Nicolas Demongeot
Marius Lumbroso

Tutorial 2

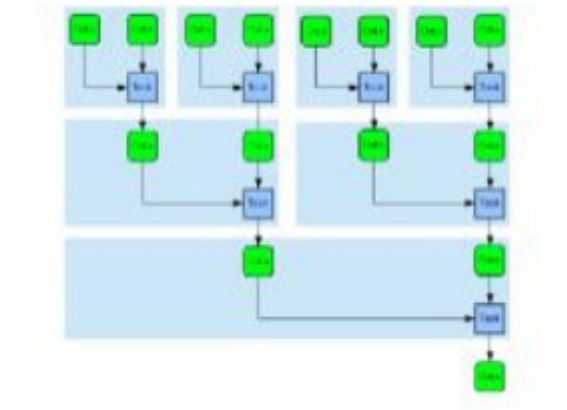
1 – Data aggregation

Q1)

If we modify the array T using more than one thread at a time, it leads to a race condition. We needs T to be access by one thread at a time. We use #pragma omp write to protect the t access from the race condition.

Q2)

We use the 'omp parallel for schedule' pragma to parallelise the for loop assigning a specific thread number per iterations bloc. We compute the reduction according to the following diagram :



A Parallel Version of the Reduce Pattern

Q3)

Instead of handling the race condition manually, open mp has built in keyword 'reduction' that permit to compute some arithmetic operations on an array. We also use the 'shared' keyword to share the array among all threads instead of providing a copy of the variable to each threads.

2 – Open mp map reduce

1)

See `func1()`, the occurrences of each character are stored in an integer array where the index represents the index of the character in the alphabet.

2,3)

Now that we have done the mapping of each character of a string. The mapping can be done in parallel on each character. We use the standard parallelization of open mp #pragma omp parallel. The reduction operator is then used to calculate the total number of characters after the mapping is complete.

4 – The dining philosophers

First, we had to think about storing booleans in each philosophers, indicating whether they had the left or right fork in hand. Little by little, we realized that the implementation could be much simpler and that the expression of these conditions could be done thanks to the open mp 'omp_lock_t' semaphores. Indeed, in the code:

```
omp_unset_lock (& forks [i1]);
omp_unset_lock (& forks [i2]);
```

the philosopher will not be able to obtain the second fork if he does not have the first in hand.
We have also implemented a 'think / eat' system. The philosopher eats for some total time. During his meal, he will eat a few bites, think, and start over until he is eaten for a certain period of time.