Университет ИТМО

Отчет
по лабораторной работе №1
«Автоматическое распараллеливание программ»

Параллельные вычисления

Выполнили:
студенты гр.P42142
Муратова У.
Туров В.

Преподаватель:
Балакшин П.В.

Санкт-Петербург — 2020

Оглавление

Описание решаемой задачи

1. На компьютере с многоядерным процессором установить ОС Linux и компилятор GCC версии не ниже 4.7.2. При невозможности установить Linux или отсутствии компьютера с многоядерным процессором можно выполнять лабораторную работу на виртуальной машине.

2. На языке Си написать консольную программу lab1.c, решающую задачу, указанную в п.IV (см. ниже). В программе нельзя использовать библиотечные функции сортировки, выполнения матричных операций и расчёта статистических величин. В программе нельзя использовать библиотечные функции, отсутствующие в стандартных заголовочных файлах stdio.h, stdlib.h, math.h, sys/time.h. Задача должна решаться 50 раз с разными начальными значениями генератора случайных чисел (ГСЧ).

3. Скомпилировать написанную программу без использования автоматического распараллеливания с помощью следующей команды:

/home/user/gcc -O3 -Wall -Werror -o lab1-seq lab1.c

4. Скомпилировать написанную программу, используя встроенное в gcc средство автоматического распараллеливания Graphite с помощью следующей команды “/home/user/gcc -O3 -Wall -Werror -floop- parallelize-all -ftree-parallelize-loops=K lab1.c -o lab1-par-K” (переменной K поочерёдно присвоить хотя бы 4 различных целых значений, выбор обосновать).

5. В результате получится одна нераспараллеленная программа и четыре или более распараллеленных.

6. Закрыть все работающие в операционной системе прикладные программы (включая Winamp, uTorrent, браузеры и Skype), чтобы они не влияли на результаты последующих экспериментов.

7. Запускать файл lab1-seq из командной строки, увеличивая значения N до значения N1, при котором время выполнения превысит 0.01с. Подобным образом найти значение N=N2, при котором время выполнения превысит 2 с.

8. Используя найденные значения N1 и N2, выполнить следующие эксперименты (для автоматизации проведения экспериментов рекомендуется написать скрипт):

- запускать lab1-seq для значений N = N 1, N 1 + Δ, N 1 + 2Δ, N 1 + 3Δ, . . . , N 2 и записывать получающиеся значения времени delta_ms(N) в функцию seq(N);
- запускать lab1-par-K для значенийN = N 1, N 1 + Δ, N 1 + 2Δ, N 1 +3Δ, . . . , N 2 и записывать получающиеся значения времени delta_ms(N) в функцию par − K(N);
- значение Δ выбрать так: Δ = (N 2 − N 1)/10.

N1 = 184
N2 = 1944
Δ = (N 2 − N 1)/10 = (1944 — 184)/10 = 176

Необязательное задание №1 (для получения оценки «четыре» и «пять»). Провести аналогичные описанным эксперименты, используя вместо gcc компилятор Solaris Studio (или любой другой на своё усмотрение). При компиляции следует использовать следующие опции для автоматического распараллеливания: «solarisstudio -cc -O3 -xautopar -xloopinfo lab1.c».

Необязательное задание №2 (для получения оценки «пять»). Это задание выполняется только после выполнения предыдущего пункта. Провести аналогичные описанным эксперименты, используя вместо gcc компилятор Intel ICC (или любой другой на своё усмотрение). В ICC следует при компиляции использовать следующие опции для автоматического распараллеливания: «icc -parallel -par-report -par-threshold K -o lab1-icc-par-K lab1.c». Если ключ «-par-report» не работает в вашей версии компилятора, то желательно использовать более актуальный ключ «-qopt-report-phase=par».

Вариант задания выбирается в соответствии с приведёнными ниже описанием этапов, учитывая, что число A = Ф*И*О, где Ф, И, О означают количество букв в фамилии, имени и отчестве студента. Номер варианта в соответствующих таблицах выбирается по формуле X = 1 + ((A mod 47) mod B), где B – количество элементов в соответствующей таблице, а операция mod означает остаток от деления. Например, при A = 476 и B = 5, получим X = 1 + ((470+6) mod 47) mod 5) = 1 + (6 mod 5) = 2.

Муратова = 8

Ульяна = 6

Дмитриевна = 10

*A = Ф*И*О*

*A = 8*6*10 = 480*

1. Этап Generate. Сформировать массив M1 размерностью N, заполнив его с помощью функции rand_r (нельзя использовать rand) случайными вещественными числами, имеющими равномерный закон распределения в диапазоне от 1 до A (включительно). Аналогично сформировать массив M2 размерностью N/2 со случайными вещественными числами в диапазоне от A до 10*A.

2. Этап Map. В массиве M1 к каждому элементу применить операцию из таблицы:

1+((480 mod 47) mod 7) = 1 + (10 mod 7) = 1+3 = 4

4	Гиперболический котангенс корня числа

Затем в массиве M2 каждый элемент поочерёдно сложить с предыдущим (для этого вам понадобится копия массива M2, из которого нужно будет брать операнды), а к результату сложения применить операцию из таблицы (считать, что для начального элемента массива предыдущий элемент равен нулю):

1+((480 mod 47) mod 8) = 1 + (10 mod 8) = 1+2 = 3

3	Модуль тангенса

3. Этап Merge. В массивах M1 и M2 ко всем элементами с одинаковыми индексами попарно применить операцию из таблицы (результат записать в M2):

1+((480 mod 47) mod 6) = 1 + (10 mod 6) = 1+4 = 5

5	Выбор меньшего

4. Этап Sort. Полученный массив необходимо отсортировать методом, указанным в таблице (для этого нельзя использовать библиотечные функции; можно взять реализацию в виде свободно доступного исходного кода):

4	Глупая сортировка (Stupid sort).

5. Этап Reduce. Рассчитать сумму синусов тех элементов массива M2, которые при делении на минимальный ненулевой элемент массива M2 дают чётное число (при определении чётности учитывать только целую часть числа). Результатом работы программы по окончании пятого этапа должно стать одно число X, которое следует использовать для верификации программы после внесения в неё изменений (например, до и после распараллеливания итоговое число X не должно измениться в пределах погрешности). Значение числа X следует привести в отчёте для различных значений N.

Характеристики инструментов

Аппаратное обеспечение:

- Процессор: Intel Core i7-4702MQ (4C/8T, 2.20 GHz)
- Оперативная память: 4x DDR3-1600 4ГБ (всего 16 ГБ)
- Графический ускоритель: Intel HD Graphics 4600 + Nvidia GeForce GTX 850M

Программное обеспечение:

- ОС: Ubuntu 20.04.1
- Компиляторы:
    - gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
    - icc version 19.1.2.254 (gcc version 9.3.0 compatibility)
    - clang version 10.0.0

Полный текст программы

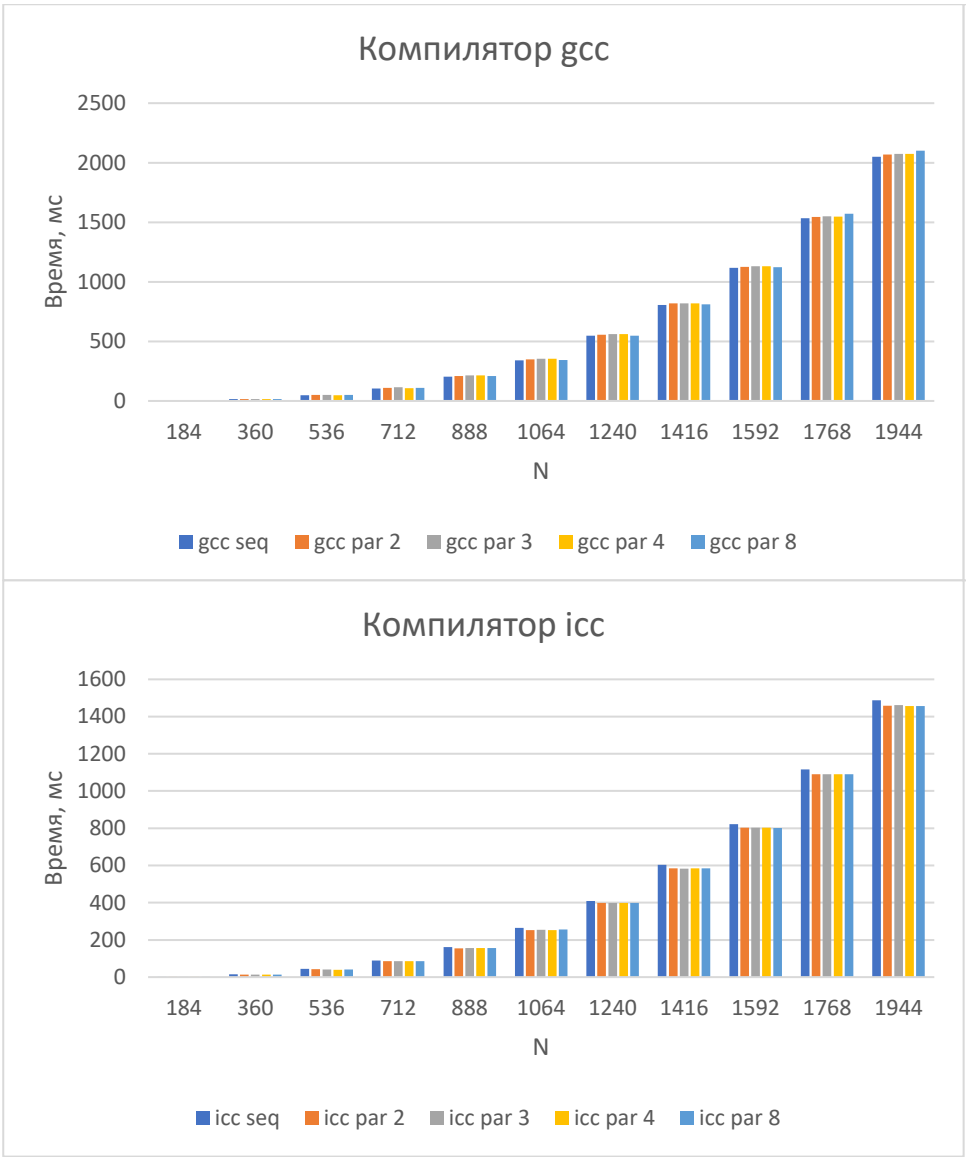
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <math.h>

#define A 480

int main(int argc, char* argv[])
{
    int i, N, j, tmp;
    float X = 0;
    struct timeval T1, T2;
    long delta_ms;
    float e = 0.00001;

    N = atoi(argv[1]); /* N равен первому параметру командной строки */
    gettimeofday(&T1, NULL); /* запомнить текущее время T1 */
    for (i=0; i<50; i++) /* 50 экспериментов */
        {

            srand(i);  /* инициализировать начальное значение ГСЧ   */
            //unsigned int *my_seed = malloc(sizeof(unsigned int));
            unsigned int my_seed[1];
            my_seed[0] = i;
            /* Заполнить массив исходных данных размером N */
            /* Решить поставленную задачу, заполнить массив с результатами*/
            float m1[N];
            for (j=0; j<N; j++)
                {
                    m1[j] = rand_r(my_seed) % A + 1;
                    m1[j] = 1 / tanh(sqrt(m1[j]));
                }

            float m2[N/2];
            float m2_copy[N/2];
            for (j=0; j<N/2; j++)
                {
                    m2[j] = rand_r(my_seed) % (9*A + 1) + A;
                    m2_copy[j] = m2[j];
                    if (j != 0)
                        {
                           m2[j] = m2[j] + m2_copy[j-1];
                        }

                    m2[j] = fabs(tan(m2[j]));
                }


            /* Этап Merge */
            for (j=0; j<N/2; j++)
                {
                    m2[j] = (m1[j] < m2[j]) ? m1[j] : m2[j];
                }

            /* Отсортировать массив с результатами указанным методом */

            j = 0;
            while (j < (N/2) - 1)
            {
                if (m2[j+1] < m2[j])
```

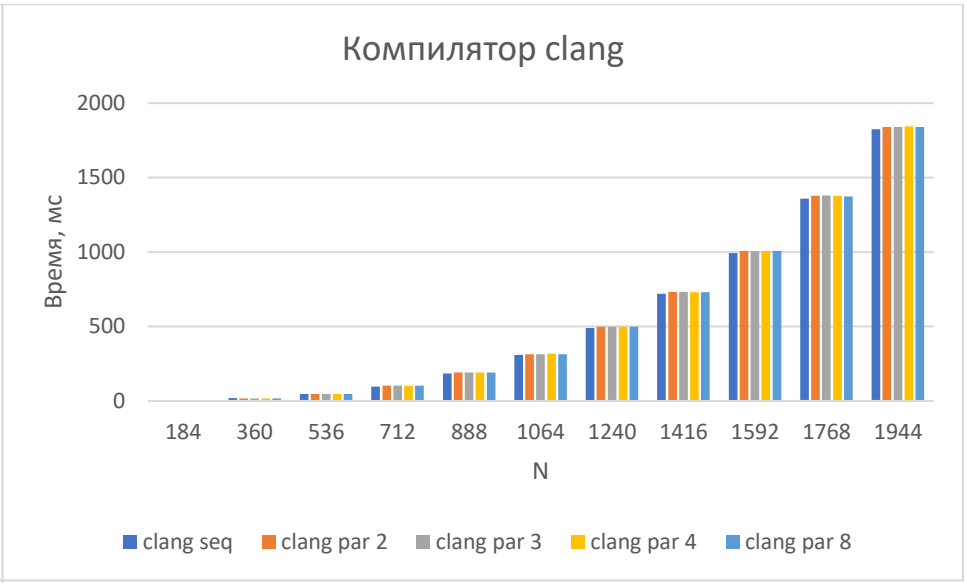
```
                {
                    tmp = m2[j];
                    m2[j] = m2[j+1];
                    m2[j+1] = tmp;
                    j = 0;
                }
                else j++;
            }
            /* Этап Reduce */

            float min = 0;
            j = 0;
            tmp = 0;

            while (min == 0)
                {
                    min = (fabs(m2[j]) < e) ? m2[j] : 0;
                    j++;
                }

            for (j=0; j<N/2; j++)
                {
                    tmp = (int)(m2[j] / min);

                    if (tmp % 2 == 0)
                        {
                            X = X + sin(m2[j]);
                        }
                }

        }
        gettimeofday(&T2, NULL);   /* запомнить текущее время T2 */
        delta_ms =  1000*(T2.tv_sec - T1.tv_sec) + (T2.tv_usec -
T1.tv_usec)/1000;
        printf("\nN=%d. Milliseconds passed: %ld\n", N, delta_ms); /* T2 -T1 */
        printf("\nX: %f\n", X); /* T2 -T1 */
        return 0;
}
```

Таблицы значений и графики функций

Таблица 1. Время выполнения программы в зависимости от N и количества потоков

N		184	360	536	712	888	1064	1240	1416	1592	1768	1944
gcc	seq	8	16	49	105	205	340	547	805	1117	1534	2049
	par 2	3	16	51	110	210	349	555	818	1125	1544	2068
	par 3	3	17	50	116	214	355	560	820	1130	1550	2074
	par 4	3	17	48	108	214	354	561	820	1131	1547	2073
	par 8	3	16	50	110	208	343	548	810	1122	1573	2101
clang	seq	7	18	46	97	185	309	489	719	993	1359	1825
	par 2	3	16	46	103	192	314	499	732	1007	1378	1841
	par 3	3	16	46	102	191	314	499	733	1006	1379	1839
	par 4	3	16	46	100	190	317	499	731	1008	1377	1844
	par 8	3	17	47	102	190	314	499	730	1008	1373	1841
icc	seq	2	15	45	90	161	264	409	603	822	1116	1487
	par 2	2	13	42	85	155	253	399	584	803	1091	1458
	par 3	2	13	41	85	157	254	399	583	803	1090	1461
	par 4	2	13	40	86	156	253	399	584	803	1090	1456
	par 8	2	13	41	85	156	256	399	584	801	1091	1456


Компилятор gcc
Время, мс
2500
2000
1500
1000
500
0
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
gcc seq
gcc par 2
gcc par 3
gcc par 4
gcc par 8


Компилятор clang
Время, мс
2000
1500
1000
500
0
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
clang seq
clang par 2
clang par 3
clang par 4
clang par 8

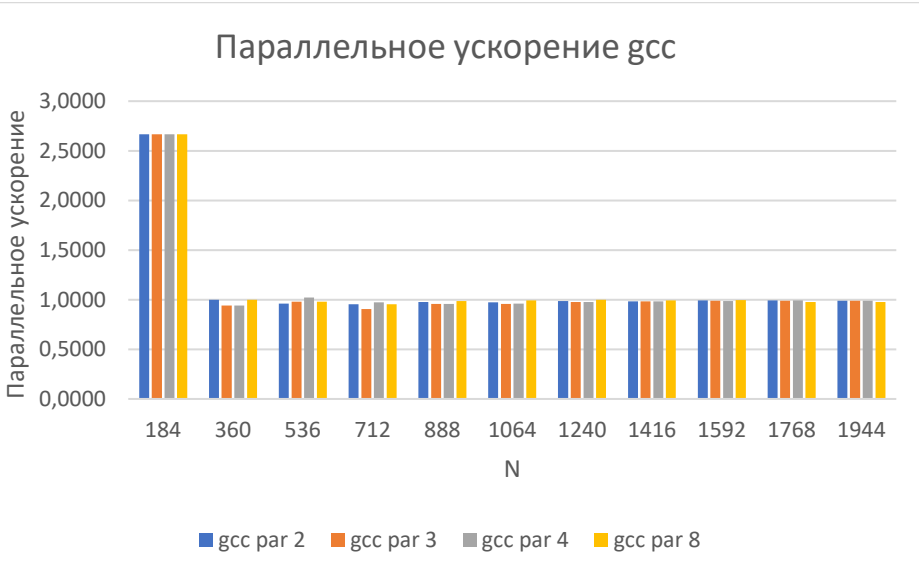
Компилятор icc
Время, мс
1600
1400
1200
1000
800
600
400
200
0
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
icc seq
icc par 2
icc par 3
icc par 4
icc par 8

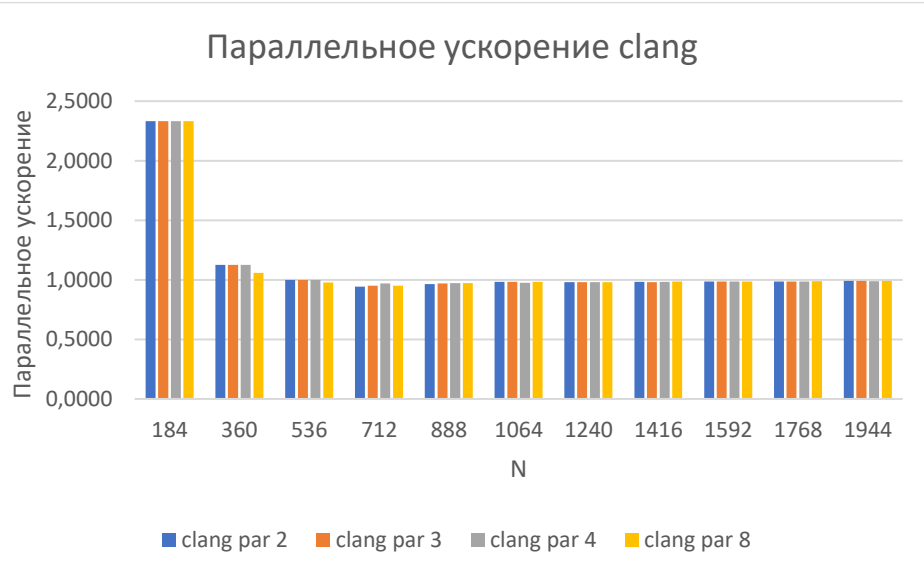
Таблица 2. Значения X при различных N

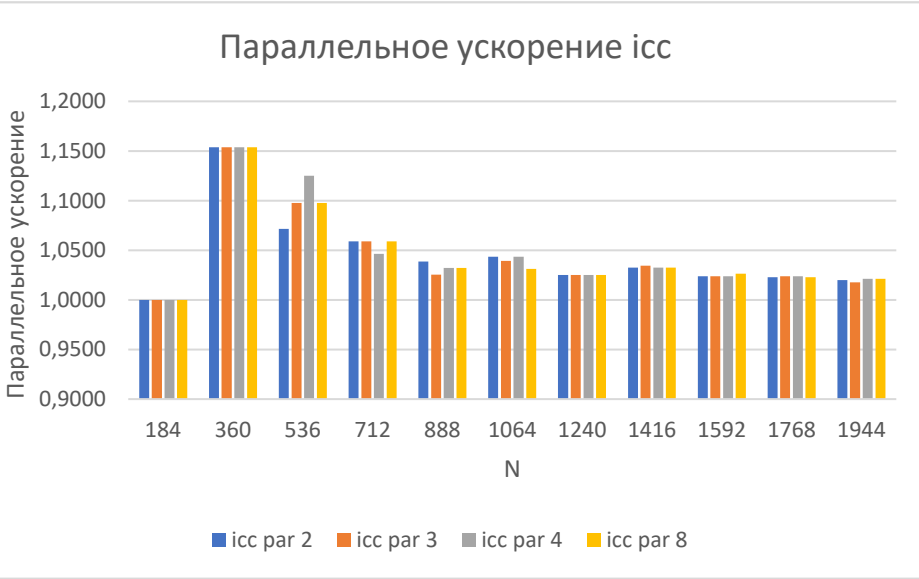
N	184	360	536	712	888	1064	1240	1416	1592	1768	1944
gcc	1925,255249	3839,78076	5617,526855	7486,914551	9371,28809	11277,95801	13134,15332	14894,33008	16738,69336	18621,79297	20393,7832
clang	1942,634644	3860,49927	5645,358398	7507,962891	9391,71289	11302,77441	13158,23828	14914,33203	16757,14258	18644,98633	20416,74805
icc	1925,282349	3839,66846	5617,643555	7487,341797	9371,44043	11277,46289	13133,03418	14892,625	16736,37109	18618,83398	20390,23438

Таблица 3. Значение параллельного ускорения

N		184	360	536	712	888	1064	1240	1416	1592	1768	1944
gcc	par 2	2,6667	1,0000	0,9608	0,9545	0,9762	0,9742	0,9856	0,9841	0,9929	0,9935	0,9908
	par 3	2,6667	0,9412	0,9800	0,9052	0,9579	0,9577	0,9768	0,9817	0,9885	0,9897	0,9879
	par 4	2,6667	0,9412	1,0208	0,9722	0,9579	0,9605	0,9750	0,9817	0,9876	0,9916	0,9884
	par 8	2,6667	1,0000	0,9800	0,9545	0,9856	0,9913	0,9982	0,9938	0,9955	0,9752	0,9752
clang	par 2	2,3333	1,1250	1,0000	0,9417	0,9635	0,9841	0,9800	0,9822	0,9861	0,9862	0,9913
	par 3	2,3333	1,1250	1,0000	0,9510	0,9686	0,9841	0,9800	0,9809	0,9871	0,9855	0,9924
	par 4	2,3333	1,1250	1,0000	0,9700	0,9737	0,9748	0,9800	0,9836	0,9851	0,9869	0,9897
	par 8	2,3333	1,0588	0,9787	0,9510	0,9737	0,9841	0,9800	0,9849	0,9851	0,9898	0,9913
icc	par 2	1,0000	1,1538	1,0714	1,0588	1,0387	1,0435	1,0251	1,0325	1,0237	1,0229	1,0199
	par 3	1,0000	1,1538	1,0976	1,0588	1,0255	1,0394	1,0251	1,0343	1,0237	1,0239	1,0178
	par 4	1,0000	1,1538	1,1250	1,0465	1,0321	1,0435	1,0251	1,0325	1,0237	1,0239	1,0213
	par 8	1,0000	1,1538	1,0976	1,0588	1,0321	1,0313	1,0251	1,0325	1,0262	1,0229	1,0213


Параллельное ускорение gcc
Параллельное ускорение
3,0000
2,5000
2,0000
1,5000
1,0000
0,5000
0,0000
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
gcc par 2
gcc par 3
gcc par 4
gcc par 8


Параллельное ускорение clang
Параллельное ускорение
2,5000
2,0000
1,5000
1,0000
0,5000
0,0000
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
clang par 2
clang par 3
clang par 4
clang par 8


Параллельное ускорение icc
Параллельное ускорение
1,2000
1,1500
1,1000
1,0500
1,0000
0,9500
0,9000
184
360
536
712
888
1064
1240
1416
1592
1768
1944
N
icc par 2
icc par 3
icc par 4
icc par 8

Выводы

В ходе выполнения лабораторной работы мы получили несколько программ для каждого из трех компиляторов – одну последовательную и 4 параллельных, выполняющихся на 2, 3, 4, 8 потоках.
В результате выполнения этих программ были получены результаты, указанные в таблицах, на основе которых можно сделать следующие выводы:

1) Распараллеливание имело смысл только в случае использования компилятора icc, предназначенного для процессора, установленного на экспериментальном компьютере; во всех остальных случаях (кроме минимального значения N) при увеличении потоков время выполнения программы только увеличивалось либо оставалось прежним.
2) Чем больше размерность массива обрабатываемых данных, тем меньше пользы от распараллеливания программ.
3) Результат выполнения программ может незначительно отличаться в зависимости от компилятора за счет различных принципов выполнения некоторый действий (например, округления и приведения типов).
4) Исходя из значения параллельного ускорения, только компилятор icc смог распараллелить исполняемые программы таким образом, чтобы уменьшить время выполнения параллельной программы по сравнению с последовательной. Следовательно, для распараллеливания лучше всего ~~ничего не параллелить~~ использовать те инструменты, которые как можно лучше совместимы с используемым аппаратным обеспечением.