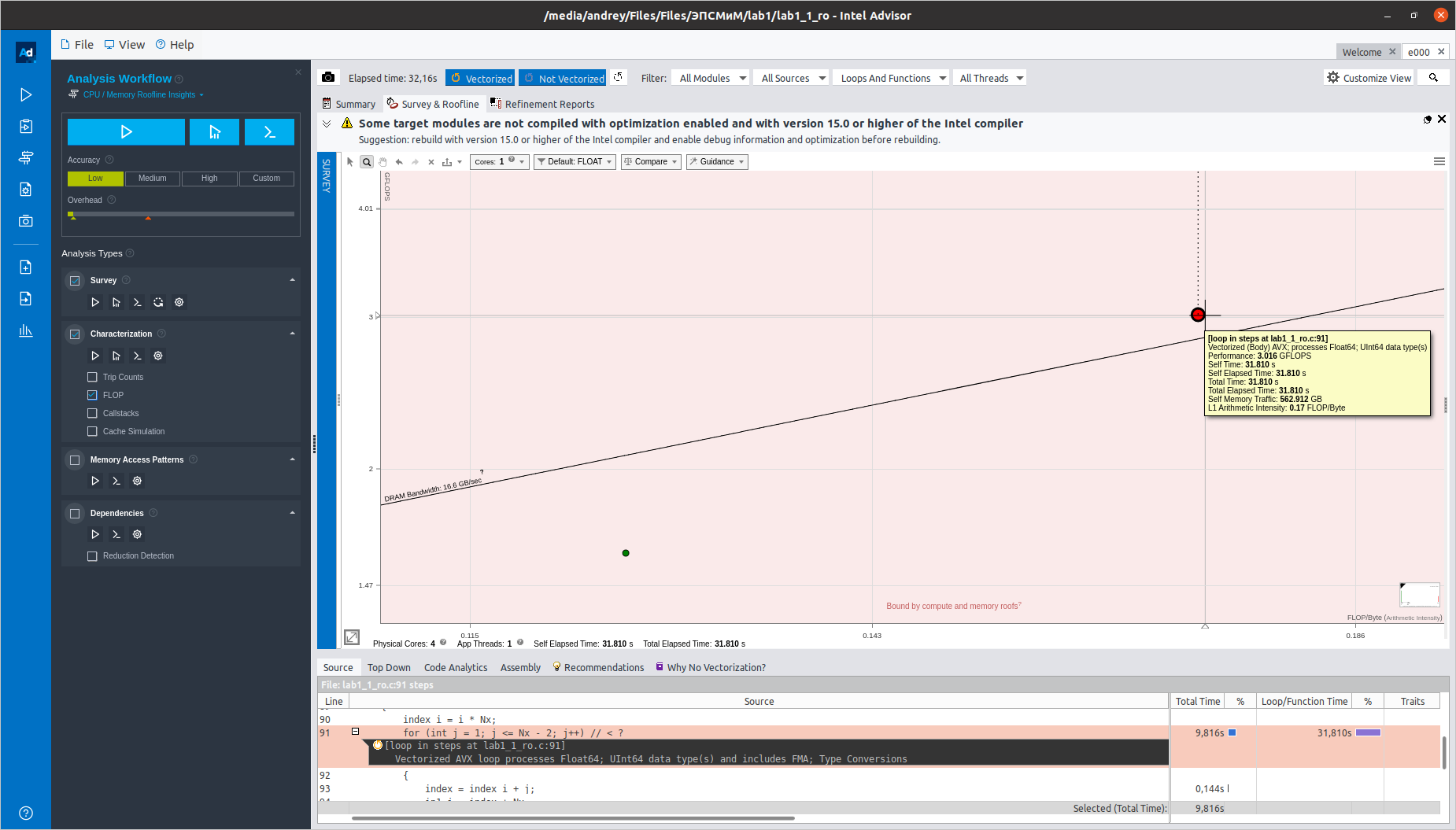


Roofline модель

На этих скриншотах lab1\_1.c:



На остальных скриншотах lab1\_1\_ro.c:



Приложение 1

Код программы, оптимизации которого были наилучшими:

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <math.h>

#include <time.h>

#pragma warning(disable : 4996)

double min(double a, double b)

{

if (a < b)

{

return a;

}

else

{

return b;

}

}

void print\_massive(double\* M, int Nx, int Ny)

{

for (int i = 0; i <= Ny - 1; i++) // < ?

{

for (int j = 0; j <= Nx - 1; j++) // < ?

{

printf("%f\t", M[i \* Nx + j]);

}

printf("\n");

}

printf("\n");

}

void init\_ro(double\*\* ro, int Nx, int Ny, double X\_s1, double X\_s2, double Y\_s1, double Y\_s2, double h\_x, double h\_y, double Xa, double Ya, double X\_ba, double Y\_ba)

{

double X\_j;

double Y\_i;

double R;

double X\_js1;

double X\_js2;

double Y\_is1;

double Y\_is2;

for (int i = 0; i <= Ny - 1; i++) // < ?

{

for (int j = 0; j <= Nx - 1; j++) // < ?

{

X\_j = Xa + j \* h\_x;

Y\_i = Ya + i \* h\_y;

R = 0.1 \* min(X\_ba, Y\_ba); //константа

X\_js1 = X\_j - X\_s1;

Y\_is1 = Y\_i - Y\_s1;

if ((X\_js1 \* X\_js1 + Y\_is1 \* Y\_is1) < R \* R)

{

ro[0][i \* Nx + j] = 1.0;

}

else

{

X\_js2 = X\_j - X\_s2;

Y\_is2 = Y\_i - Y\_s2;

if ((X\_js2 \* X\_js2 + Y\_is2 \* Y\_is2) < R \* R)

{

ro[0][i \* Nx + j] = -1.0;

}

}

}

}

for (int i = 0; i <= Ny - 1; i++) // < ?

{

for (int j = 0; j <= Nx - 1; j++) // < ?

{

ro[1][i \* Nx + j] = 2 \* ro[0][i \* Nx + j] + 0.25 \* (ro[0][i \* Nx + j + 1] + ro[0][i \* Nx + j - 1] + ro[0][(i + 1) \* Nx + j] + ro[0][(i - 1) \* Nx + j]);

}

}

}

double steps(double\* F\_new, double\* F\_old, double\* ro, double X\_h\_x2\_h\_y2, double Y\_h\_x2\_h\_y2, double koef\_1\_5, double koef\_1\_4, int Nx, int Ny)

{

double index\_i;

int index;

int ip1\_j;

int im1\_j;

int i\_jp1;

int i\_jm1;

double max = 0.0;

double delta\_ij;

for (int i = 1; i <= Ny - 2; i++) // < ?

{

index\_i = i \* Nx;

for (int j = 1; j <= Nx - 2; j++) // < ?

{

index = index\_i + j;

ip1\_j = index + Nx;

im1\_j = index - Nx;

i\_jp1 = index + 1;

i\_jm1 = index - 1;

F\_new[index] = koef\_1\_5 \* (

X\_h\_x2\_h\_y2 \* (F\_old[i\_jm1] + F\_old[i\_jp1])

+ Y\_h\_x2\_h\_y2 \* (F\_old[ip1\_j] + F\_old[im1\_j])

+ koef\_1\_4 \* (F\_old[im1\_j - 1] + F\_old[im1\_j + 1] + F\_old[ip1\_j - 1] + F\_old[ip1\_j + 1])

+ ro[index] //old sdvig

);

delta\_ij = fabs(F\_new[index] - F\_old[index]);

if (max < delta\_ij)

{

max = delta\_ij;

}

}

}

//print\_massive(F\_new, Nx, Ny);

return max;

}

int main()

{

int id = 0;

int Nx = 8000;

int Ny = 8000;

int Nt = 100;

double Xa = 0.0;

double Xb = 4.0;

double Ya = 0.0;

double Yb = 4.0;

double X\_ba = Xb - Xa;

double Y\_ba = Yb - Ya;

double h\_x = X\_ba / (Nx - 1);

double h\_y = Y\_ba / (Ny - 1);

double X\_s1 = Xa + X\_ba / 3.0;

double Y\_s1 = Ya + Y\_ba \* 2.0 / 3.0;

double X\_s2 = Xa + X\_ba \* 2.0 / 3.0;

double Y\_s2 = Ya + Y\_ba / 3.0;

double h\_x2 = h\_x \* h\_x;

double h\_y2 = h\_y \* h\_y;

double \_h\_x2\_h\_y2 = 1.0 / h\_x2 + 1.0 / h\_y2;

double X\_h\_x2\_h\_y2 = 0.5 \* (5.0 / h\_x2 - 1.0 / h\_y2);

double Y\_h\_x2\_h\_y2 = 0.5 \* (5.0 / h\_y2 - 1.0 / h\_x2);

double koef\_1\_5 = 0.2 \* (1.0 / \_h\_x2\_h\_y2);

double koef\_1\_4 = 0.25 \* \_h\_x2\_h\_y2;

double delta = 999999999.0;

double delta\_new;

double\*\* F = (double\*\*)calloc(2, sizeof(double\*));

F[0] = (double\*)calloc(Nx \* Ny, sizeof(double));

F[1] = (double\*)calloc(Nx \* Ny, sizeof(double));

memset(F[0], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

memset(F[1], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

double\*\* ro = (double\*\*)calloc(2, sizeof(double\*));

ro[0] = (double\*)calloc(Nx \* Ny, sizeof(double));

ro[1] = (double\*)calloc(Nx \* Ny, sizeof(double));

memset(ro[0], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

memset(ro[1], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

//double\* ro = (double\*)calloc(Nx \* Ny, sizeof(double));

//memset(ro, 0.0, Nx \* Ny \* sizeof(double));

init\_ro(ro, Nx, Ny, X\_s1, X\_s2, Y\_s1, Y\_s2, h\_x, h\_y, Xa, Ya, X\_ba, Y\_ba);

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

for (int i = 0; i < Nt; i++)

{

delta\_new = steps(F[(id + 1) % 2], F[id % 2], ro[1], X\_h\_x2\_h\_y2, Y\_h\_x2\_h\_y2, koef\_1\_5, koef\_1\_4, Nx, Ny);

//print\_massive(F[(id + 1) % 2], Nx, Ny);

//printf("[%d] %lf\n", i, delta\_new);

if (delta < delta\_new)

{

//printf("Waring... [%d] delta\_now = %lf, delta\_prev = %lf\n", i, delta\_new, delta);

}

delta = delta\_new;

id++;

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

printf("Time: %lf\n", (finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec)));

printf("delta = %e; Nx = %d, Ny = %d, Nt = %d\n", delta, Nx, Ny, Nt);

return 0;

}

Приложение 2

Код исходной программы:

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <math.h>

#include <time.h>

#pragma warning(disable : 4996)

double min(double a, double b)

{

if (a < b)

{

return a;

}

else

{

return b;

}

}

void print\_massive(double\* M, int Nx, int Ny)

{

for (int i = 0; i <= Ny - 1; i++) // < ?

{

for (int j = 0; j <= Nx - 1; j++) // < ?

{

printf("%f\t", M[i \* Nx + j]);

}

printf("\n");

}

printf("\n");

}

void init\_ro(double\* ro, int Nx, int Ny, double X\_s1, double X\_s2, double Y\_s1, double Y\_s2, double h\_x, double h\_y, double Xa, double Ya, double X\_ba, double Y\_ba)

{

double X\_j;

double Y\_i;

double R;

double X\_js1;

double X\_js2;

double Y\_is1;

double Y\_is2;

for (int i = 0; i <= Ny - 1; i++) // < ?

{

for (int j = 0; j <= Nx - 1; j++) // < ?

{

X\_j = Xa + j \* h\_x;

Y\_i = Ya + i \* h\_y;

R = 0.1 \* min(X\_ba, Y\_ba); //константа

X\_js1 = X\_j - X\_s1;

Y\_is1 = Y\_i - Y\_s1;

if ((X\_js1 \* X\_js1 + Y\_is1 \* Y\_is1) < R \* R)

{

ro[i \* Nx + j] = 1.0;

}

else

{

X\_js2 = X\_j - X\_s2;

Y\_is2 = Y\_i - Y\_s2;

if ((X\_js2 \* X\_js2 + Y\_is2 \* Y\_is2) < R \* R)

{

ro[i \* Nx + j] = -1.0;

}

}

}

}

}

double steps(double\* F\_new, double\* F\_old, double\* ro, double h\_x2, double h\_y2, int Nx, int Ny)

{

double \_h\_x2\_h\_y2 = 1.0 / h\_x2 + 1.0 / h\_y2;

double X\_h\_x2\_h\_y2 = 0.5 \* (5.0 / h\_x2 - 1.0 / h\_y2);

double Y\_h\_x2\_h\_y2 = 0.5 \* (5.0 / h\_y2 - 1.0 / h\_x2);

double koef\_1\_5 = 0.2 \* (1.0 / \_h\_x2\_h\_y2);

double koef\_1\_4 = 0.25 \* \_h\_x2\_h\_y2;

double index\_i;

int index;

int ip1\_j;

int im1\_j;

int i\_jp1;

int i\_jm1;

double max = 0.0;

double delta\_ij;

for (int i = 1; i <= Ny - 2; i++) // < ?

{

index\_i = i \* Nx;

for (int j = 1; j <= Nx - 2; j++) // < ?

{

index = index\_i + j;

ip1\_j = index + Nx;

im1\_j = index - Nx;

i\_jp1 = index + 1;

i\_jm1 = index - 1;

F\_new[index] = koef\_1\_5 \* (

X\_h\_x2\_h\_y2 \* (F\_old[i\_jm1] + F\_old[i\_jp1])

+ Y\_h\_x2\_h\_y2 \* (F\_old[ip1\_j] + F\_old[im1\_j])

+ koef\_1\_4 \* (F\_old[im1\_j - 1] + F\_old[im1\_j + 1] + F\_old[ip1\_j - 1] + F\_old[ip1\_j + 1])

+ 2.0 \* ro[index] + 0.25 \* (ro[im1\_j] + ro[ip1\_j] + ro[i\_jm1] + ro[i\_jp1]) //old sdvig

);

delta\_ij = fabs(F\_new[index] - F\_old[index]);

if (max < delta\_ij)

{

max = delta\_ij;

}

}

}

//print\_massive(F\_new, Nx, Ny);

return max;

}

int main()

{

int id = 0;

int Nx = 8000;

int Ny = 8000;

int Nt = 100;

double Xa = 0.0;

double Xb = 4.0;

double Ya = 0.0;

double Yb = 4.0;

double X\_ba = Xb - Xa;

double Y\_ba = Yb - Ya;

double h\_x = X\_ba / (Nx - 1);

double h\_y = Y\_ba / (Ny - 1);

double X\_s1 = Xa + X\_ba / 3.0;

double Y\_s1 = Ya + Y\_ba \* 2.0 / 3.0;

double X\_s2 = Xa + X\_ba \* 2.0 / 3.0;

double Y\_s2 = Ya + Y\_ba / 3.0;

double h\_x2 = h\_x \* h\_x;

double h\_y2 = h\_y \* h\_y;

double delta = 999999999.0;

double delta\_new;

double\*\* F = (double\*\*)calloc(2, sizeof(double\*));

F[0] = (double\*)calloc(Nx \* Ny, sizeof(double));

F[1] = (double\*)calloc(Nx \* Ny, sizeof(double));

memset(F[0], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

memset(F[1], 0.0, Nx \* Ny \* sizeof(double)); //надо только по бокам

double\* ro = (double\*)calloc(Nx \* Ny, sizeof(double));

memset(ro, 0.0, Nx \* Ny \* sizeof(double));

init\_ro(ro, Nx, Ny, X\_s1, X\_s2, Y\_s1, Y\_s2, h\_x, h\_y, Xa, Ya, X\_ba, Y\_ba);

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

for (int i = 0; i < Nt; i++)

{

delta\_new = steps(F[(id + 1) % 2], F[id % 2], ro, h\_x2, h\_y2, Nx, Ny);

//print\_massive(F[(id + 1) % 2], Nx, Ny);

//printf("[%d] %lf\n", i, delta\_new);

if (delta < delta\_new)

{

//printf("Waring... [%d] delta\_now = %lf, delta\_prev = %lf\n", i, delta\_new, delta);

}

delta = delta\_new;

id++;

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

printf("Time: %lf\n", (finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec)));

printf("delta = %e; Nx = %d, Ny = %d, Nt = %d\n", delta, Nx, Ny, Nt);

return 0;

}

Вывод:

По roofline модели, мужду двумя программами можно заметить что, как только был пересчитан массив с константами, то увеличилась интенсивность (точка сместилась вправо)

По моей архитектуре процессора 4 декодера, поэтому пиковое значение операций в секунду 4. У меня получилось 1.9, что в целом можно считать неплохим результатом