Эффективное программирование современных микропроцессоров и мультипроцессоров

Задание 4

Выполнил:

Пирожков Андрей

19210

Преподаватель:

Константин Викторович

Задание

Постановка задачи

1. Программу из практического задания 2 необходимо распараллелить на множество потоков над общей памятью.
   * Для распараллеливания следует использовать метод одномерной декомпозиции пространства моделирования – множество строк разделяется на несколько подмножеств по числу потоков, каждый поток обрабатывает свои строки. В конце каждой итерации ставится барьерная синхронизация.
   * Массивы выделяются одним блоком, как в последовательной программе. Каждый поток осуществляет инициализацию своих элементов массивов (осуществляет первое обращение к своим данным). Потоки должны быть привязаны к ядрам.
2. Выполните оптимизацию программы по памяти, аналогичную оптимизации из практического задания 3 – выполнение нескольких итераций за одни проход по массивам. Пример порядка обработки строк массива приведён на рис. 1.
   * Разрешается реализовать вариант программы только с одним фиксированным количеством итераций на проход: 4 или более.
   * Сбалансируйте внутренние границы между подобластями с учётом выступающих «зелёных» строк.
   * В приведённой схеме есть два места, где требуется синхронизация: переход с «красного» этапа на «синий» и переход с «синего» этапа на «зелёный». Используйте в этих местах барьерную синхронизацию.
3. Измените барьерную синхронизацию на попарную синхронизацию между потоками с помощью флаговых переменных и атомарных операций с ними. Флаговые переменные разных потоков должны отстоять в памяти друг от друга не менее чем на размер кэш-строки.
4. Сравните и проанализируйте время работы программ из пунктов 1, 2, 3 на различном числе ядер. Для тестирования используйте многоядерную вычислительную систему с числом ядер не менее 4-х.
5. Проанализируйте производительность программ из пунктов 1 и 3 на любой многоядерной вычислительной системе на числе ядер, дающем наименьшее время, аналогично анализу в предыдущих заданиях (включая roofline-модель). Сравните результаты анализа с результатами в задании 3. На основе результатов анализа (roofline и других) дайте заключение о том, что именно ограничивает возможности достижения большего ускорения при распараллеливании.

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

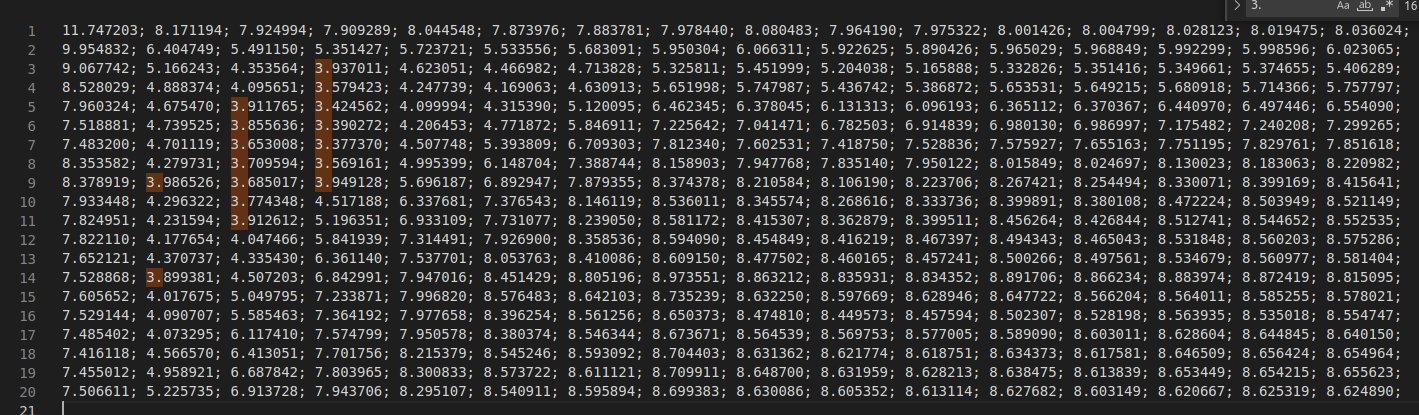
Результат: 9.95

* Выполнение программы для r = 10 получил наименьшее время выполнения (тоже предыдущий результат)

Ключи: -Ofast -march=native

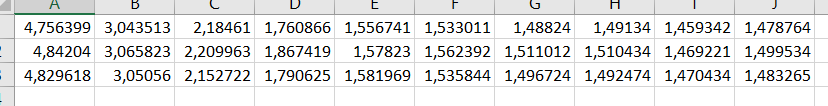
Результат: 5.92

* Провёл тестирование для разных потоков и разных размеров лестницы. Потоков от 1 до 16, лестниц от 1 до 20. Результаты представлены на графике и в сводной таблице. Местами программа работала чуть более 3 секунд. Это происходило чаще всего на 3ех или 4ех потоках. Иногда на 2ух. Оптимальные размеры лесенки — от 4 до 8.

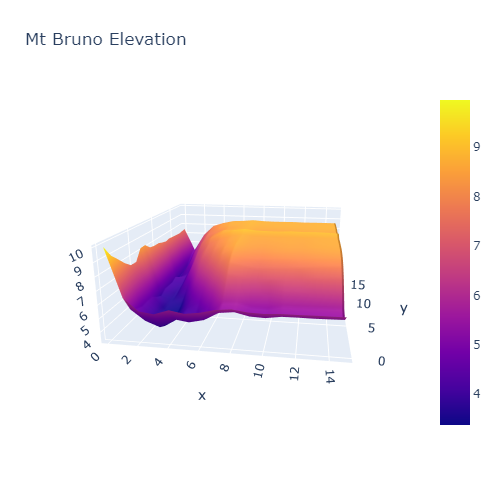


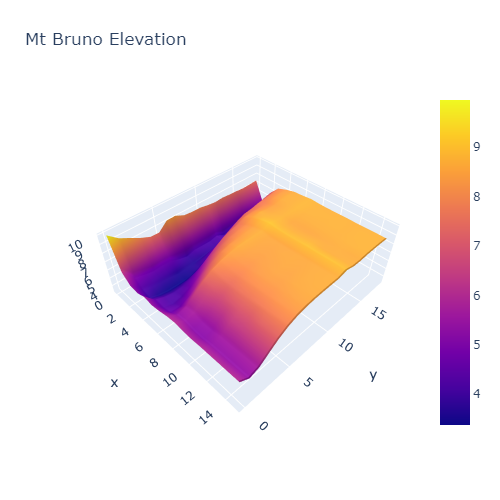
Эти данные сформированные на основе барьеров (графики ниже)

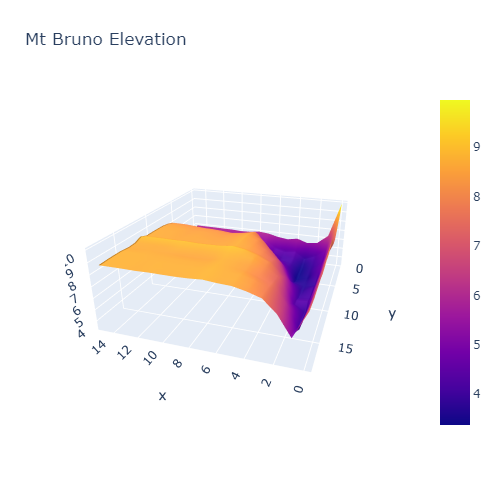
* Выбрал я 4 потока и размер r=6. Результат получился 3.10 с. Эту программу я использовал для профилирования в профилировщиках
* Решил провести сравнение между синхронизациями: 1-ая строка – без синхронизации, 2-ая строка с барьерной синхронизацией, 3-яя строка – с атомарной синхронизацией на volatile переменных массивах

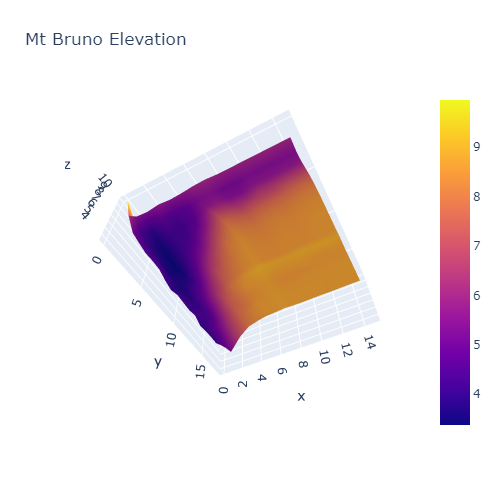


Исходя из полученных данных, в среднем атомарная синхронизация по времени меньше, чем барьерная, но не больше, чем вообще без синхронизации. Однако есть исключения, которые я бы отнёс к погрешностям измерениям. Эти тесты проводил на Nx=Ny=2000 Nt=1000.



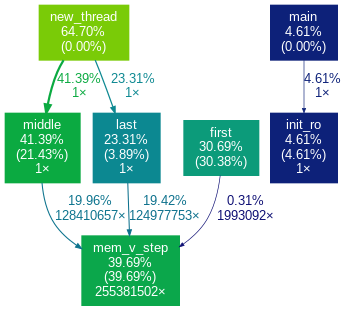




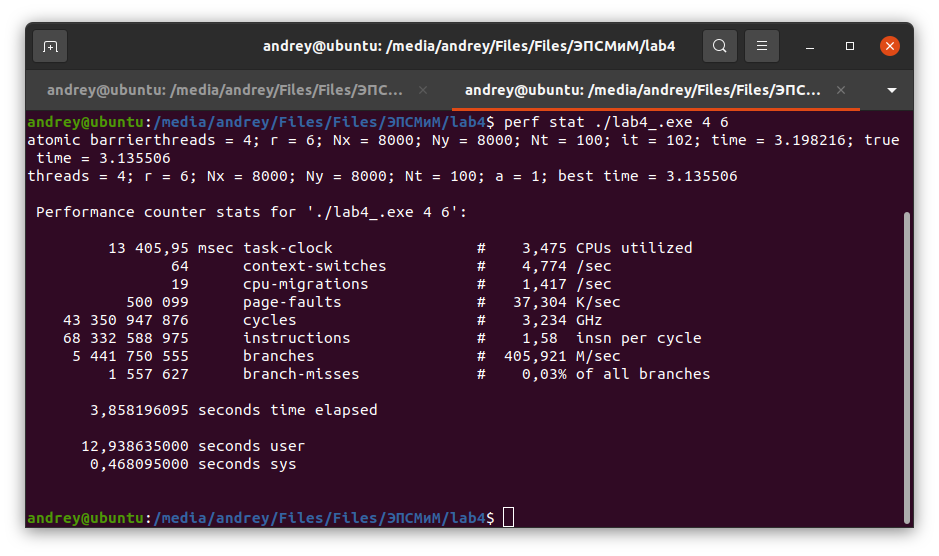


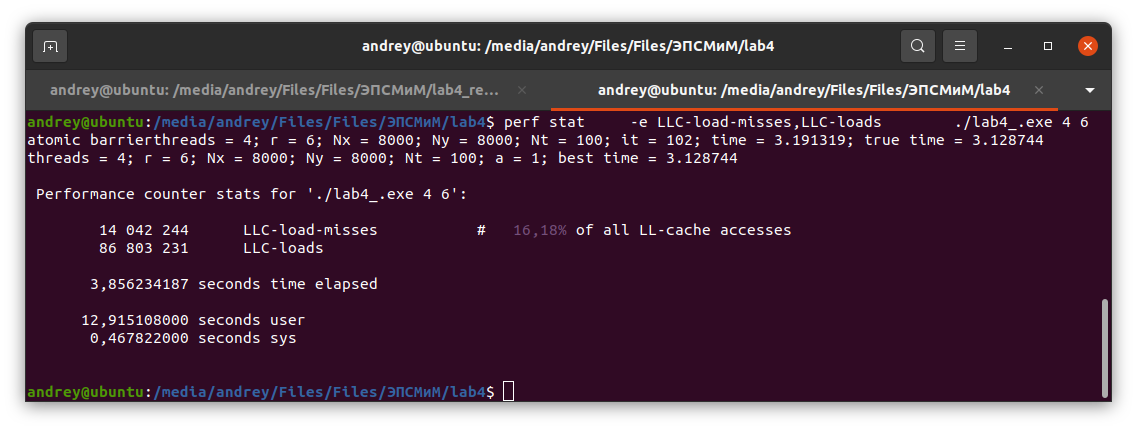
Результаты профилирования программы

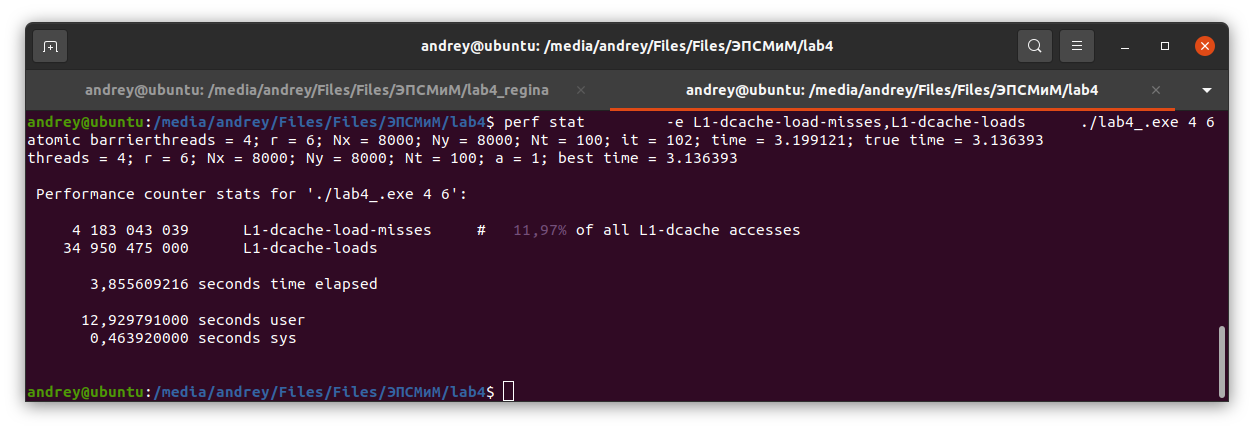
Граф вызовов:



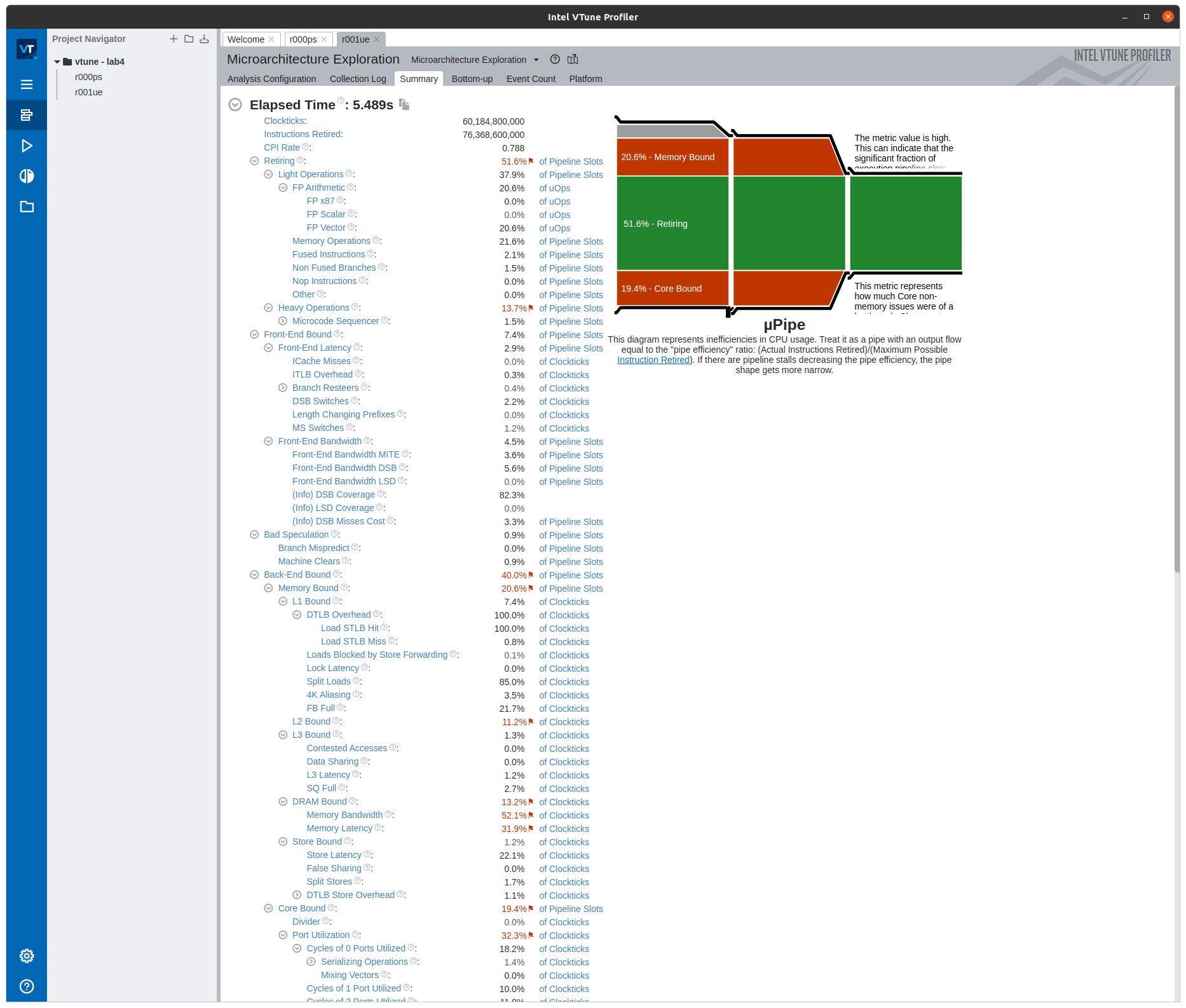
Характеристика исполнения программы:

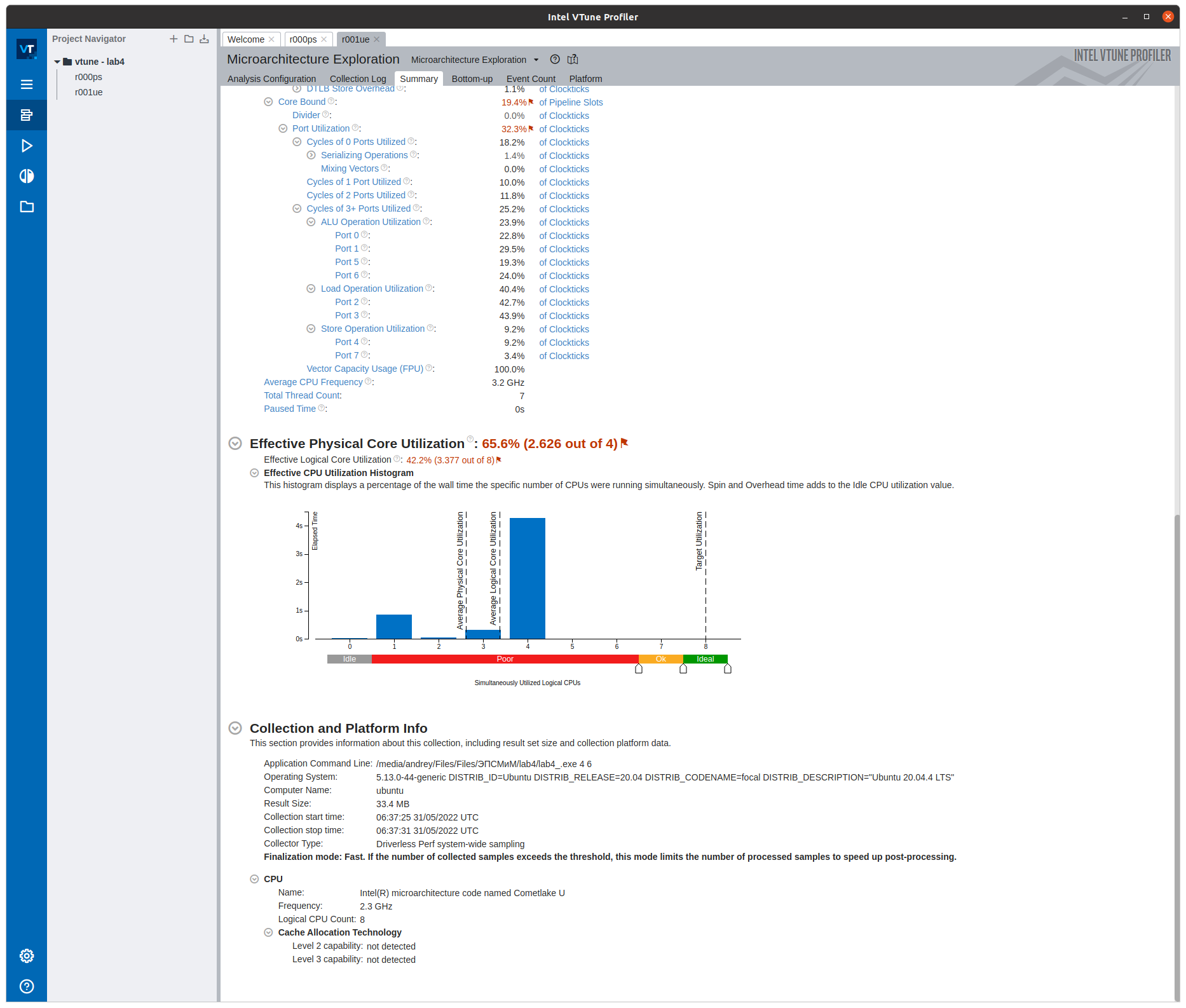




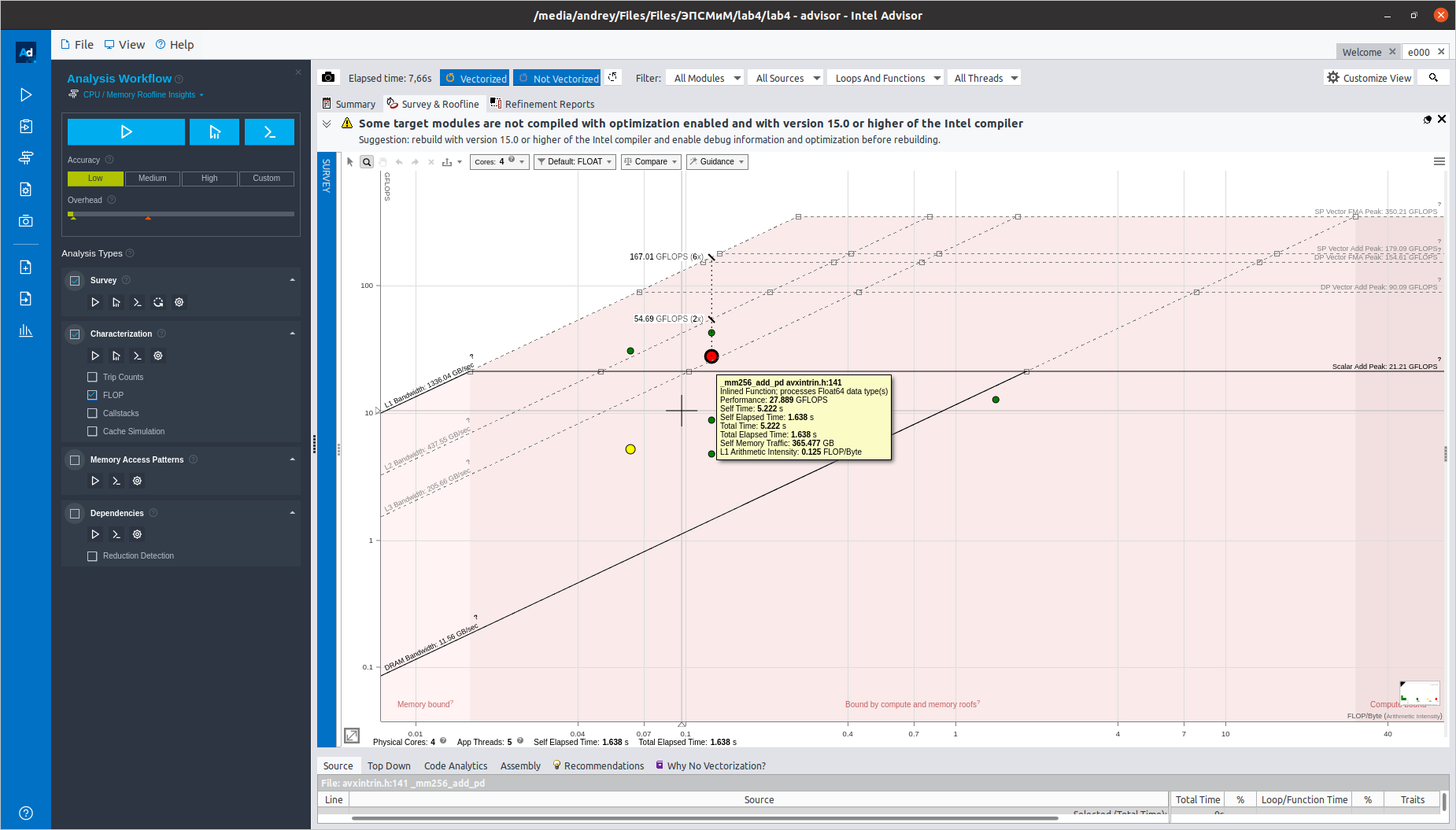


Скриншоты из Vtune:





Roofline модель



Приложение 1

Код программы lab4\_.c:

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <math.h>

#include <time.h>

#include <immintrin.h>

#include <pthread.h>

#pragma warning(disable : 4996)

//#define threads 6

int r = 1;

int threads = 1;

pthread\_t\* thread;

int to\_a = 1;

double\*\* F;

double\*\* ro;

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;

double Y\_ba;

double h\_x;

double h\_y;

double X\_s1;

double Y\_s1;

double X\_s2;

double Y\_s2;

double h\_x2;

double h\_y2;

double \_h\_x2\_h\_y2;

double X\_h\_x2\_h\_y2;

double Y\_h\_x2\_h\_y2;

double koef\_1\_5;

double koef\_1\_4;

\_\_m256d v\_\_X\_h\_x2\_h\_y2;

\_\_m256d v\_\_Y\_h\_x2\_h\_y2;

\_\_m256d v\_\_koef\_1\_5;

\_\_m256d v\_\_koef\_1\_4;

int\* green;

int\* red;

int\* blue;

volatile int green\_wait\_blue[16 \* 32];

volatile int blue\_wait\_green[16 \* 32];

//int count\_cells[threads] = {0};

//int c1[threads] = {0};

//int c2[threads] = {0};

//int c3[threads] = {0};

pthread\_barrier\_t barrier;

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 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, int i, int j, int thread\_id)

{

//count\_cells[thread\_id]++;

\_\_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\_load\_pd(F\_old + i \* Nx + j - 1);

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

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

result1\_2 = \_mm256\_mul\_pd(v\_\_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\_load\_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(v\_\_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\_load\_pd(F\_old + i \* Nx + j - Nx - 1);

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

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

fouth3 = \_mm256\_load\_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(v\_\_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(v\_\_koef\_1\_5, result5\_1);

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

}

//#define pthread\_barrier\_wait(x)

void one\_thread(int thread\_id)

{

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

{

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

{

for (int p = green[thread\_id \* 16] + (k - 1), l = 1; l <= k; p--, l++)

{

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

{

//c1[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

}

}

}

for (int p = red[thread\_id \* 16]; p <= blue[thread\_id \* 16]; p++)

{

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

{

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

{

//c2[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p - (l - 1), j, thread\_id);

}

}

}

pthread\_barrier\_wait(&barrier);

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

{

for (int p = blue[thread\_id \* 16], l = k; l <= r; p--, l++)

{

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

{

//c3[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

}

}

}

pthread\_barrier\_wait(&barrier);

}

}

void first(int thread\_id)

{

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

{

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

{

for (int p = green[thread\_id \* 16] + (k - 1), l = 1; l <= k; p--, l++)

{

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

{

//c1[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

}

}

}

for (int p = red[thread\_id \* 16]; p <= blue[thread\_id \* 16]; p++)

{

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

{

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

{

//c2[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p - (l - 1), j, thread\_id);

}

}

}

while(blue\_wait\_green[(thread\_id + 1) \* 16] < it);

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

{

for (int p = blue[thread\_id \* 16] + (k - 2), l = k; l <= r; p--, l++)

{

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

{

//c3[thread\_id] += 2;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p + 1, j, thread\_id);

}

}

}

green\_wait\_blue[thread\_id \* 16] = it;

}

}

void middle(int thread\_id)

{

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

{

while (green\_wait\_blue[(thread\_id - 1) \* 16] < it - r);

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

{

for (int p = green[thread\_id \* 16] + (k - 1) \* 2, l = 1; l <= k; p--, l++)

{

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

{

//c1[thread\_id] += 2;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p + 1, j, thread\_id);

}

}

}

blue\_wait\_green[thread\_id \* 16] = it;

for (int p = red[thread\_id \* 16]; p <= blue[thread\_id \* 16]; p++)

{

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

{

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

{

//c2[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p - (l - 1), j, thread\_id);

}

}

}

while(blue\_wait\_green[(thread\_id + 1) \* 16] < it);

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

{

for (int p = blue[thread\_id \* 16] + (k - 2), l = k; l <= r; p--, l++)

{

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

{

//c3[thread\_id] += 2;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p + 1, j, thread\_id);

}

}

}

green\_wait\_blue[thread\_id \* 16] = it;

}

}

void last(int thread\_id)

{

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

{

while (green\_wait\_blue[(thread\_id - 1) \* 16] < it - r);

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

{

for (int p = green[thread\_id \* 16] + (k - 1) \* 2, l = 1; l <= k; p--, l++)

{

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

{

//c1[thread\_id] += 2;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p + 1, j, thread\_id);

}

}

}

blue\_wait\_green[thread\_id \* 16] = it;

for (int p = red[thread\_id \* 16]; p <= blue[thread\_id \* 16]; p++)

{

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

{

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

{

//c2[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p - (l - 1), j, thread\_id);

}

}

}

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

{

for (int p = blue[thread\_id \* 16], l = k; l <= r; p--, l++)

{

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

{

//c3[thread\_id]++;

mem\_v\_step(F[(it + l) % 2], F[(it + l - 1) % 2], ro[1], p, j, thread\_id);

}

}

}

}

}

void\* new\_thread(void\* param)

{

int thread\_id = \*((int\*)param);

free((int\*)param);

//printf("[ %d | %d | new\_thread()] Thread started!\n", thread\_id, (int)pthread\_self());

if (threads == 1)

{

one\_thread(thread\_id);

}

else if (thread\_id == 0)

{

first(thread\_id);

}

else if (thread\_id == threads - 1)

{

last(thread\_id);

}

else

{

middle(thread\_id);

}

//printf("[ %d ] finish\n", thread\_id);

pthread\_exit(NULL);

}

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

{

if (argv[1] != NULL)

{

threads = atoi(argv[1]);

}

if ((argv[1] != NULL) && (argv[2] != NULL))

{

r = atoi(argv[2]);

}

if ((argv[1] != NULL) && (argv[2] != NULL) && (argv[3] != NULL))

{

to\_a = atoi(argv[3]);

}

X\_ba = Xb - Xa;

Y\_ba = Yb - Ya;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

double delta = 999999999.0;

double delta\_new;

double best\_time = 9999.9;

for (int a = 0; a < to\_a; a++)

{

printf("atomic barrier");

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

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();

green = (int\*)calloc(threads \* 16, sizeof(int));

red = (int\*)calloc(threads \* 16, sizeof(int));

blue = (int\*)calloc(threads \* 16, sizeof(int));

memset(green, 0, threads \* 16\* sizeof(int));

memset(red, 0, threads \* 16 \* sizeof(int));

memset(blue, 0, threads \* 16 \* sizeof(int));

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

{

green[i \* 16] = i \* (Ny / threads) - (r - 1);

red[i \* 16] = i \* (Ny / threads) + (r - 1);

blue[i \* 16] = (i + 1) \* (Ny / threads) - r;

}

green[0] = 1;

red[0]++;

blue[(threads - 1) \* 16] = Ny - 2;

//green\_wait\_blue = (int\*)calloc(threads \* 16, sizeof(volatile int));

//blue\_wait\_green = (int\*)calloc(threads \* 16, sizeof(volatile int));

//memset(green\_wait\_blue, 0, threads \* 16 \* sizeof(volatile int));

//memset(blue\_wait\_green, 0, threads \* 16 \* sizeof(volatile int));

for (int i = 0; i < 16 \* 32; i++)

{

green\_wait\_blue[i] = 0;

blue\_wait\_green[i] = 0;

}

/\*

printf("green: ");

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

{

printf("%d; ", green[i]);

}

printf("\n");

printf("red: ");

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

{

printf("%d; ", red[i]);

}

printf("\n");

printf("blue: ");

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

{

printf("%d; ", blue[i]);

}

printf("\n");

\*/

pthread\_barrier\_init(&barrier, NULL, threads);

struct timespec start, finish;

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

//pthread\_t thread[threads]; //= (pthread\_t\*)calloc(thread\_pool, sizeof(pthread\_t));

thread = (pthread\_t\*)calloc(threads, sizeof(pthread\_t));

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

{

int\* thread\_ID = (int\*)malloc(sizeof(int));

(\*thread\_ID) = i;

int error\_code = pthread\_create(&thread[i], NULL, new\_thread, thread\_ID);

if (error\_code != 0)

{

printf("[MAIN | Waring] Error in create thread.");

}

}

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

{

pthread\_join(thread[i], NULL);

}

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

int it = Nt;

if (Nt % r != 0)

{

int a = Nt / r;

it = a \* r + r;

}

/\*

int t\_count\_cells = 0, t\_c1 = 0, t\_c2 = 0, t\_c3 = 0;

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

{

t\_count\_cells += count\_cells[i];

t\_c1 += c1[i];

t\_c2 += c2[i];

t\_c3 += c3[i];

}

\*/

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

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

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

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

//int true\_count = t\_count\_cells / it \* Nt;

//printf("threads = %d; Nx = %d, Ny = %d, Nt = %d, r = %d; it = %d; time = %lf; true time = %lf; total count = %d; true count = %d; count 1 = %d, count 2 = %d, count 3 = %d\n", threads, Nx, Ny, Nt, r, it, time, true\_time, t\_count\_cells, true\_count, t\_c1, t\_c2, t\_c3);

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

pthread\_barrier\_destroy(&barrier);

free(F[0]);

free(F[1]);

free(ro[0]);

free(ro[1]);

free(green);

free(red);

free(blue);

//free(blue\_wait\_green);

//free(green\_wait\_blue);

free(thread);

if (true\_time < best\_time)

{

best\_time = true\_time;

}

}

FILE\* stat = fopen("statistic.txt", "a");

fprintf(stat, "%lf; ", best\_time);

fclose(stat);

printf("threads = %d; r = %d; Nx = %d; Ny = %d; Nt = %d; a = %d; best time = %lf\n", threads, r, Nx, Ny, Nt, to\_a, best\_time);

return 0;

}

Вывод:

Программа стала больше упираться в память и в процессор, относительно предыдущего задания. Однако сохраняется достаточная большая зелёная зона «Retiring» в Vtune. Порты загружены в целом равномерно, больше всего это «load operation», так как используется 8-и точечный шаблон. По памяти больше страдает пропускная способность чем латентность. Кэш-память 2-ого уровня тоже подсвечивается красным цветном. Скорее всего всё необходимое не взлезает в 1-ый кэш. На 3-ий нагрузка всего 1%.

Кстати, кэш промахи на первом и на последнем уровнях достаточно низкие, относительно предыдущих заданиях. Видимо от того, что подобраны оптимальные параметры количества потоков и размер лестницы.

Roofline модель хорошо выросла относительно производительности.

Были проведены исследования на разных количествах потоках и на разных количествах размеров лестницы. Лучшие результаты получались на 3 и 4 потоках от 4 до 8 размер лестницы.

Если подводить общий итог, то с 1-ого задания из 23 секунд мы приблизились к 3-ем секундам. Причем сам компилятор способен разогнать исходную программу до 10 секунд. А если вообще не проводить никакие оптимизации программа выполняется больше 100 секунд.