Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 2

по дисциплине «**Параллельные вычислительные технологии**»

Выполнил:
студент гр. ИС-242
«27» мая 2024 г. _____ /Журбенко В. Е./

Проверил:
Ассистент кафедры ВС
«27» мая 2024 г. _____ //

Оценка «_____»

Новосибирск 2024

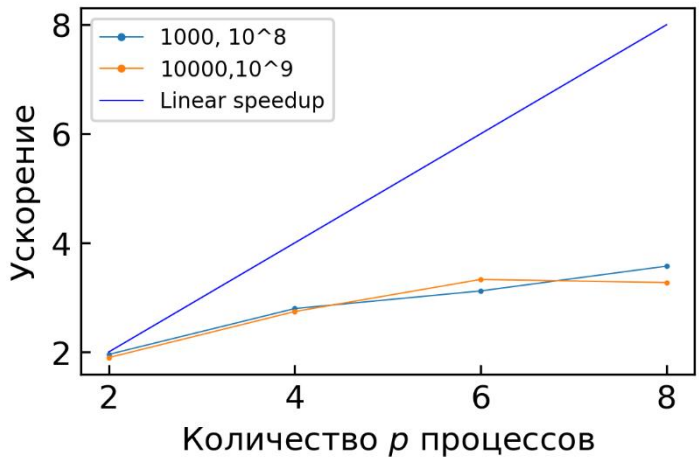
Оглавление

ЗАДАНИЕ

На базе директив **#pragma omp task** реализовать многопоточный рекурсивный алгоритм быстрой сортировки (QuickSort). Опорным выбирать центральный элемент подмассива (функция partition, см. слайды к лекции). При достижении подмассивами размеров THREASHOLD = 1000 элементов переключаться на последовательную версию алгоритма.

Выполнить анализ масштабируемости алгоритма для различного числа сортируемых элементов и порогового значения THRESHOL

ГРАФИК УСКОРЕНИЯ




```
└─$ ./parallel
79.401629 - время последовательной программы
время работы паралел прог - 41.847646, потоков - 2  speedup:1.897398
время работы паралел прог - 28.964735, потоков - 4  speedup:2.741321
время работы паралел прог - 23.841805, потоков - 6  speedup:3.330353
время работы паралел прог - 24.253557, потоков - 8  speedup:3.273814
└─$ vim parallel.c
└─$ gcc -Wall ./parallel.c -fopenmp -lm -o ./parallel
└─$ ./parallel
7.283109 - время последовательной программы
время работы паралел прог - 3.724186, потоков - 2  speedup:1.955624
время работы паралел прог - 2.604757, потоков - 4  speedup:2.796080
время работы паралел прог - 2.334472, потоков - 6  speedup:3.119810
время работы паралел прог - 2.037538, потоков - 8  speedup:3.574465
```

Описание функций

```
void partition(int *v, int *i, int *j, int low, int high)
{
    *i = low;
    *j = high;
    int pivot = v[(low + high) / 2];
    do
    {
        while (v[*i] < pivot)
            (*i)++;
        while (v[*j] > pivot)
            (*j)--;
        if (*i <= *j)
        {
            swap(&(v[*i]), &(v[*j]));
            (*i)++;
            (*j)--;
        }
    } while (*i <= *j);
}
```

Функция partion делит массив на две части, относительно опорного элемента и сортирует 2 подмассива.

```
void quicksort(int *v, int low, int high)
{
    int i, j;
    // print_arr(v);
    partition(v, &i, &j, low, high);
    if (low < j)
        quicksort(v, low, j);
    if (i < high)
        quicksort(v, i, high);
}
```

Рекурсивная сортировка подмассивов.

Параллельная программа

```
void quicksort_tasks(int *v, int low, int high)
{
    int i, j;
    partition(v, &i, &j, low, high);
    if (high - low < THRESHOLD || (j - low < THRESHOLD || high - i < THRESHOLD))
    {
        if (low < j)
            quicksort_tasks(v, low, j);
        if (i < high)
            quicksort_tasks(v, i, high);
    }
    else
    {
#pragma omp task untied
        {
            quicksort_tasks(v, low, j);
        }
        quicksort_tasks(v, i, high);
    }
}
```

Проверяется условие, что ни подмассивы, ни весь массив не превышают граничное значение (THRESHOLD), иначе программа выполняется последовательно (не разумно тратить ресурсы процессора на параллелизм при сортировке массива <= 1000 элементов). Если пороговое значение не достигнуто, то с помощью pragma omp task untied создаём неявную задачу сортировки левого подмассива, которая pushится в дек наименее загруженного потока (потому что untied отвязывает задачу от главного потока), правый подмассив сортируется потоком, который открыл задачу и т.к он не загружен в этот момент, он выполняет сортировку правого подмассива.

Алгоритм быстрой сортировки наиболее эффективно распараллеливается за счёт параллелизма задач, потому что мы заранее не знаем, сколько будет рекурсивных вызовов, поэтому балансировать загрузку потоков получается, только используя неявные задачи (task).