ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ “СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИИ И ИНФОРМАТИКИ”

Кафедра ВС

Лабораторная работа № 2
««ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НА АППАРАТУРНОМ УРОВНЕ»

Выполнил:
ст. гр. МГ-165 Марков В.А.
Проверил:
Фульман В.О.

Новосибирск 2016

Задание на лабораторную работу

1. Разработайте программу, реализующее псевдопараллельное выполнение двух функций: одна из которых непрерывно выводит на экран символ A, а другая непрерывно на экран выводит символ B. Переключение между выполнением функций должно осуществляться раз в три секунды по сигналу от таймера.

Листинг 1 –Псевдопараллельное исполнение

```
#include <stdio.h>
#include <signal.h>
#include <sys/time.h>

#include <unistd.h>

void print_a() { printf ("a"); fflush(stdout); }
void print_b() { printf ("b"); fflush(stdout); }

void (*call_back)();

void signalhandler(int signo)
{
    if (call_back == &print_a) {
        call_back = &print_b;
    } else {
        call_back = &print_a;
    }
}

int main(int argc, char const *argv[])
{
    struct itimerval nval, oval;

    signal(SIGALRM, signalhandler);

    nval.it_interval.tv_sec  = 3; // interval
    nval.it_interval.tv_usec = 0;

    nval.it_value.tv_sec     = 3; // time until next
expiration
    nval.it_value.tv_usec    = 0;

    setitimer(ITIMER_REAL, &nval, &oval);

    call_back = &print_a;

    while(1) {
        (*call_back)();
        sleep(1);
    }

    return (0);
}
```

2. Доработайте программу умножения прямоугольных матриц, разработанную в лабораторной работе 1, так, чтобы имелась возможность сгенерированную комбинацию матриц A и B записать в файл и считать из файла, чтобы провести расчет умножения матриц повторно.

Листинг 3 – Считывая параметров командой строки

```
int main(int argc, char const *argv[])
{
    std::string line;
    std::ifstream myfile("workfile");

    if (!myfile.is_open()) {
        cerr << "can't open file" << endl;
        return -1;
    }
```

```
    while (getline(myfile,line)) {
        std::vector<vector<int>> matrix1;
        std::vector<vector<int>> matrix2;

        int param = stoi(line);

        generatingMatrix(matrix1, param, param);
        generatingMatrix(matrix2, param, param);

        chrono::high_resolution_clock::time_point         t1         =
chrono::high_resolution_clock::now();

        multiplyMatrix(matrix1, matrix2, param, param, param);

        chrono::high_resolution_clock::time_point         t2         =
chrono::high_resolution_clock::now();

        chrono::duration<double>              time_span              =
chrono::duration_cast<chrono::duration<double>>(t2 - t1);

        std::cout << __FILE__<< " It took on "<< param << " - " <<
time_span.count() << " seconds."  << std::endl;

    }
    myfile.close();

    return 0;
}
```

3. Сгенерируйте наборы матриц A и B разных размеров (количество строк и столбов в матрицах в размерах от 16 до 4096, с шагом 16).

Листинг 2 –Генерация размеров матриц

```
#!/usr/bin/python
import commands
import os

count = 16
f = open('workfile', 'w')

while count <= 4096:
   string = str(count)
   f.write(string)
   f.write("\n")

   count += 16

f.close()
```

4. Используя ресурсы кластера Jet проведите исследования эффективности разработанной про- граммы умножения матриц в зависимости от способа обхода матриц при расчете результата (всего 4 комбинации обхода: A по строкам и B по строкам, A по строкам и B по столбцам и т.д.). Необходимо построить для каждого из способов обхода матриц A и B графики зависимости времени умножения матриц от размера матрицы1.

Programm	Matrix size			
	512	1024	256	2048
col_row.cpp	4.64563 sec	5,68509 sec	0.500259 sec	55,2283 sec
col_col.cpp	**5.09936 sec**	**12.297 sec**	**0.560252 sec**	**126,445 sec**
row_col.cpp	4.63441 sec	5.6991 sec	0.501678 sec	62,985 sec
row_row.cpp	*3.9551 sec*	*1.33044 sec*	*0.495362 sec*	*13,577 sec*

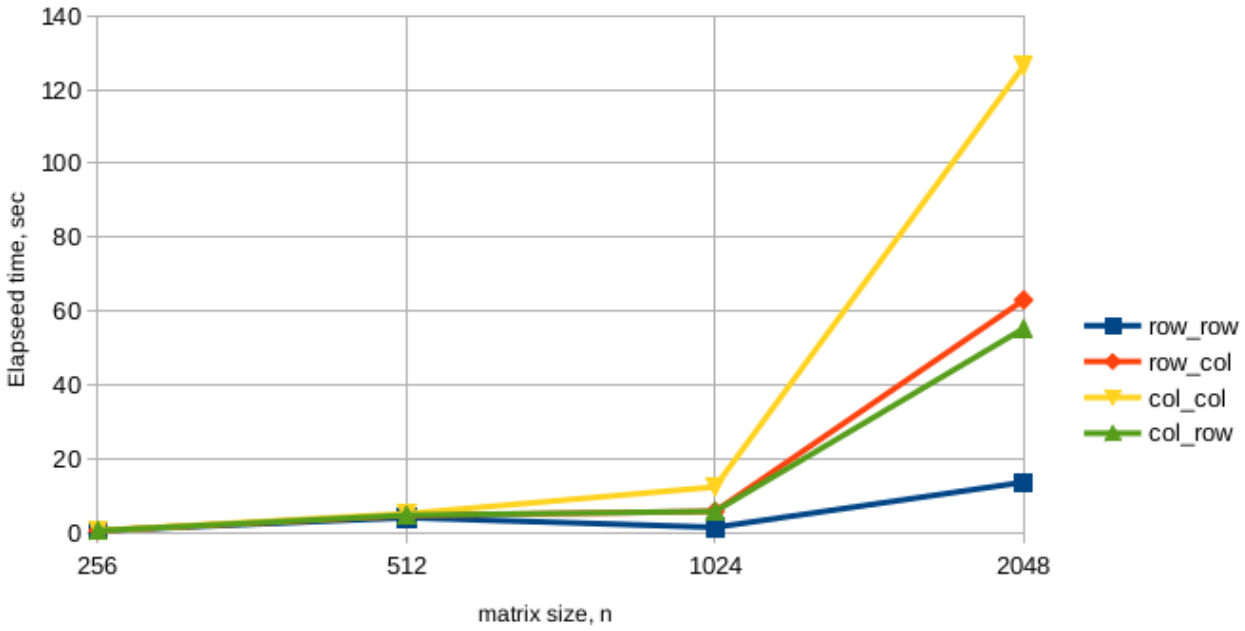

Рисунок 1 – Результат работы разных способов обходов матриц

5. Разработайте программу, выполняющую над двумя векторами вещественных чисел размером 1024 элемента следующие вычисления:

$$Ci = \sqrt{Ai} * Bi$$

Листинг 3 – Векторные операции

```
#include <stdio.h>
#include <math.h>

#define SIZE 1024

void init_vector(double *v)
{
    for (int i = 0; i < SIZE; i++) {
        v[i] = 2.0;
    }
}

void calc_vector(double *a, double *b)
{
    //float c[SIZE] __attribute__((aligned(16)));
    double c[SIZE];

    for (int i = 0; i < SIZE; i++) {
        c[i] = sqrt(a[i] * b[i]);
```

```
    }
}

int main(int argc, char const *argv[])
{
    double A[SIZE];
    double B[SIZE];

    init_vector(A);
    init_vector(B);

    calc_vector(A, B);

    return 0;
}
```

6. Используя компилятор GCC получили текст программы на ассемблере (опция -S) для всех уровней оптимизации кода (опции -O0, -O1, -O2, -O3, -Ofast).
7. Проанализируйте сколько операций выполнится для получения результата в каждом из способов оптимизации кода.

gcc -O option flag

Set the compiler's optimization level.

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more

Рисунок 2 – Сводная таблица компиляторных оптимизаций