Homework 4: Divide and Conquer
Han Tran

Task 1

The code is attached as **hw4_task1.zip**.
Following are the results and comments.

Strong scalability:
Problem size N=4*1,000,000 long integer

# threads	1	2	4	8	16	32	64
time	16.76132	16.70962	16.87644	17.6	16.78754	16.69592	17.0316

Weak scalability:
Grain size N/p=1,000,000

N	1,000,000	2,000,000	3,000,000	4,000,000
# threads	1	2	3	4
time	1.133321	4.286088	9.54914	16.7073

Comments:

- The code does not have a good strong scalability (i.e. when number of threads increases, the time does not change much. I actually do not quite understand. What I can explain is that I did not do the parallelization in the for loop of partition subroutine. The reason is that the algorithm used for partition is not parallelable. The only parallelization is done in quicksort when the 2 subsets of the array (after the partition) is distributed to threads using task.
- The code does not have a good weak scalability either (e.i. when number of threads increases, the time also increases). Indeed, I do not understand why.
- This task could be improved if I change the algorithm in partition using paralleled scan algorithm.

Task 2

The code does not work correctly because the second loop depends on the first loop (i.e. c[i] has to be finished in order to have correct d[i] = sqrt(c[i]). However, due to **nowait**, the second loop may go before the first loop finishes. Similarly, the third loop depends on the second loop. To fix, we could put a barrier in between loops (and delete the **nowait**) to make sure those are finished before going to the next one:

```
int nowait_ex(int n, int m, float* a, float* b, float* c, float* d)
  int i;
  #pragma omp parallel {
    #pragma omp for schedule(static)
    for (i=0; i<n; ++i)
      c[i] = (a[i] + b[i])*0.5;

    #pragma omp barrier

    #pragma omp for schedule(static)
    for (i=0; i<n; ++i)
      d[i] = sqrt(c[i]);

    #pragma omp barrier

    #pragma omp for schedule(static)
    for (i=0; i<n; ++i)
      e[i] = d[i-1] + a[i];
  }
}
```

Task 3

The code for this task is attached as **hw4_task3.zip**
Comments:

- The code use Scan algorithm (Upsweep & Downsweep) according to the Lecture note "Prefix Sum" presented in class.
- The code can work with an array of generic type (e.g., long, double, float,…) . However, the algorithm used in the code in ONLY works with arrays of size $N=2^m$, where m is positive integer.
- I have not done the other requirements of this task (generic operator, test the code with three-dimensional vector,…). If extra time for this homework is allowed, I could complete these requirements.