МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Эффективное программирование современных микропроцессоров и мультипроцессоров»

(Вариант №3)

Выполнил: студент 3-го курса гр. 17208

Гафиятуллин А.Р

Новосибирск, 2020

1. **ЦЕЛИ РАБОТЫ:**

   Научиться разрабатывать простые программы численного моделирования, применять базовые средства оптимизации программ, выполнять оценку и анализ производительности программ, пользоваться средствами профилирования.

   Вариант №3: *решение уравнения Пуассона методом Якоби на float-ах.* Алгоритм моделирует установление стационарного распределение тепла в пластинке с заданным распределением источников и стоков тепла. В начальный момент времени значения искомой функции на сетке инициализируются нулями. На каждом шаге моделирования значения искомой функции пересчитываются по заданной формуле. $\mathbf{N_x = N_y = 9000, N_t = 110}$.

2. **ХОД РАБОТЫ:**

   2.1. Тестирование происходило на процессоре **Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz (CPU max MHz: 4700.0000 (Turbo Boost))**.

   2.2. Текст 1-го работающего варианта программы **(см. приложение 4.1)**.

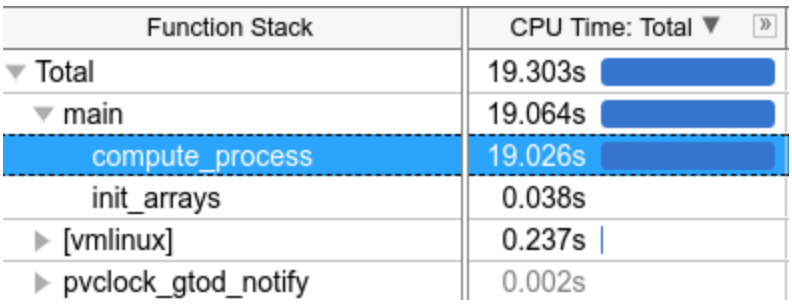
   2.3. Текст самого быстрого варианта программы **(см. приложение 4.2)**.

   2.4. Описание использованных способов оптимизации программы с результатами:

Оптимизация, тип	**Время, сек.**
Без оптимизаций (-O0, алгоритмических оптимизаций нет)	204
Замена дефайна max на inline-функцию max (-O0)	218
Переупорядочено обращение к массивам, чтобы уменьшить количество кэш-промахов (-O0)	200
-O1	40
-O2	24
-O2 -ip -xcoffeelake -axcoffeelake	22

Оптимизация, тип	Время, сек.
Переиспользование полученной из массивов информации на следующих итерациях, чтобы уменьшить количество обращений в кучу	22
Переупорядочивание некоторых команд после многократного получения результатов профилирования	19

2.5. Граф вызовов программы:



Синим выделена горячая точка.

2.6. Аннотированный листинг «горячей точки» программы **(см. приложение 4.3).**

2.7. Аннотированный ассемблерный листинг «горячей точки» программы **(см. приложение 4.4).**

2.8. Характеристики исполнения программы были получены с использованием утилиты perf (**только для загрузок(load)**):

2.8.1. Число инструкций на такт: 3.01;

2.8.2. Процент кэш-промахов для кэша 3 уровня: 44.54%;

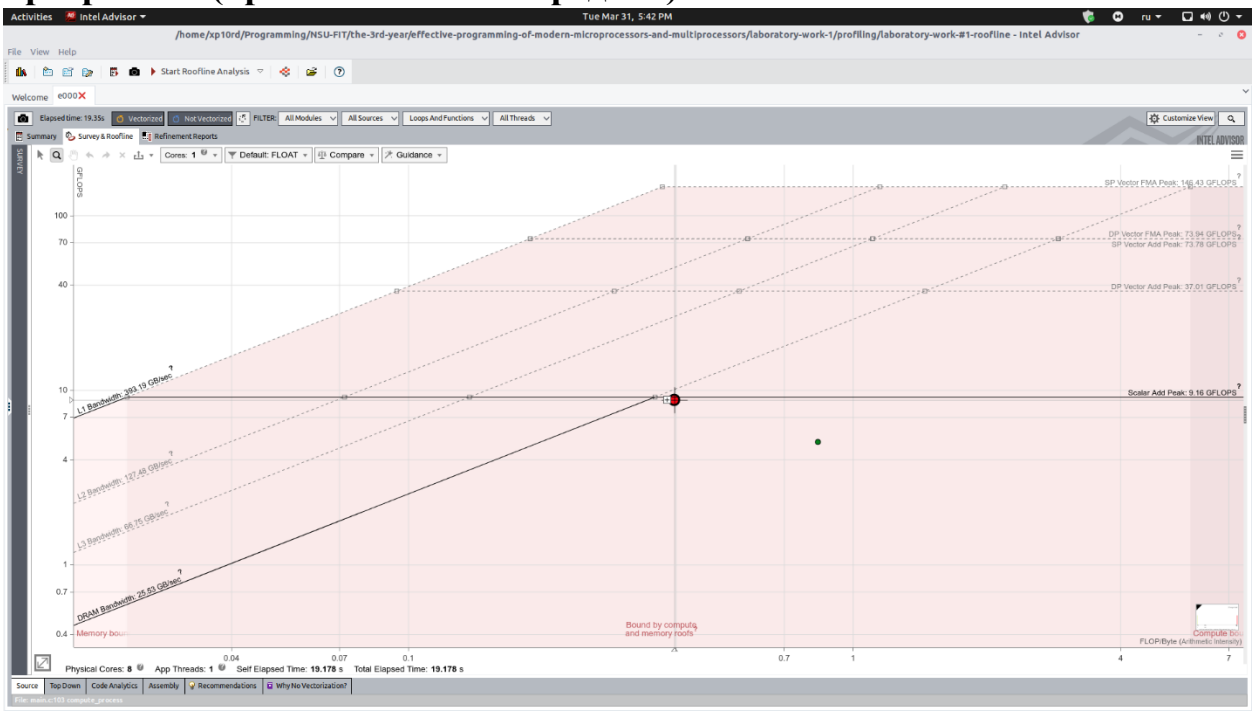
2.8.3. Процент кэш-промахов для кэша-данных 1 уровня: 5.04%;

2.8.4. Процент неправильно предсказанных переходов: 0.03%.

2.8.5. По всей видимости, основной причиной временных затрат являются **вычислительные операции**. Так как процент промахов при обращении к кэшу 1 уровня мал, а количество обращений к кэшу 3 уровня по сравнению с кэшом 1 уровня крайне мало (для одного из замеров: к 3 уровню - **43,019,723** обращений, а к 1

уровню - **71,513,217,255**), то, скорее всего, это особо не влияет на производительность.

2.9. **Roofline-модель с точкой, соответствующей основному циклу программы (красная точка посередине):**



Автоматическое заключение Intel Advisor: производительность основного цикла программы ограничена **вычислительными операциями**.

3. ВЫВОДЫ:

3.1. Научились разрабатывать простые программы численного моделирования, применять базовые средства оптимизации программ, выполнять оценку и анализ производительности программ, пользоваться средствами профилирования.

3.2. На данный момент производительность самой быстрой версии программы ограничена скоростью вычислительных операций.

4. ПРИЛОЖЕНИЕ:

4.1. Текст 1-го работающего варианта программы.

```
#define comp_type float

#define Nx  9000
#define Ny  9000
#define Nt  111

#define Xa  (comp_type)0.0
#define Xb  (comp_type)4.0
```

```
#define Ya  (comp_type)0.0
#define Yb  (comp_type)4.0

#define hx (comp_type)((Xb - Xa) / (Nx - 1))
#define hy (comp_type)((Yb - Ya) / (Ny - 1))
#define coeff1 ((comp_type)0.2 / ((comp_type)1.0 / (hx * hx) + (comp_type)1.0
/ (hy * hy)))
#define coeff2 ((comp_type)0.5 * ((comp_type)5.0 / (hx * hx) - (comp_type)1.0
/ (hy * hy)))
#define coeff3 ((comp_type)0.25 * ((comp_type)1.0 / (hx * hx) +
(comp_type)1.0 / (hy * hy)))
#define X(j) (Xa + (j) * hx)
#define Y(i) (Ya + (i) * hy)

#define Xs1 (Xa + (Xb - Xa) / (comp_type)3.0)
#define Xs2 (Xa + (Xb - Xa) * (comp_type)2.0 / (comp_type)3.0)
#define Ys1 (Ya + (Yb - Ya) * (comp_type)2.0 / (comp_type)3.0)
#define Ys2 (Ya + (Yb - Ya) / (comp_type)3.0)
#define R ((comp_type)0.1 * ((Xb - Xa) > (Yb - Ya) ? (Yb - Ya) : (Xb - Xa)))

#define GRID_SIZE (Nx * Ny)
#define TIME_LAYERS 2

#define get3(F, n, i, j)    F[(n) * GRID_SIZE + (i) * Nx + (j)]
#define get2(p, i, j)       p[(i) * Nx + (j)]

#define max(a,b) ((a) > (b) ? (a) : (b))

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>

int main() {
    /* allocate memory */
    comp_type* F = malloc(TIME_LAYERS * GRID_SIZE * sizeof(comp_type));
    comp_type *p = malloc(GRID_SIZE * sizeof(comp_type));
    if(!F || !p) {
        perror("malloc");
        exit(errno);
    }

    /* init arrays */
    for(int i = 0; i < GRID_SIZE; i++) {
        F[i] = (comp_type)0.0;
    }
    for(int i = 0; i < Ny; i++) {
        for(int j = 0; j < Nx; j++) {
            if((X(j) - Xs1) * (X(j) - Xs1) + (Y(i) - Ys1) * (Y(i) - Ys1) < R
* R) {
                get2(p, i, j) = (comp_type)0.1;
            } else if((X(j) - Xs2) * (X(j) - Xs2) + (Y(i) - Ys2) * (Y(i) -
Ys2) < R * R) {
                get2(p, i, j) = (comp_type)-0.1;
```

```
            } else {
                get2(p, i, j) = (comp_type)0.0;
            }
        }
    }

    /* compute process */
    time_t start_time = time(NULL);

    comp_type delta = 0;
    for(int n = 0; n < Nt - 1; n++) {
        delta = 0;
        for(int i = 1; i < Ny - 1; i++) {
            for(int j = 1; j < Nx - 1; j++) {
                int n_idx = n % 2;
                int nplus1_idx = (n + 1) % 2;
                get3(F, nplus1_idx, i, j) = coeff1 * (
                        coeff2 * (get3(F, n_idx, i, j - 1) + get3(F, n_idx,
i, j + 1))  +
                        coeff2 * (get3(F, n_idx, i - 1, j) + get3(F, n_idx, i
+ 1, j))  +
                        coeff3 * (get3(F, n_idx, i - 1, j - 1)      +
                                get3(F, n_idx,  i - 1, j + 1)   +
                                get3(F, n_idx, i + 1, j - 1)    +
                                get3(F, n_idx, i + 1, j + 1))   +
                        2.0f * get2(p, i, j)            +
                        0.25f * (get2(p, i - 1, j)      +
                                get2(p, i + 1, j)   +
                                get2(p, i, j - 1)   +
                                get2(p, i, j + 1)));
                delta = max(delta, fabs(get3(F, n_idx, i, j) - get3(F,
nplus1_idx, i, j)));
            }
        }
    }
    time_t end_time = time(NULL);
    printf("n = %d, sigma = %.8f\n", Nt - 1, delta);
    printf("Total time: %lld sec.\n", end_time - start_time);
}
```

4.2. Текст самого быстрого варианта программы.

```
#define Nx  9000
#define Ny  9000
#define Nt  111

#define Xa  0.0f
#define Xb  4.0f
#define Ya  0.0f
#define Yb  4.0f

#define hx ((Xb - Xa) / (Nx - 1))
#define hy ((Yb - Ya) / (Ny - 1))
#define coeff1 (0.2f / ((1.0f / (hx * hx) + 1.0f / (hy * hy))))
#define coeff2 (0.5f * (5.0f / (hx * hx) - 1.0f / (hy * hy)))
#define coeff2b (0.5f * (5.0f / (hy * hy) - 1.0f / (hx * hx)))
#define coeff3 (0.25f * (1.0f / (hx * hx) + 1.0f / (hy * hy)))
```

```
#define X(j) (Xa + (j) * hx)
#define Y(i) (Ya + (i) * hy)

#define Xs1 (Xa + (Xb - Xa) / 3.0f)
#define Xs2 (Xa + (Xb - Xa) * 2.0f / 3.0f)
#define Ys1 (Ya + (Yb - Ya) * 2.0f / 3.0f)
#define Ys2 (Ya + (Yb - Ya) / 3.0f)
#define R (0.1f * ((Xb - Xa) > (Yb - Ya) ? (Yb - Ya) : (Xb - Xa)))

#define GRID_SIZE (Nx * Ny)
#define TIME_LAYERS 2

#define get3(F, n, i, j)    F[(n) * GRID_SIZE + (i) * Nx + (j)]
#define get2(p, i, j)       p[(i) * Nx + (j)]

#define max(a, b) ((a) > (b) ? (a) : (b))

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>

void init_arrays(float *F, float  *p);
float compute_process(float *F, float *p);

int main() {
    /* allocate memory */
    float* F = malloc(TIME_LAYERS * GRID_SIZE * sizeof(float));
    float *p = malloc(GRID_SIZE * sizeof(float));
    if(!F || !p) {
        perror("malloc");
        exit(errno);
    }

    init_arrays(F, p);
    time_t start_time = time(NULL);
    float delta = compute_process(F, p);
    time_t end_time = time(NULL);

    printf("n = %d, sigma = %.8f\n", Nt - 1, delta);
    printf("Total time: %ld sec.\n", end_time - start_time);
}

void init_arrays(float *F, float  *p) {
    for(int i = 0; i < Ny; i++) {
        for(int j = 0; j < Nx; j++) {
            float xj = X(j);
            float yi = Y(i);
            if((xj - Xs1) * (xj - Xs1) + (yi - Ys1) * (yi - Ys1) < R * R) {
                get2(p, i, j) = 0.1f;
            } else if((xj - Xs2) * (xj - Xs2) + (yi - Ys2) * (yi - Ys2) < R *
R) {
                get2(p, i, j) = -0.1f;
            } else {
```

```
                get2(p, i, j) = 0.0f;
            }
            get2(F, i, j) = 0.0f;
        }
    }
}

float compute_process(float *F, float *p) {
    float delta = 0;
    for(int n = 0; n < Nt - 1; n++) {
        int n_idx = n % 2;
        int nplus1_idx = (n + 1) % 2;

        delta = 0;
        for(int i = 1; i < Ny - 1; i++) {
            float f_left_down_cell = get3(F, n_idx, i - 1, 0);
            float f_current_down_cell = get3(F, n_idx, i - 1, 1);
            float f_right_down_cell = get3(F, n_idx,  i - 1, 2);

            float f_left_current_cell = get3(F, n_idx, i, 0);
            float f_current_current_cell = get3(F, n_idx, i, 1);
            float f_right_current_cell = get3(F, n_idx, i, 2);

            float f_left_up_cell = get3(F, n_idx, i + 1, 0);
            float f_current_up_cell = get3(F, n_idx, i + 1, 1);
            float f_right_up_cell = get3(F, n_idx, i + 1, 2);

            float p_left_current_cell = get2(p, i, 0);
            float p_current_current_cell = get2(p, i, 1);
            float p_right_current_cell = get2(p, i, 2);

            for(register int j = 1; j < Nx - 1; j++) {
                float rez = coeff1 * (
                        coeff2b * (f_current_down_cell + f_current_up_cell)
+
                        coeff3 * (f_left_up_cell + f_right_up_cell +
f_left_down_cell + f_right_down_cell)  +
                        coeff2 * (f_left_current_cell + f_right_current_cell)
+
                        0.25f * (
                        2.0f * p_current_current_cell       +
                        p_left_current_cell             +
                        p_right_current_cell)           +
                        0.25f * get2(p, i - 1, j)       +
                        get2(p, i + 1, j));

                get3(F, nplus1_idx, i, j) = rez;
                delta = max(delta, fabs(f_current_current_cell - rez));

                f_left_down_cell = f_current_down_cell;
                f_current_down_cell = f_right_down_cell;
                f_left_up_cell = f_current_up_cell;
                f_current_up_cell = f_right_up_cell;
                f_left_current_cell = f_current_current_cell;
                f_current_current_cell = f_right_current_cell;
                f_right_down_cell = get3(F, n_idx,  i - 1, j + 2);
                f_right_current_cell = get3(F, n_idx,  i, j + 2);
```

```
                f_right_up_cell = get3(F, n_idx, i + 1, j + 2);

                p_left_current_cell = p_current_current_cell;
                p_current_current_cell = p_right_current_cell;
                p_right_current_cell = get2(p, i, j + 2);
            }
        }
    }

    return (delta);
}
```

4.3. Аннотированный ассемблерный листинг «горячей точки» программы **(ссылка на картинку кликабельна, сохранена локально в документе)**:

hotspot-cpl.png

синим отмечена самая «горячая» точка.

4.4. Аннотированный ассемблерный листинг «горячей точки» программы: **(ссылка на картинку кликабельна, сохранена локально в документе)**:

hotspot-assembly.png

синим отмечены команды, соответствующие самой «горячей» точке.