Project 3 Report

Chai Yi 9008308232
chyi13@student.bth.se

Content List

1.Parallel Quicksort Implementation

1.1 Work partition

My approach is generally the same as practical quicksort in the lecture. It is described as follows:

My approach is restricted to 8 cpus environment since I have not found a better solution. It will divide the target array into 4 parts which will be sorted individually by two steps. First step is to divide the array into two parts by local re-arrangement and global re-arrangement.

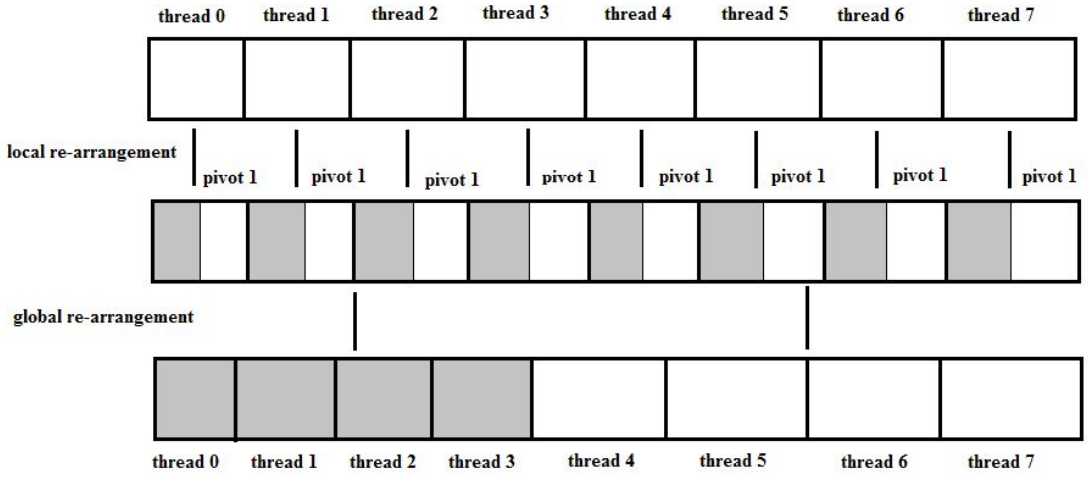

Figure 1. First step

As in figure 1, target array is first divided into 8 equal parts, that every part corresponds to a thread. Then a pivot value is selected and local re-arrangement is performed that the elements with values less than pivot 1 come before the elements with values greater than pivot 1. At last global re-arrangement is performed to put all elements with values less than pivot in the front of the array while the other elements come after them.

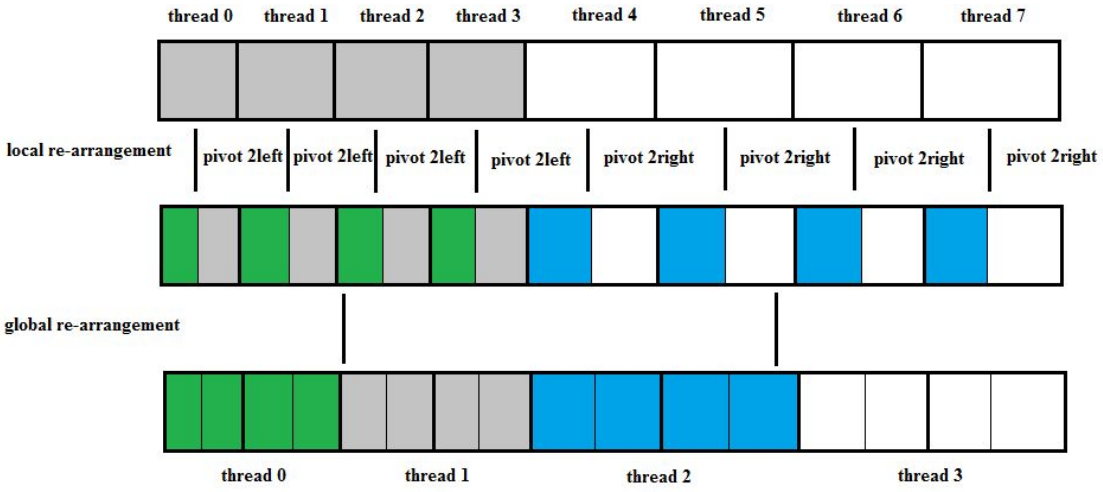

Figure 2. Second step

The second step is shown in figure 2. Two pivot values are selected to divide the array into four parts. The green part contains the elements with values less than pivot 2 left. The gray part contains the elements with values less than pivot 1 and greater than pivot2 left. The blue part is the elements with values less than pivot 2 right and greater than pivot 1. The white part contains the rest elements.

With the previous 2 division steps, four orderer parts are created. Then each of the four parts is assigned to a single thread. Serial quicksort is performed on each thread to achieve the final result.

1.2 OpenMp Implementation

1.2.1 Procedure

//First step:

divide the array into 8 equal parts and assign them to 8 threads.

```
#pragma omp parallel{
    id = omp_get_thread_num();
    local_rearrangement(id);
}
calculate_prefix_sum();
global_rearrangement();
```

// Second step

divide the left part into 4 equal parts and assign them to first 4 threads.

divide the right part into 4 equal parts and assign them to last 4 threads.

```
#pragma omp parallel{
    id = omp_get_thread_num();
    local_rearrangement(id);
}
calculate_prefix_sum();
global_rearrangement();
```

// Serial quick sort

assign the four ordered part to 4 threads.

```
#pragma omp parallel{
    id = omp_get_thread_num();
    sequential_quick_sort(id);
}
```

1.2.2 Pivot selection

According to the random array initialization, I take MAX_RAND/2 as first pivot value.

1.3 Experiment

Environment: 8 cpus. Array size: (64*(1024*1024))

1.3.1 Compile OpenMP version

```
$ gcc -fopenmp -o qsort_omp qsort_omp.c
qsort_omp.c: In function 'main':
qsort_omp.c:64:2: warning: incompatible implicit declaration of built-in functio
n 'memset' [enabled by default]
qsort_omp.c:89:2: warning: incompatible implicit declaration of built-in functio
n 'memcpy' [enabled by default]
qsort_omp.c: In function 'init_array':
qsort_omp.c:218:17: warning: incompatible implicit declaration of built-in funct
ion 'malloc' [enabled by default]
```

1.3.2 Sequential version of Quicksort result: 31.71s

```
$ /usr/bin/time ./qsort_seq -p 0
31.40user 0.30system 0:31.71elapsed 100%CPU (0avgtext+0avgdata 1050320maxresiden
t)k
0inputs+0outputs (0major+65684minor)pagefaults 0swaps
```

1.3.3 OpenMP version of Quicksort result: 9.66s

```
$ /usr/bin/time ./qsort_omp -p 0
p: 1073741824
33.92user 0.46system 0:09.66elapsed 355%CPU (0avgtext+0avgdata 2100224maxresiden
t)k
0inputs+0outputs (0major+131316minor)pagefaults 0swaps
```

1.3.4 Speedup 8 cpus

Speedup = 31.71/9.66 = 3.2826

2.OpenMP Gaussian Elimination

2.1 Work partition

My work partition is the same as project 2 which is based on interlaced rows. Given 8 threads, the 1st row, 9th row and (1+8n)th row will be assigned to the first thread. If the row index is i, the thread which index is (i%8) will process this row.

2.2 OpenMP Implementation

```
#pragma omp parallel{
    for row_id←0 to N {
        if (row_id = omp_get_thread_num()){
            Division of this row;
        }
        #pragma omp barrier

        for subrow_id←row_id+1 to N{
            if (subrow_id = omp_get_thread_num()){
                Elimination of this row
            }
        }
        #pragma omp barrier
    }
}
```

2.3 Experiment

Matrix size : 4096*4096 Initialization: Rand

2.3.1 Compile

```
$ gcc -fopenmp -o gaussian_omp gaussian_omp.c
gaussian_omp.c: In function ‘Read_Options’:
gaussian_omp.c:181:5: warning: incompatible implicit declaration of built-in fun
ction ‘exit’ [enabled by default]
```

2.3.2 Sequential version: 3min 12.73s

```
$ /usr/bin/time ./gaussian_seq -n 4096 -P 0

size      = 4096x4096
maxnum    = 15
Init      = rand
Initializing matrix...done

192.60user 0.14system 3:12.73elapsed 100%CPU (0avgtext+0avgdata 526480maxresiden
t)k
0inputs+0outputs (0major+32944minor)pagefaults 0swaps
```

2.3.3 OpenMP version: 44.61s

```
$ /usr/bin/time ./gaussian_omp -n 4096 -P 0

thread_num = 8
size      = 4096x4096
maxnum    = 15
Init      = rand
Initializing matrix...done

351.83user 0.18system 0:44.61elapsed 788%CPU (0avgtext+0avgdata 527536maxresiden
t)k
0inputs+0outputs (0major+33022minor)pagefaults 0swaps
```

2.3.4 Speedup 8 cpus

Speedup = 192.73 / 44.61 = 4.3203