Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 2 по дисциплине "Параллельные вычисления"

Выполнили:

Айтуганов Д. А.
Чебыкин И. Б.

Группа: P42111, P42101

Проверяющий: Балакшин П. В.

Санкт-Петербург, 2019

Цель работы

В исходном коде программы, полученной в результате выполнения лабораторной работы No1, нужно на этапах Map и Merge все циклы с вызовами математических функций заменить их векторными ана- логами из библиотеки «AMD Framewave» (http://framewave. sourceforge.net). При выборе конкретной Framewave-функции необходимо убедиться, что она помечена как MT (Multi-Threaded), т.е. распараллеленная. Полный перечень доступных функций на- ходится по ссылке: http://framewave.sourceforge.net/ Manual/fw_section_060.html#fw_section_060. Например, Framewave-функция min в списке поддерживаемых технологий име- ет только SSE2, но не MT.

Конфигурация

Процессор

```
CPU(s):                          16
Thread(s) per core:              1
Core(s) per socket:              8
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
Model name:                      AMD Ryzen 7 1700 Eight-Core Processor
CPU MHz:                         2645.861
CPU max MHz:                     3000.0000
CPU min MHz:                     1550.0000
```

RAM: 32 GB

Компиляторы

```
gcc (GCC) 9.1.0
clang version 8.0.0
icc (ICC) 19.0.5.281 20190815
```

Исходный код

```
#include "fwBase.h"
#include "fwSignal.h"

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

const static int c_a = 567;
const static int c_experiments = 50;

void generate(unsigned int seed, double *p, unsigned int N,
```

```
              unsigned int min, unsigned int max) {
    unsigned int i;
    for (i = 0; i < N; i++) {
        p[i] = (rand_r(&seed) % max) + min;
    }
}

void lab_swap(double * lhs, double * rhs) {
    double tmp = *lhs;
    *lhs = *rhs;
    *rhs = tmp;
}

int correct(double *arr, int n) {
    while (n-- > 0) {
        if (arr[n - 1] > arr[n]) {
            return 0;
        }
    }
    return 1;
}

void shuffle(double *arr, int n) {
    int i;
    for (i = 0; i < n; i++) {
        lab_swap(&arr[i], &arr[(rand() % n)]);
    }
}

void bogo_sort(double *arr, int n) {
    while (!correct(arr, n))
        shuffle(arr, n);
}

double lab_abs(double v) {
    if (v < 0) {
        return -v;
    }
    return v;
}

double lab_min(double lhs, double rhs) {
    return lhs > rhs ? rhs : lhs;
}

double lab_max(double lhs, double rhs) {
    return lhs < rhs ? rhs : lhs;
}

double lab_cot(double val) {
    return cos(val) / sin(val);
}

double lab_coth(double val) {
    return cosh(val) / sinh(val);
}
```

```
void print_array(double *p, unsigned int N) {
    unsigned int i = 0;
    for (i = 0; i < N - 1; i++) {
        printf("%f ", p[i]);
    }
    printf("%f\n", p[N - 1]);
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        puts("lab2-par <N size of problem> <M number of threads>");
        return -1;
    }

    struct timeval begin, end;
    struct timeval par_begin, par_end;
    const int N = atoi(argv[1]);
    if (N < 0) {
        return -2;
    }

    const int M = atoi(argv[2]);
    if (M < 0) {
        return -2;
    }
    fwSetNumThreads(M);
    double * m1 = malloc(sizeof(double) * N);
    double * m1_sqrt_dst = malloc(sizeof(double) * N);
    double * m1_cosh_dst = malloc(sizeof(double) * N);
    double * m1_sinh_dst = malloc(sizeof(double) * N);
    double * m1_dst = malloc(sizeof(double) * N);

    double * m2 = malloc(sizeof(double) * N / 2);
    double * m2_cos_dst = malloc(sizeof(double) * N / 2);
    double * m2_sin_dst = malloc(sizeof(double) * N / 2);
    double * m2_dst = malloc(sizeof(double) * N / 2);

    gettimeofday(&begin, NULL);
    double reduced_sum = 0.0;
    unsigned int i;
    for (i = 0; i < c_experiments; i++) {
        // 1. Generate: M1 of N elements, M2 of N/2 elements
        generate(i, m1, N, 1, c_a);
        //puts("M1");
        //print_array(m1, N);
        generate(i, m2, N / 2, c_a, 10 * c_a);
        //puts("M2");
        //print_array(m2, N / 2);
        // 2. Map: coth(sqrt(M1[j])) ; M2[j] = abs(cot(M2[j]))
        gettimeofday(&par_begin, NULL);
        unsigned int j;
        fwsSqrt_64f_A53(m1, m1_sqrt_dst, N);
        fwsCosh_64f_A53(m1_sqrt_dst, m1_cosh_dst, N);
        fwsSinh_64f_A53(m1_sqrt_dst, m1_sinh_dst, N);
        fwsDiv_64f_A53(m1_cosh_dst, m1_sinh_dst, m1_dst, N);
```

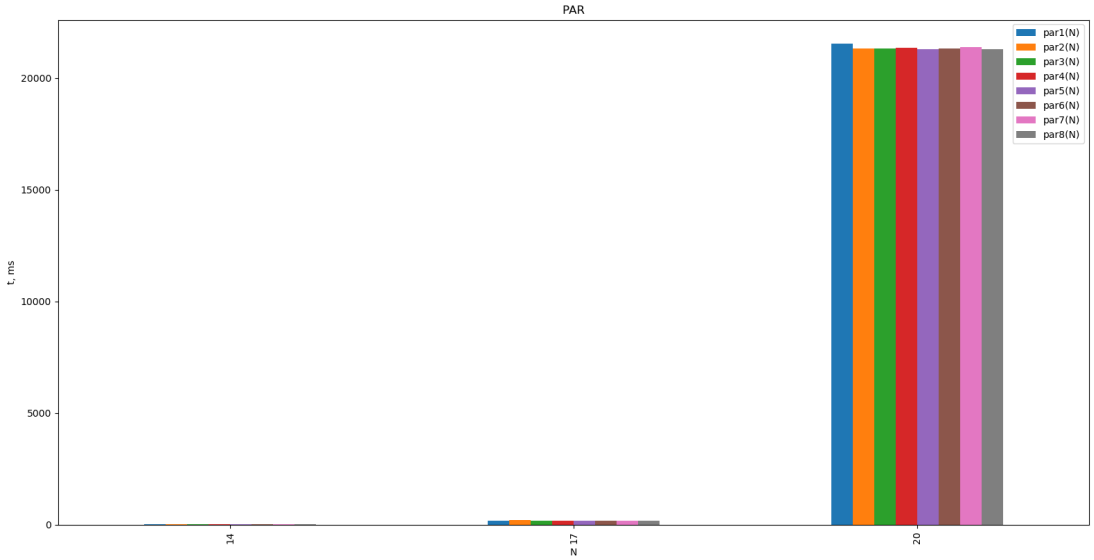
```
        for (j = 0; j < N; j++) {
            m1[j] = m1_dst[j];
        }
        //puts("M1 coth");
        //print_array(m1, N);
        //m2[0] = lab_abs(cot(m2[0] [> + 0.0 <]));
        fwsCos_64f_A53(m2, m2_cos_dst, N / 2);
        fwsSin_64f_A53(m2, m2_sin_dst, N / 2);
        fwsDiv_64f_A53(m2_cos_dst, m2_sin_dst, m2_dst, N / 2);
        for (j = 0; j < N / 2; j++) {
            m2[j] = lab_abs(m2_dst[j]);
        }
        gettimeofday(&par_end, NULL);
     double par_delta_ms = 1000.0 * (par_end.tv_sec - par_begin.tv_sec) + (par_end.tv_usec - par_begin.tv_usec) / 1
        printf("parallel milliseconds passed: %lf\n", par_delta_ms);
        //puts("M2 abs cot");
        //print_array(m2, N / 2);
        // 3. Merge: M2[j] = max(M1[j], M2[j]) , j e N/2
        //fwsMaxEvery_32f_I(m1)
        for (j = 0; j < N / 2; j++) {
            m2[j] = lab_max(m1[j], m2[j]);
        }
        //puts("max of M1 M2");
        //print_array(m2, N / 2);
        // 4. Sort: gnome_sort(M2, N/2)
        bogo_sort(m2, N/2);
        //puts("sorted");
        //print_array(m2, N / 2);
        // 5. Reduce: 1. min_non_zero(M2)
        //            2. if (((long)(M2[j] / min_non_zero)) & ~(1))
        //                   sum += sin(M2[j])
        double min_non_zero = DBL_MAX;
        for (j = 0; j < N / 2; j++) {
            if (m2[j] != 0) {
                min_non_zero = lab_min(min_non_zero, m2[j]);
            }
        }
        //printf("Min non zero: %f\n", min_non_zero);
        for (j = 0; j < N / 2; j++) {
            if (((long)(m2[j] / min_non_zero)) & ~(1)) {
                reduced_sum += sin(m2[j]);
            }
        }
        //printf("Sum: %e\n", reduced_sum);
    }
    gettimeofday(&end, NULL);
    long delta_ms = 1000 * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec) / 1000;
    printf("N = %d. milliseconds passed: %ld\n", N, delta_ms);
    printf("N = %d. X=%e\n", N, reduced_sum / c_experiments);

    return 0;
}
```

Результаты

Выполнение при $M <= K$ потоках

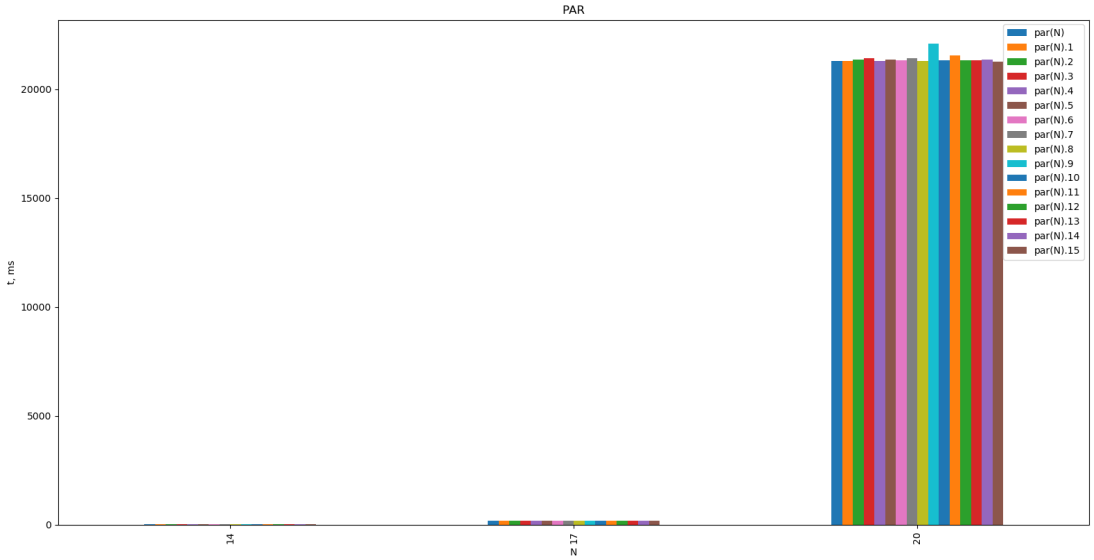
N	par1(N)	par2(N)	par3(N)	par4(N)	par5(N)	par6(N)	par7(N)	par8(N)
14	23	22	23	23	22	23	22	21
17	177	202	175	175	193	177	174	176
20	21531	21317	21327	21361	21297	21321	21373	21291



.

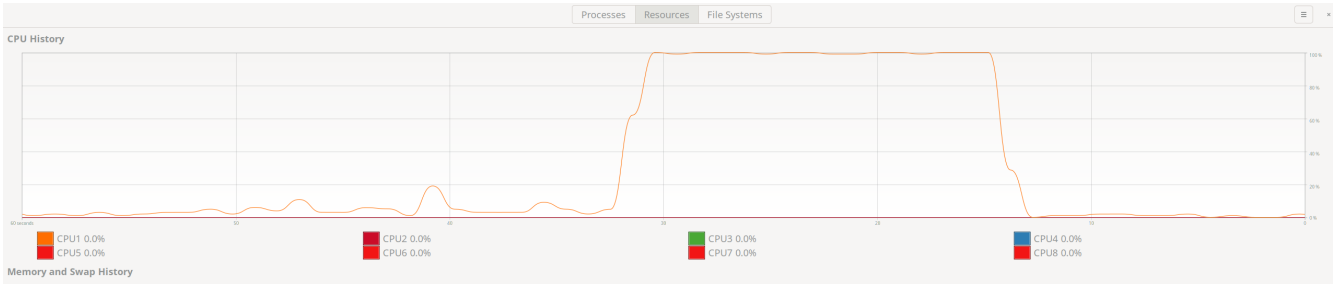
Выполнение при $M > K$ потоках

N	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	par(N)	p
14	22	23	21	23	21	22	23	22	23	23	22	24	2
17	177	195	177	176	176	176	176	180	196	176	177	175	2
20	21286	21300	21377	21413	21303	21361	21331	21431	21299	22088	21330	21561	2



.

Загрузка процессора



.

Расчет закона Амдала

Время выполнения программы (ms):

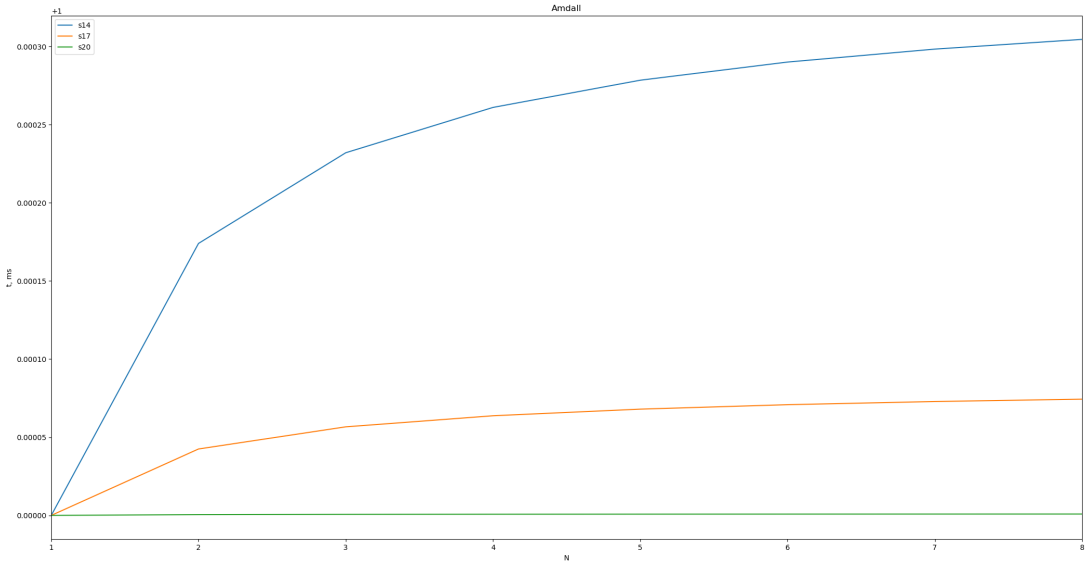
N	t
14	23
17	176
20	21589

Время выполнения распараллеленной части программы (ms):

N	t
14	0.008

N	t
17	0.015
20	0.021

N	k
14	0.000348
17	0.000085
20	0.000001



.

Выводы

После выполнения лабораторной работы можно сказать, что распараллеливание данной программы не приносит существенного эффекта.