**Лабораторная работа 2.5**

**«Параллельная реализация алгоритма Quick sort с помощью OpenMP»**

Параллельные задачи

Директива OpenMP task позволяет выделять отдельные блоки кода, находящиеся внутри параллельной секции, в независимые задачи, которые будут помещены в пул-задач, после чего исполнены рабочими потоками. В листинге 1 приведен пример создания задач. Обратите внимание, данная директива должна быть помещена внутри параллельной секции. Вопрос - *Сколько задач будет создано в примере ниже?*

#pragma omp parallel num\_threads(4)

{

#pragma omp task

{

std::cout << "hello world";

} // #pragma omp task

} // #pragma omp parallel

OpenMP также предоставляет директиву ожидания потоком завершения порожденных им задач: taskwait.

Пример:

#pragma omp parallel num\_threads(4)

{

#pragma omp task

{

std::cout << "hello world";

} // #pragma omp task

#pragma omp taskwait

std::cout << "bye" << std::endl;

} // #pragma omp parallel

Параллельная быстрая сортировка

Алгоритм быстрой сортировки (Quick sort) является одним из самых быстрых алгоритмов сортировки массивов и реализован во множестве стандартных библиотек различных языков. Его отличительной особенностью является то, что он в своей основе опирается на принцип «разделяй и властвуй», заключающегося в рекурсивном разбиении решаемой задачи на две или более подзадачи того же типа, но меньшего размера, и комбинировании их решений для получения ответа к исходной задаче [1].

Алгоритм быстрой сортировки состоит из четырех шагов:

1. Выбор опорного элемента.
2. Переупорядочивание массива таким образом, чтобы все элементы меньше опорного размещались перед ним, а больше и равные после.
3. Рекурсивное применение шагов 1 и 2 к подмассиву слева от опорного элемента.
4. Рекурсивное применение шагов 1 и 2 к подмассиву справа от опорного элемента.

Заметим, что третий и четвертый шаги алгоритма совершенно не зависят друг от друга, и, следовательно, могут быть выполнены в отдельных задачах.

Обратите внимание, что до некоторого размера входного массива накладные расходы на создание задач могут оказаться выше, чем выигрыш от их параллельной сортировки. Учтите это с помощью использования параметра условного создания задачи *if(expr)*.

Задание:

1. Экспериментальным путем определить минимальный размер массива, для которого целесообразно выполнять рекурсивные вызовы функций в отдельный задачах.
2. Постройте график зависимости времени сортировки массива от числа используемых потоков, оценить эффективность распараллеливания.