Эффективное программирование современных микропроцессоров и мультипроцессоров

Задание 3

Выполнил:

Пирожков Андрей

19210

Преподаватель:

Константин Викторович

Задание

1. В векторизованной программе из практического задания 2 реорганизовать вычисления таким образом, чтобы за один проход по сетке выполнялось сразу несколько итераций метода. Сделать сначала реализацию для двух итераций за проход. Если есть ускорение, сделать реализацию для трёх итераций за проход. И так далее, пока ускорение не перестанет иметь место.

2. Проанализировать производительность наиболее быстрой версии программы аналогично анализу в

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

Версия компилятора: gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

Результаты тестирования программ

* Результаты самой первой программы

Ключи: -Ofast -march=native

Результат: 23.44

* Автоматическая векторизация компилятором (тоже предыдущий результат)

Ключи: -Ofast -march=native

Результат: 10.21

* Моя ручная векторизация (тоже предыдущий результат)

Ключи: -Ofast -march=native

Результат: 9.95

* Выполнение программы для r = 1

Ключи: -Ofast -march=native

Результат: 10.52

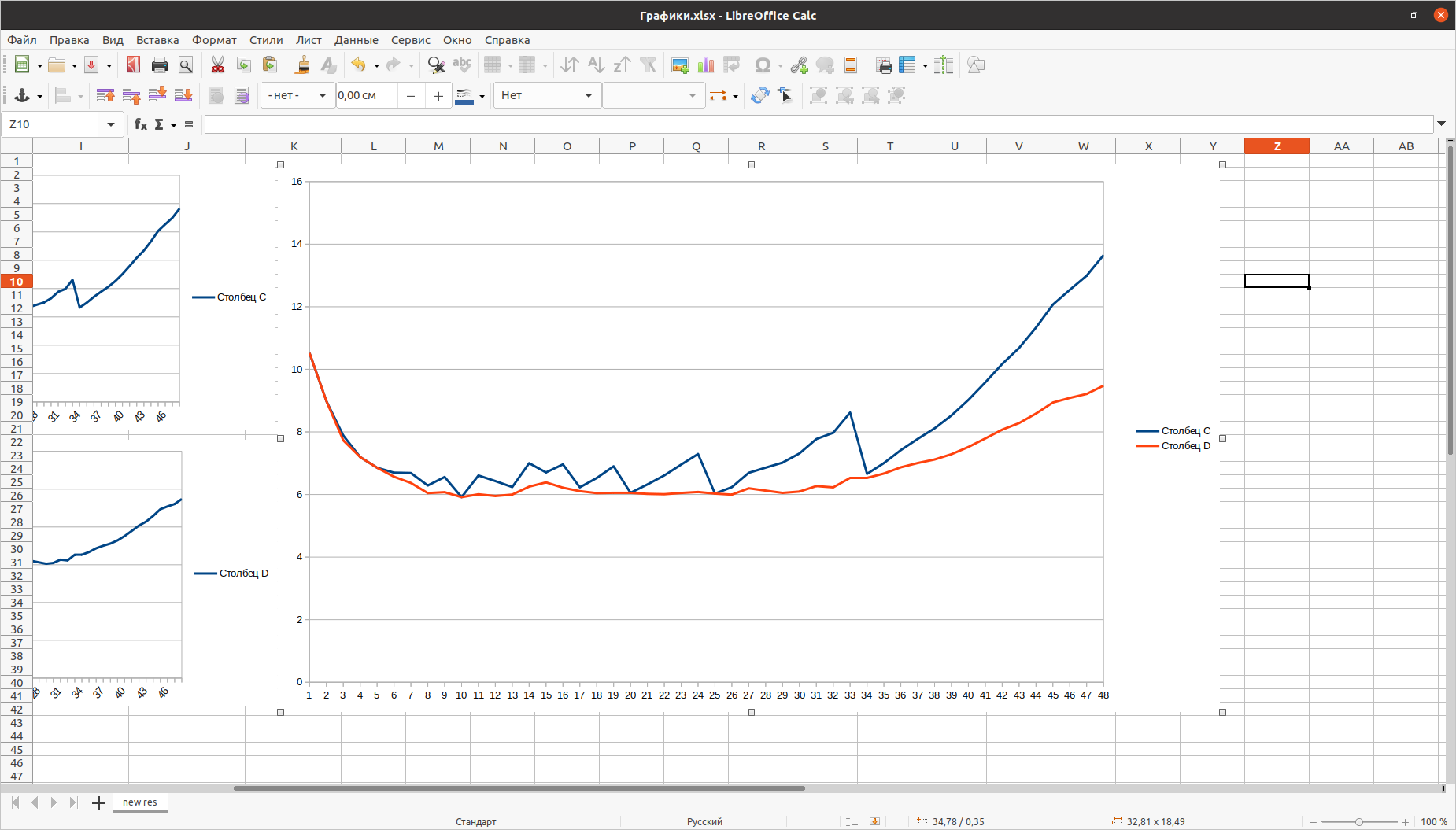
Получилось немного медленнее, но это вероятно из-за погрешности ,а также других накладных вычислений

* Выполнение программы для r = 10 получил наименьшее время выполнения

Ключи: -Ofast -march=native

Результат: 5.92

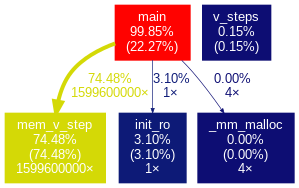
В среднем при r = 8 .. 30 получаются самые быстрые результаты в районе 6-и секунд



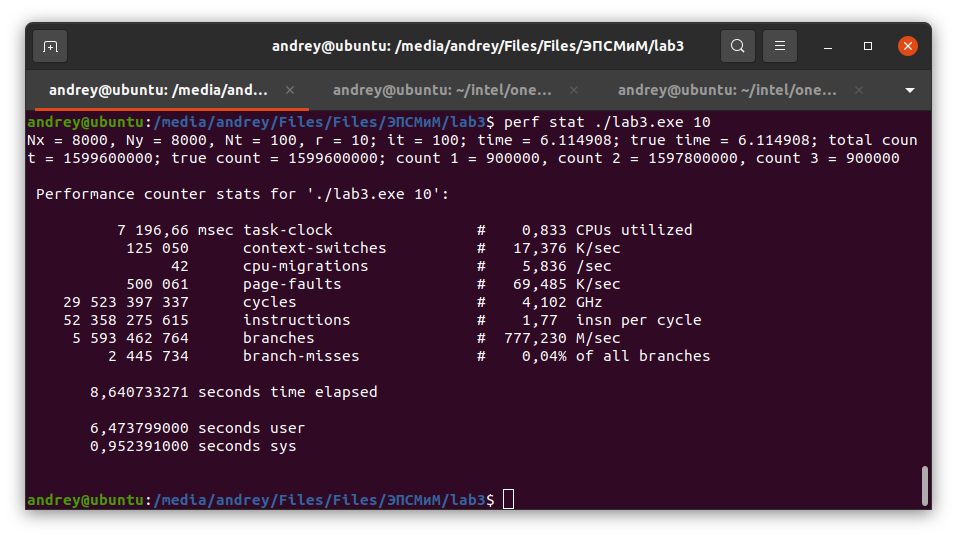
На графике по вертикали — время, по горизонтали — r (одновременная обработка строк)

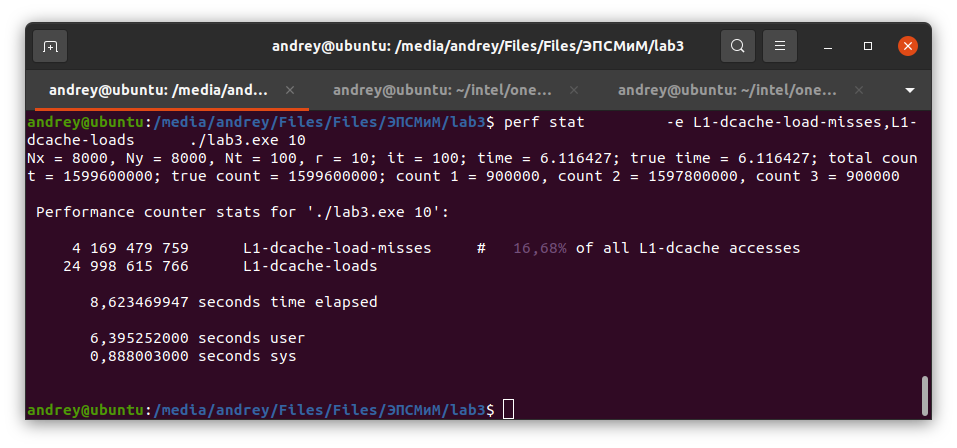
Результаты профилирования программы

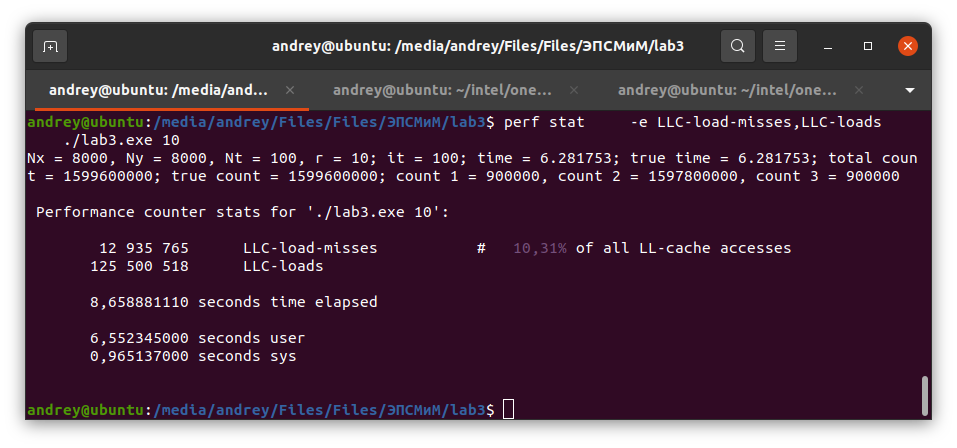
Граф вызовов:



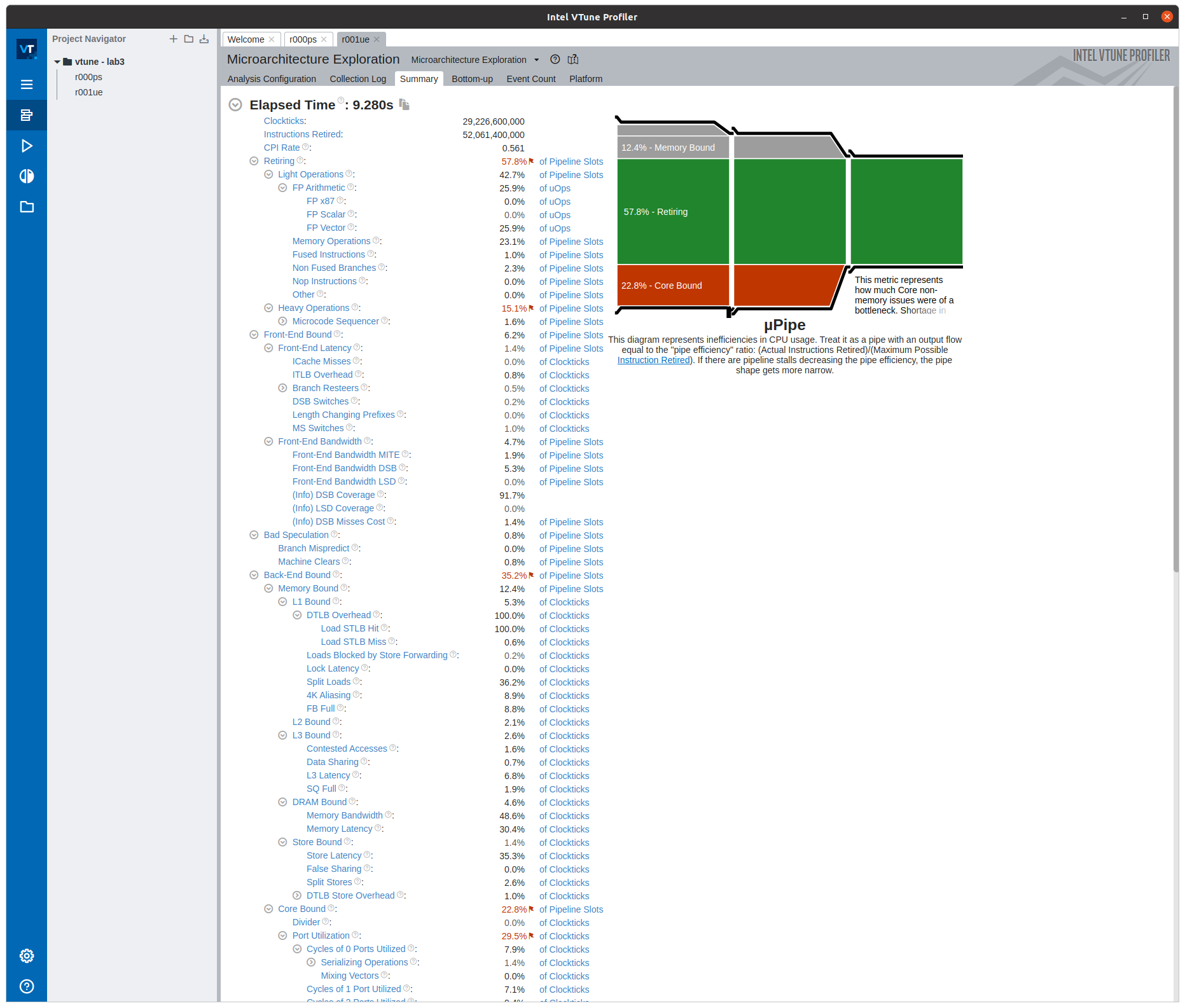
Характеристика исполнения программы:

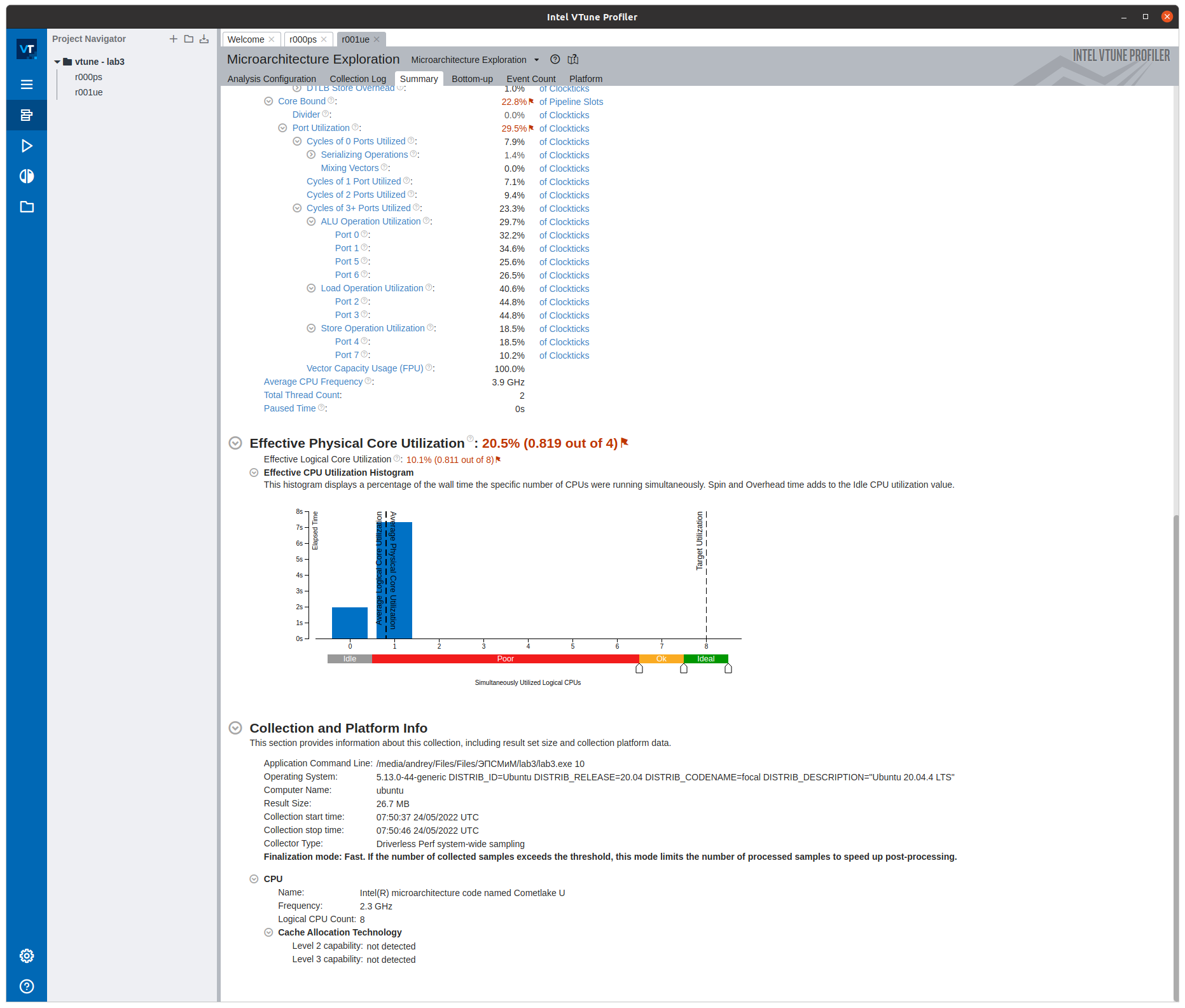




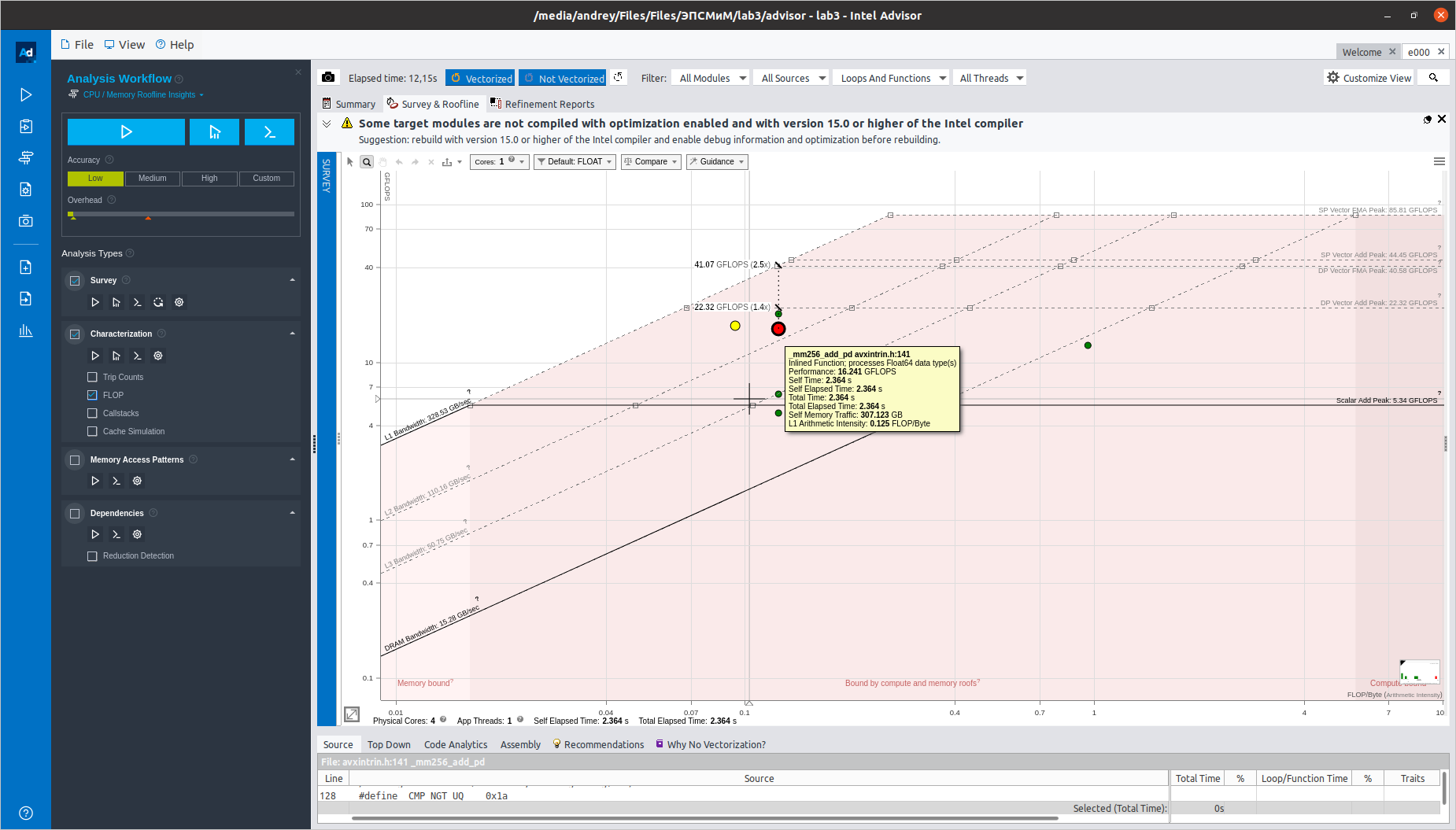


Скриншоты из Vtune:





Roofline модель



Приложение 1

Код программы lab3.c:

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <math.h>

#include <time.h>

#include <immintrin.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;

}

else

{

ro[0][i \* Nx + j] = 0.0;

}

}

}

}

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

{

for (int j = 1; j <= Nx - 2; 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]);

}

}

}

void mem\_v\_step(double\* F\_new, double\* F\_old, double\* ro, \_\_m256d X\_h\_x2\_h\_y2, \_\_m256d Y\_h\_x2\_h\_y2, \_\_m256d koef\_1\_5, \_\_m256d koef\_1\_4, int Nx, int Ny, int i, int j/\*, \_\_m256d frist1,\_\_m256d second1, \_\_m256d frist2, \_\_m256d second2, \_\_m256d frist3, \_\_m256d second3, \_\_m256d thrid3, \_\_m256d fouth3, \_\_m256d frist5, \_\_m256d second5, \_\_m256d result1\_1, \_\_m256d result1\_2, \_\_m256d result2\_1, \_\_m256d result2\_2, \_\_m256d result3\_1, \_\_m256d result3\_2, \_\_m256d result3\_3, \_\_m256d result3\_4, \_\_m256d result4\_1, \_\_m256d result5\_1, \_\_m256d total\*/)

{

\_\_m256d frist1;

\_\_m256d second1;

\_\_m256d frist2;

\_\_m256d second2;

\_\_m256d frist3;

\_\_m256d second3;

\_\_m256d thrid3;

\_\_m256d fouth3;

\_\_m256d frist5;

\_\_m256d second5;

\_\_m256d result1\_1;

\_\_m256d result1\_2;

\_\_m256d result2\_1;

\_\_m256d result2\_2;

\_\_m256d result3\_1;

\_\_m256d result3\_2;

\_\_m256d result3\_3;

\_\_m256d result3\_4;

\_\_m256d result4\_1;

\_\_m256d result5\_1;

\_\_m256d total;

frist1 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j - 1);

second1 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j + 1);

result1\_1 = \_mm256\_add\_pd(frist1, second1);

result1\_2 = \_mm256\_mul\_pd(X\_h\_x2\_h\_y2, result1\_1);

//Y\_h\_x2\_h\_y2 \* (F\_old[i \* Nx + j + Nx] + F\_old[i \* Nx + j - Nx])

frist2 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j + Nx);

second2 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j - Nx);

result2\_1 = \_mm256\_add\_pd(frist2, second2);

result2\_2 = \_mm256\_mul\_pd(Y\_h\_x2\_h\_y2, result2\_1);

//koef\_1\_4 \* (F\_old[i \* Nx + j - Nx - 1] + F\_old[i \* Nx + j - Nx + 1] + F\_old[i \* Nx + j + Nx - 1] + F\_old[i \* Nx + j + Nx + 1])

frist3 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j - Nx - 1);

second3 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j - Nx + 1);

thrid3 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j + Nx - 1);

fouth3 = \_mm256\_loadu\_pd(F\_old + i \* Nx + j + Nx + 1);

result3\_1 = \_mm256\_add\_pd(frist3, second3);

result3\_2 = \_mm256\_add\_pd(thrid3, fouth3);

result3\_3 = \_mm256\_add\_pd(result3\_1, result3\_2);

result3\_4 = \_mm256\_mul\_pd(koef\_1\_4, result3\_3);

//ro[i \* Nx + j]

result4\_1 = \_mm256\_loadu\_pd(ro + i \* Nx + j);

//Sum results

frist5 = \_mm256\_add\_pd(result1\_2, result2\_2);

second5 = \_mm256\_add\_pd(result3\_4, result4\_1);

result5\_1 = \_mm256\_add\_pd(frist5, second5);

total = \_mm256\_mul\_pd(koef\_1\_5, result5\_1);

\_mm256\_storeu\_pd(F\_new + i \* Nx + j, total);

}

int main(int argc, char\* argv[])

{

int r = 1;

if (argv[1] != NULL)

{

r = atoi(argv[1]);

}

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;

\_\_m256d v\_\_X\_h\_x2\_h\_y2 = \_mm256\_set1\_pd(X\_h\_x2\_h\_y2);

\_\_m256d v\_\_Y\_h\_x2\_h\_y2 = \_mm256\_set1\_pd(Y\_h\_x2\_h\_y2);

\_\_m256d v\_\_koef\_1\_5 = \_mm256\_set1\_pd(koef\_1\_5);

\_\_m256d v\_\_koef\_1\_4 = \_mm256\_set1\_pd(koef\_1\_4);

double delta = 999999999.0;

double delta\_new;

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

F[0] = (double\*)\_mm\_malloc(Nx\*Ny\*sizeof(double), 32);

F[1] = (double\*)\_mm\_malloc(Nx\*Ny\*sizeof(double), 32);

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\*)\_mm\_malloc(Nx\*Ny\*sizeof(double), 32);

ro[1] = (double\*)\_mm\_malloc(Nx\*Ny\*sizeof(double), 32);

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

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);

int it, i, j;

for (it = 0; it < Nt; it += r)

{

for (int k = 1; k < r; k++)

{

i = k;

for (int l = 1; l <= i; l++)

{

for (j = 1; j <= Nx - 2; j+=4)

{

mem\_v\_step(F[(id + l) % 2], F[(id + l - 1) % 2], ro[1], v\_\_X\_h\_x2\_h\_y2, v\_\_Y\_h\_x2\_h\_y2, v\_\_koef\_1\_5, v\_\_koef\_1\_4, Nx, Ny, i - (l - 1), j);

}

}

}

for (i = r; i <= Ny - 2; i++)

{

for (int l = 1; l <= r; l++)

{

for (j = 1; j <= Nx - 2; j += 4)

{

//printf("id = %d, l = %d, i = %d, j = %d\n", id, l, i, j);

mem\_v\_step(F[(id + l) % 2], F[(id + l - 1) % 2], ro[1], v\_\_X\_h\_x2\_h\_y2, v\_\_Y\_h\_x2\_h\_y2, v\_\_koef\_1\_5, v\_\_koef\_1\_4, Nx, Ny, i - (l - 1), j);

}

}

}

for (int k = 2; k <= r; k++)

{

i = Ny - 1;

for (int l = k, m = 1; l <= r; l++, m++)

{

mem\_v\_step(F[(id + l) % 2], F[(id + l - 1) % 2], ro[1], v\_\_X\_h\_x2\_h\_y2, v\_\_Y\_h\_x2\_h\_y2, v\_\_koef\_1\_5, v\_\_koef\_1\_4, Nx, Ny, i - m, j);

}

}

if (delta < delta\_new)

{

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

}

delta = delta\_new;

id += r;

}

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

FILE\* file = fopen("file", "wb");

fwrite(F[id % 2], sizeof(double), Nx \* Ny, file);

//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);

double time = finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec);

double true\_time = time / (double)it \* (double)Nt;

printf("Nx = %d, Ny = %d, Nt = %d, r = %d; it = %d; time = %lf; true time = %lf\n", Nx, Ny, Nt, r, it, time, true\_time);

return 0;

}

Вывод:

Программа теперь больше не упирается в память так сильно как предыдущая. График в Vtune теперь похож больше на график из 1-ой лабы. Порты на процессоре более равномерно загружены относительно предыдущей программы.

Число инструкций повысились на каждый цикл. Скорее всего это связано с тем что появилось много дополнительные вычислений в циклах. Также снизились кэш-промахи. Необходимо сначала загружать, а потом считать. Видимо это более долгие операции

Roofline модель улучшилась и теперь она находится в среднем на уровне L2 кэша.