5 Лабораторная работа №1. «Автоматическое распараллеливание программ»

5.1 Порядок выполнения работы

1. На компьютере с многоядерным процессором установить Unix-подобную операционную систему и компилятор GCC версии не ниже 9.x. При невозможности установить Unix-подобную операционную систему или отсутствии компьютера с многоядерным процессором можно выполнять лабораторную работу на виртуальной машине. Минимальное число ядер при использовании виртуальной машины – два. Важным условием является отключение гипертрединга, для того, чтобы выполнить честные замеры времени.

2. На языке Си написать консольную программу lab1.c, решающую задачу, указанную в п. 5.5 (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, sys/time.h, math.h. Задача должна решаться 100 раз с разными начальными значениями генератора случайных чисел (ГСЧ). Структура программы примерно следующая:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
int main(int argc, char* argv[]) {
    int i, N;
    struct timeval T1, T2;
    long delta_ms;
    N = atoi(argv[1]); // N равен первому параметру командной строки
    gettimeofday(&T1, NULL); // запомнить текущее время T1
    for (i=0; i<100; i++) { // 100 экспериментов
        srand(i);  // инициализировать начальное значение ГСЧ
        // Заполнить массив исходных данных размером N
        // Решить поставленную задачу, заполнить массив с результатами
        // Отсортировать массив с результатами указанным методом
    }
    gettimeofday(&T2, NULL);   // запомнить текущее время T2
    delta_ms = (T2.tv_sec - T1.tv_sec) * 1000 +
               (T2.tv_usec - T1.tv_usec) / 1000;
    printf("\nN=%d. Milliseconds passed: %ld\n", N, delta_ms);
    return 0;
}
```

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

```
/home/user/gcc -O3 -Wall -Werror -lm -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 -lm -floop-parallelize-all
↪  -ftree-parallelize-loops=K lab1.c -o lab1-par-K
```

   (переменной K поочерёдно присвоить хотя бы четыре значения: один, меньше числа физических ядер, равное числу физических ядер и больше числа физических ядер).

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, Telegram и Skype), чтобы они не влияли на результаты последующих экспериментов. При использовании ноутбука **необходимо иметь постоянное подключение к сети питания** на время проведения эксперимента.

7. Запускать файл lab1-seq из командной строки, увеличивая значения N до значения N1, при котором время выполнения превысит 0.01 с. Подобным образом найти значение N=N2, при котором время выполнения превысит 5 с.

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

   - запускать lab1-seq для значений $N = N1, N1 + \Delta, N1 + 2\Delta, N1 + 3\Delta, \ldots, N2$ и записывать получающиеся значения времени `delta_ms(N)` в функцию `seq(N)`;
   - запускать lab1-par-K для значений $N = N1, N1 + \Delta, N1 + 2\Delta, N1 + 3\Delta, \ldots, N2$ и записывать получающиеся значения времени `delta_ms(N)` в функцию `par-K(N)`;

- значение Δ выбрать так: $\Delta = (N2 - N1)/10$.

9. Провести верификацию значения X. Добавить в конец цикла вывод значения X и изменить число экспериментов на 5. Сравнить значения X для распараллеленной программы и нераспараллеленной.

10. Написать отчёт о проделанной работе.

11. Подготовиться к устным вопросам на защите.

12. Найти вычислительную сложность алгоритма до и после распараллеливания, сравнить полученные результаты.

13. **Необязательное задание №1 (для получения оценки «4» и «5»).** 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
```

14. **Необязательное задание №2 (для получения оценки «5»).** 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-threshold=0 -par-num-threads=K -o lab1-icc-par-K
↪  lab1.c
```

5.2 Состав отчета

1. Титульный лист с названием вуза, ФИО студента и названием работы.

2. Содержание отчета (с указанием номера страниц и т.п.).

3. Описание решаемой задачи (взять из п. 5.5).

4. Краткая характеристика использованного для проведения экспериментов процессора, операционной системы и компилятора GCC (официальное название, номер версии/модели, разрядность, число ядер, ёмкость ОЗУ, размер кэша и т.п.).

5. Полный текст программы lab1.c в виде отдельного файла.

6. Таблицы значений и графики функций seq(N), par-K(N) с указанием времени выполнения и величины параллельного ускорения. Предпочтительно использовать столбчатые гистограммы, показывающие зависимости времени или ускорения от размера массива.

7. Подробные выводы с анализом приведённых графиков и полученных результатов.

8. Отчёт предоставляется в бумажном или электронном виде вместе с полным текстом программы. По требованию преподавателя нужно быть готовыми скомпилировать и запустить этот файл на компьютере в учебной аудитории (или своём ноутбуке).

5.3 Вопросы для самопроверки

1. На что влияет параметр `seed`?

2. Для чего может понадобиться ключ `-lm`?

3. Почему значения X могут отличаться для последовательной и для параллельной программ?

4. Параллельное ускорение составило 0.1. Когда возможна такая ситуация, и о чём это свидетельствует?

5. В каком случае значение параллельной эффективности будет больше значения параллельного ускорения для некоторой программы на некоторой вычислительной машине?

5.4 Подготовка к защите

1. Уметь объяснить каждую строку программы, представленной в отчёте.

2. Знать о назначении и основных особенностях GCC, а также о назначении всех использованных в работе ключей компиляции GCC.

3. Знать материал лекции №1.

4. Взять с собой все нужные файлы для демонстрации работы программы.

5.5 Варианты заданий

Вариант задания выбирается в соответствии с приведёнными ниже описанием этапов, учитывая, что число A = Ф*И*О, где Ф, И, О означают число букв в фамилии, имени и отчестве студента. Номер варианта в соответствующих таблицах выбирается по формуле X = 1 + ((A mod 47) mod B), где B – число 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**. Порядок вычислений должен быть следующим:

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:

Номер варианта	**Операция**
1	Гиперболический синус с последующим возведением в квадрат
2	Гиперболический косинус с последующим увеличением на 1
3	Гиперболический тангенс с последующим уменьшением на 1
4	Гиперболический котангенс корня числа
5	Деление на Пи с последующим возведением в третью степень
6	Кубический корень после деления на число e
7	Экспонента квадратного корня (т.е. M1[i] = exp(sqrt(M1[i])))

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):

Номер варианта	**Операция**
1	Модуль синуса (т.е. M2[i] = |sin(M2[i] + M2[i-1])|)
2	Модуль косинуса
3	Модуль тангенса
4	Модуль котангенса
5	Натуральный логарифм модуля тангенса
6	Десятичный логарифм, возведенный в степень e
7	Кубический корень после умножения на число Пи
8	Квадратный корень после умножения на e

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

Номер варианта	**Операция**
1	Возведение в степень (т.е. M2[i] = M1[i]^M2[i])
2	Деление (т.е. M2[i] = M1[i]/M2[i])
3	Умножение
4	Выбор большего (т.е. M2[i] = max(M1[i],M2[i]))
5	Выбор меньшего
6	Модуль разности

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):

Номер варианта	**Операция**
1	Сортировка выбором (Selection sort)
2	Сортировка расчёской (Comb sort)
3	Пирамидальная сортировка (HeapSort, сортировка кучи)
4	Гномья сортировка (Gnome sort)
5	Сортировка вставками (Insertion sort)
6	Сортировка выбором (Selection 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). Данное число необходимо выводить на каждой итерации на этапе верификации. Значение числа X следует привести в отчёте для различных значений N.