

Лабораторная работа 5. Параллелизм задач

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 `THRESHOLD = 1000` элементов переключаться на последовательную версию алгоритма.

Выполнить анализ масштабируемости алгоритма для различного числа сортируемых элементов и порогового значения `THRESHOLD`.

Описание функций

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

#define CLOCK_MONOTONIC 1
int size = 1000000;
int THRESHOLD = 1000;
int threshold = 1000;

double speedup(double time_nomp, double time_omp){
    return time_nomp / time_omp;
}

double wtime() {
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return ts.tv_sec + ts.tv_nsec * 1E-9;
}

void write(double S, int n) {
    FILE *f;
    f = fopen("res.txt", "a");
    fprintf(f, "%d %f\n", n, S);
    fclose(f);
}
```

```
void swap(int* a, int* b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

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

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

```
void quicksort_tasks(int *v, int low, int high) {  
    int i, j;
```

```

partition(v, low, high, &i, &j);
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);
}
}
void print_arr(int *arr)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
void fillArrayWithRandomValues(int arr[]) {
    srand(time(0));
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 1000 + 1;
    }
}

int main() {
    int arr[size];
    fillArrayWithRandomValues(arr);

    double t = wtime();
    quicksort_serial(arr, 0, size - 1);
    t = wtime() - t;
    printf("Serial time: %f \n", t);
    //print_arr(arr);

    for (int i = 2; i <= 8; i += 2) {
        double time_omp;
        printf("-----%d-----\n", i);
        fillArrayWithRandomValues(arr);
        time_omp = wtime();
    }
}

```

```

#pragma omp parallel num_threads(i)
{
    #pragma omp single
    quicksort_tasks(arr, 0, size - 1);
}
//print_arr(arr);
time_omp = wtime() - time_omp;
printf("Tasks time: %f \n", time_omp);
printf("Speedup: %.6f\n\n", speedup(t, time_omp));
write(speedup(t, time_omp), i);
}
return 0;
}

```

main

- Заполняет массив случайными значениями.
- Измеряет время выполнения последовательной сортировки и выводит его.
- Для каждого количества потоков от 2 до 8 (с шагом 2) выполняет параллельную сортировку:
 - Заполняет массив случайными значениями.
 - Измеряет время выполнения параллельной сортировки и выводит его.
 - Вычисляет и выводит ускорение.
 - Записывает результат в файл.

fillArrayWithRandomValues

- Заполняет массив случайными значениями от 1 до 1000.

print_arr

- Печатает массив.

quicksort_tasks

- Рекурсивно выполняет быструю сортировку, используя прагмы OpenMP для параллельной реализации.

quicksort_serial

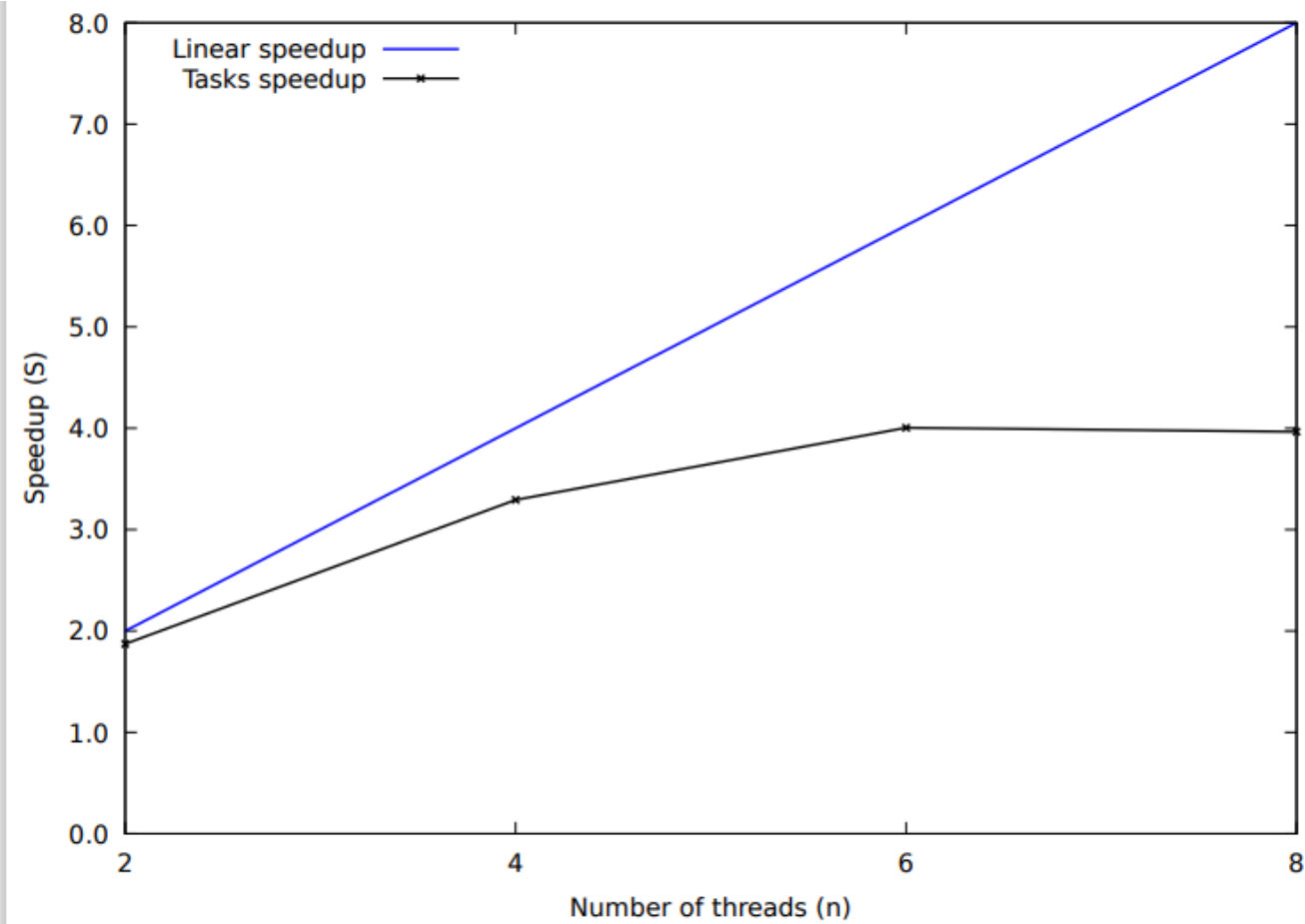
- Рекурсивно выполняет быструю сортировку. Последовательная версия.

partition

- Разделяет массив на две части относительно опорного элемента.

```
39eb52a..552b3a7 main -> main
● grogu@я-ПК:~/2 семестр/PVT/pct-spring-lab5$ make run
gcc main.c -o prog -fopenmp
./prog
Serial time: 0.084897
-----2-----
Tasks time: 0.045736
Speedup: 1.856240
...
-----4-----
Tasks time: 0.027133
Speedup: 3.128913
...
-----6-----
Tasks time: 0.023683
Speedup: 3.584639
-----8-----
Tasks time: 0.020739
Speedup: 4.093582
○ grogu@я-ПК:~/2 семестр/PVT/pct-spring-lab5$
```

график



```
int size = 1000000;
```

Размер массива один миллион чисел.

Процессор ryzen 5 3600 6c/12t