SPP_Lab1

Aufgabe 1

e)&f)

Serial daxpy: 0.714048s.

threads_num	1	2	4	8	16
par-time	0.708938s	0.358548s	0.17953s	0.137688s	0.114912s
with scheduling	0.716574s	0.409928s	0.179564s	0.136177s	0.115908s

Computing with scheduling(static,3) costs more time. 3 is chunk-size, OpenMP divides the iterations into chunks of size chunk-size and it distributes the chunks to threads in a circular order.It creates chunks with size 3, and the problem size is 500000000, so the load is imbalance.Load imbalence causes synchronization delay at the end of the loop, faster threads have to wait for slower threads. Then the execution time will increase.

Aufgabe 2

Record:
Serial full-matrix: 0.197814s
Serial sym-matrix: 0.714048s

threads_num	1	2	4	8	16	32
full-matrix-par c)	0.200811s	0.102252s	0.056245s	0.026848s	0.013448s	0.013964s
full-matrix-par d)	0.202780s	0.103714s	0.053278s	0.026861s	0.012941s	0.018807s
sym-matrix-par h)	0.759033s	0.559502s	0.325615s	0.176292s	0.095447s	0.049101s

d)
no big change of time.

h)
Begruenden Sie welche der beiden Varianten die bessere ist.
Full symmetric matrix multiplication is better. Because it conforms to the concept of caching, it consumes less time in memory access and can make more effective use of parallel resources.
Was sind Vor- und Nachteile der beiden Varianten.
1.Parallel full matrix multiplication:
Advantage: Because memory access is continuous in the address space, the memory access speed is

faster and the program execution time is shorter.
Disadvantage: Requires twice as much memory as the second variant.

2.Parallel symmetric matrix multiplication:
Advantage: Saved memory
Disadvantage:Because memory access is continuous in the address space, the memory access speed is faster and the program execution time is shorter.And there are branches, which also affect the speed of execution

Aufgabe 3

threads_num	1	2	4	8
a)	2.09715s	1.08085s	0.607341s	0.469696s
b)	1.90159s	0.987616s	0.549554s	0.333201s

a)
Output:

```
Generating took 0.478697 seconds.
Serial QuickSort took 1.86367 seconds.
(1): It took me 2.09715 seconds.
(2): It took me 1.08085 seconds.
(4): It took me 0.607341 seconds.
(8): It took me 0.469696 seconds.
```

Wie entwickelt sich die Laufzeit?

1. Parallel program with 1 thread cost more time than serial program.
2. With the increasing of the number of threads, the time cost decreases.
3. Time cost of 1 thread is about twice(1.88) that of 2 threads
4. Time cost of 2 thread is about twice(1.74) that of 4 threads
5. But the decreasing from 4 to 8 threads is much lower, time cost of 4 threads is 1.37 times that of 8 threads

b)
Output:

```
Generating took 0.477081 seconds.
Serial QuickSort took 1.88812 seconds.
(1): It took me 1.90159 seconds.
(2): It took me 0.987616 seconds.
(4): It took me 0.549554 seconds.
(8): It took me 0.333201 seconds.
```

Wie erklaren Sie das Laufzeitverhalten?
1.Parallel program with 1 thread cost more time than serial program.
Reason: Parallel program with 1 thread have overhead, e.g. Task management.

2.With the increasing of the number of threads, the time cost decreases.
Reason: Because there are more threads for independent tasks, so that it can be faster.
3.&4.&5. Because with the increasing of the number of threads. the overhead for parallel program is increasing, e.g threads creation, distributing the work, Synchronizing at the end of the loop.

Add code to speed up

```
void quickSort_par(int64_t* arr, int64_t low, int64_t high) {
  // TODO: modify/implement a), and later b)
  if (low < high) {
    // pi is partitioning index, arr[p] is now
    // at right place
    int64_t pi = partition(arr, low, high);

    // Separately sort elements before
    // partition and after partition
    if (((high - low) < 10000)){
        quickSort_par(arr, low, pi - 1);
        quickSort_par(arr, pi + 1, high);
    }
    else {
#pragma omp task firstprivate(arr, low, pi)
      quickSort_par(arr, low, pi - 1);
#pragma omp task firstprivate(arr, high, pi)
      quickSort_par(arr, pi + 1, high);
    }
  }
}
```

Aufgabe 4

Aufgabe 4

a)

1. i: shared
2. j: shared
3. g1: private
4. g2: shared

b)

1. p: private
2. g1: private