**Лабораторная номер 3, 4. Работа с OpenMP. Распараллеливание циклов.**

**Язык:** любой, поддерживающий openmp ;)

**Цель:** изучение основ работы с библиотекой openmp при распараллеливании циклов for.

Немного теории:  
Рассказать про OpenMP

Используем omp parallel for

На что следует обратить внимание:

1. Поддержку OpenMP надо включать в настройках компилятора. Не достаточно добавить «include <omp.h>”.

- ддля Visual Studio находится во вкладке свойств проекта -> язык -> Openmp,

- для gcc надо добавлять флаг ‘-fopenmp’

- обычно все настройки компилятора делаются отдельно для debug и release конфигураций, так что не забываем убедиться, что флаг добавлен в нужную вам конфигурацию

1. Программа будет компилироваться и работать если флаг не добавлен. Компилятор даже не будет проверять ошибки в директивах openmp.  
   Если вы видите, что время работы параллельной и последовательной версии алгоритма совпадает – проверьте, действительно ли openmp включен для нужной вам конфигурации.

**Задание:**

Реализовать параллельный алгоритм на С/С++ с использованием возможностей библиотеки OpenMP.

Замерить время работы последовательной и параллельной версий алгоритма.

Для каждого запуска сравнивать результат с результатом работы последовательной версии.

Исследовать влияние на время работы параллельной версии алгоритма:

- числа используемых потоков (omp\_set\_num\_threads), от 1 до количества ядер процессора.

- объема обрабатываемых данных (N, M)

**Варианты:**

1. Вариант 1-6 из 1лабораторной.
2. Вариант 7-8 из 1 лабораторной
3. Сортировку массива целых чисел (алгоритм выбирайте самостоятельно), массив из N случайных чисел.
4. Алгоритм Гаусса для решения системы из N линейных уравнений с N неизвестными.
5. Сортировку подсчетом массива целых чисел, массив из N случайных чисел от 0 до M < N/10
6. Алгоритм флойда (поиск путей в графе из каждой вершины в каждую), для из N вершин, 3N ребер со случайными весами.
7. Поиск строки длиной M символов в тексте длиной N символов. Текст и искомую строку загружать из файла. Для сравнения строк использовать расстояние Хэмминга.
8. \*Поиск строки длиной M символов в тексте длиной N символов. Текст и искомую строку загружать из файла. Для сравнения строк использовать расстояние Левенштейна.
9. Brute-Force matcher: Есть 2 матрицы из N(строк)\*M(столбцов) float элементов, необходимо реализовать алгоритм, для каждой строки первой матрицы подбирающий номер строки во второй матрице, так, что бы сумма квадратов разностей элементов выбранных строк была минимальна. Можно использовать каждую строку второй матрицы несколько раз.
10. \* То же что и 9, но каждая строка второй матрицы должна быть отображена ровно на одну строку первой матрицы, минимизировать общую сумму квадратов отклонений по всем элементам всех строк.
11. Для массива из N чисел, случайно распределенных от 0 до 1 (лучше использовать нелинейное распределение чисел по интервалу) найти границы интервала x1 и x2, такого, что бы ему принадлежало не менее 50% чисел массива а ширина интервала была минимальна.
12. \*Свой вариант. Можно алгоритм из своего диплома взять.

**Баллы:**

1. Реализация алгоритма **2б 3б** (берете предыдущую лабораторную и добавляете вариант распараллеливания через omp) **4б**  (пишете новый код, а не берете реализацию из предыдущей работы)
2. Добиться ускорения параллельной версии, сохраняя корректность результата **2б**

Итого: 4 – 6 баллов.

**Лабораторная номер 4. Task-based параллелизм в openmp, вложенный параллелизм.**

**Язык:** любой, поддерживающий openmp ;)

**Цель:** изучение работы с библиотекой openmp на уровне задач

**Секции**

#pragma omp sections

{

#pragma section

foo();

#pragma section

bar();

}

**Задачи**

#pragma omp task

foo();

#pragma omp task

bar();

**Вложенный параллелизм**

Если дирактива omp parallel находится внутри другой omp parallel, то образуется вложенный параллелизм. Такое возможно если директивы явно вложены или если в функции используется рекурсивный вызов самой себя.

По умолчанию, openmp для вложенной конструкции parallel не будет создавать потоков.

Что бы потоки были созданы необходимо добавить в начало программы вызов omp\_set\_nested(1)

**Ограничение параллелизма для рекурсивных функций.**

**TODO**

**Задача:**

Реализовать параллельное вычисление рекурсивного алгоритма с использованием библиотеки OpenMP.

Рекомендуется использовать Section или Task для реализации параллелизма, желательно сбалансировать вычисления между потоками, но не делать слишком мелких задач.

Замерить время работы последовательной и параллельной версий алгоритма, добиться ускорения параллельной версии.

Данные генерировать псевдо-рандомно (с использованием зерна) или читать из файла.

**Варианты заданий:**

1. Распараллелить Quicksort или merge sort (или любую другую рекурсивную сортировку).
2. Фибоначи. Реализовать рекурсивное параллельное вычисление чисел Фибоначи с кешированием (никогда не делайте так IRL, но на лабораторной можно)
3. Параллельная реализация поиска кратчайшего пути в графе.
4. Построение Kd-дерева. Реализовать построение дерева (добавление элемента) и поиск ближайшей точки для двух (k=2) или большего числа измерений. Распараллелить построение дерева по заданному набору точек (параллельно добавлять их в дерево).
5. Распараллелить построение BSP-дерева (для двухмерного набора отрезков).
6. Реализовать разбиение невыпуклого замкнутого контура без самопересечений на выпуклые части (Convex Decomposition)
   1. Реализовать Approximate Convex Decomposition
7. Выбрать свой алгоритм

**Баллы:**

1. Реализация алгоритма 2б 3б.
2. Добиться ускорения параллельной версии, сохраняя корректность результата 3б 4б

Итого: 5 – 7 баллов.