IN3200/IN4200 Exercise Set 8

Exercise 1: Parallel processing of dependent tasks

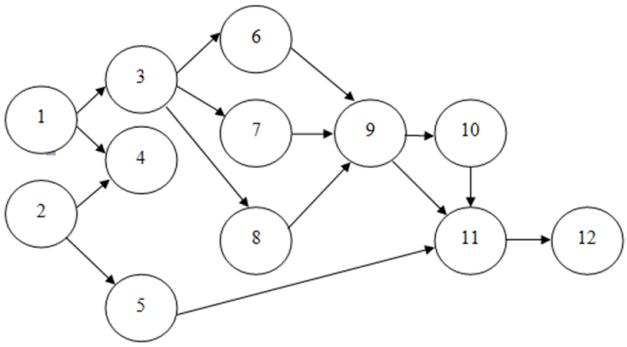

Figure 1: Task-dependency graph of 12 tasks.

Figure 1 is a task-dependency graph showing how 12 tasks depend on each other. Each directed edge in the graph connects a pair of "source" and "destination" tasks. A detination task cannot be started until all its source tasks are carried out. All the 12 tasks are equally time-consuming, requiring one hour of a worker. (The digits in the circles show the numbering of the tasks. We also assume that it is not possible to let two or more workers collaborate on one task for a faster execution.)

a. The case of two workers

If there are two workers, how many hours minimum do they need to carry out all the 12 tasks?

b. The case of three workers

If there are three workers, how many hours minimum do they need to carry out all the 12 tasks?

Exercise 2: Result of running an OpenMP code

If the following code snippet is executed by 4 OpenMP threads, what will be written as output?

```
int total_sum = 0;
int i;
#pragma omp parallel default(shared) reduction(+:total_sum)
{
int my_id = omp_get_thread_num();
int my_sum = 0;
#pragma omp for schedule(static,10)
for (i=1; i<=100; i++)
my_sum += i;
printf("From thread No.%d: my_sum=%d\n", my_id, my_sum);
total_sum += my_sum;
}
printf("Total sum=%d\n",total_sum);
```

Exercise 3: Sorting by odd-even transposition

Given a list of numbers: $a_1, a_2, \ldots, a_n$, the following **odd-even transposition** algorithm can be used to sort the list into an increasing order:

1. **procedure** ODD-EVEN(n)
2. **begin**
3. **for** $i := 1$ **to** n **do**
4. **begin**
5. **if** i is odd **then**
6. **for** $j := 0$ **to** $n/2 - 1$ **do**
7. *compare-exchange*(a_{2j+1}, a_{2j+2});
8. **if** i is even **then**
9. **for** $j := 1$ **to** $n/2 - 1$ **do**
10. *compare-exchange*(a_{2j}, a_{2j+1});
11. **end for**
12. **end** ODD-EVEN

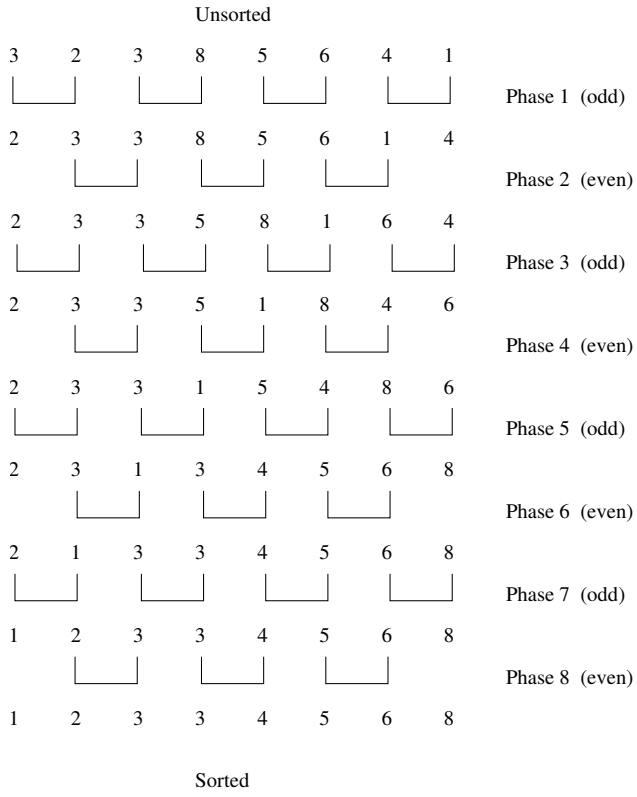

Figure 2: An example using odd-even transposition to sort a list of 8 values.

Note that the action of, for example, *compare-exchange*(a_{2j+1}, a_{2j+2}) is to swap the positions of a_{2j+1} and a_{2j+2} in the case of $a_{2j+1} > a_{2j+2}$. An example of sorting a list of 8 values using odd-even transposition is shown in Figure 2.

a. Serial implementation

Please program a serial function `void serial_odd_even(int n, float *a)` that implements the odd-even transposition algorithm. Write a simple program to test the correctness of this function.

b. Handle a list of "odd" length

Note that the original odd-even transposition algorithm has assumed the length of the input list `a` to be an even integer. Please extend the `serial_odd_even` function to be able to also handle a list of odd length.

c. Earlier termination

The original algorithm requires n phases ($i = 1, \ldots, n$). Improve the `serial_odd_even` function by allowing earlier termination (when the list is already sorted).

d. OpenMP parallelization

Parallelize your program by OpenMP.