Odd-even transposition network

Student: OMRACHI Soukayna
Professors: Bruno Raffin
Arnaud Legrand
Frederic Wagner

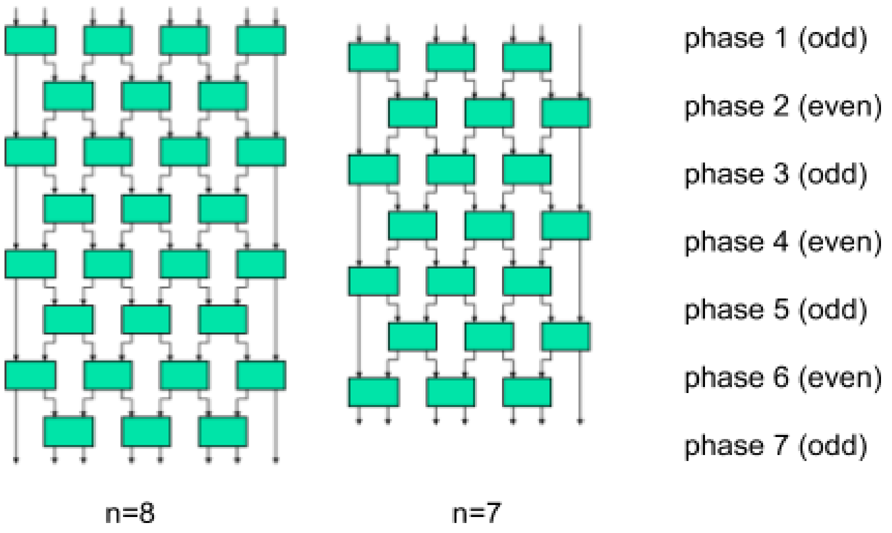
Introduction:

Sorting is one of the most important operations in networks and systems and its efficiency can influences the performance of the system or the network. To speed up this performance, parallelism is applied to the execution of those operations. This work has been carried out on performance evaluations of parallel sorting algorithms that the ***Fast Fourier Transformation*** was based on, the algorithm was running in a cluster of workstations. I based on the Grid5000 when I used a CPU Intel Xeon Gold 5220 with 18 cores, with RAM of 96 Gbps and a network of 2*25*Gps. I used the multi processes and avoid the hyperthreading (2 threads for each node) in order to get a good rate in term of performances.

Odd-even transposition network:

Odd-even transposition algorithm is a parallel sorting algorithm. It is based on comparing every two consecutive numbers in the array and scatter and gather them. It compares the two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order list. The opposite case applies for a descending order series.
Odd-Even transposition sort operates in two phases. In both the phases, processes exchange numbers with their adjacent number in the right.:

- Odd phase: Every odd indexed element is compared with the next even indexed element.
- Even phase: Every even indexed element is compared with the next odd indexed element.



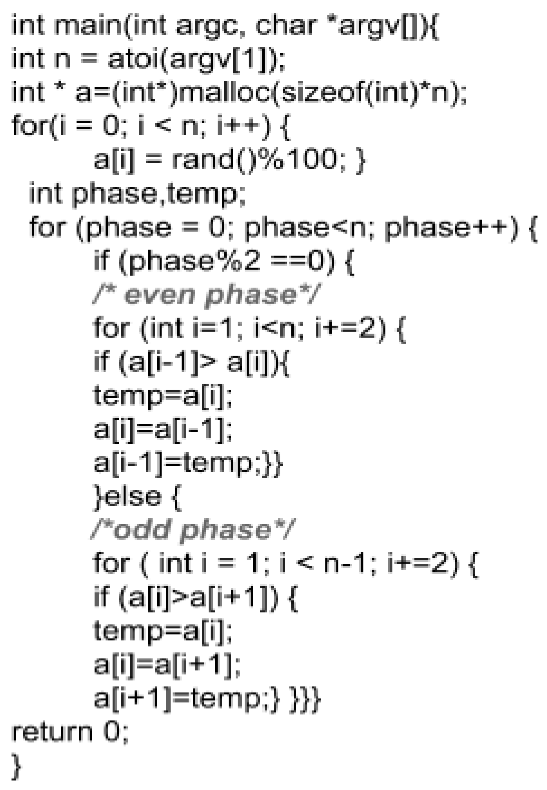
How implement this sorting in MPI (parallel code):

We starts by distributing n/p sub-arrays of our array (p is the number of processors and n the size of the array to sort) to all the processors. we scatter the data by sending it from the process 0 (the root of the processes with which we launch our computation) to each of the other processes by using MPI Scatter() function. After that each process has to receive this data and each process sort sequentially its sub-array.
Then we move between odd and even phases and we have one function to send data to the next process and keep the lower part of the arrays and another one to send data to the previous process and keep the higher part.

- ➢ In the even phase, we have a one sided communication between an even process i communicate and the next odd process i+1. We have then a function to send data to this next process and keep the lower part of the array (The upper half of the array is then kept in the higher process i+1 and the lower half is put in the lower process i). so an even process will have a partner of rank i + 1. An odd process will have also a partner of rank i - 1.
- ➢ In the odd phase, we have a one sided communication between an odd process i and the previous even process i-1 in exactly the same way as in the even phase. Here an even process will have a partner of rank i- 1. and an odd process will have a partner of rank i + 1.

Finally, we will have a sorted array with the lower part of the array stored in the process 0 and the higher part of the array stored in the process p-1, we use then MPI-Gather() to gather the two parts of the array and to have the final array that is already sorted.

<u>How implement this sorting (sequential code):</u>



```
int main(int argc, char *argv[]){
int n = atoi(argv[1]);
int * a=(int*)malloc(sizeof(int)*n);
for(i = 0; i < n; i++) {
        a[i] = rand()%100; }
  int phase,temp;
  for (phase = 0; phase<n; phase++) {
        if (phase%2 ==0) {
        /* even phase*/
        for (int i=1; i<n; i+=2) {
        if (a[i-1]> a[i]){
        temp=a[i];
        a[i]=a[i-1];
        a[i-1]=temp;}}
        }else {
        /*odd phase*/
        for ( int i = 1; i < n-1; i+=2) {
        if (a[i]>a[i+1]) {
        temp=a[i];
        a[i]=a[i+1];
        a[i+1]=temp;} }}}
return 0;
}
```

The theoretical complexity of the algorithm:

The performance of the odd-even transposition algorithm is:

$$\sum_{i=1}^{n/p} i = 1+2+3+........+n/p = \frac{n/p(n/p-1)}{2} = \Theta(\frac{n^2}{2*p*p})$$

The efficiency of the odd-even transposition algorithm in parallel is:

$$\Theta(\frac{n*log(n)}{2*n*n})$$

Practical evaluation of performances for both sequential and parallel algorithm:

As we said earlier, we work here with the Grid5000 when we used a CPU Intel Xeon Gold 5220 with 18 cores, with RAM of 96 Gbps and a network of 2*25*Gps.

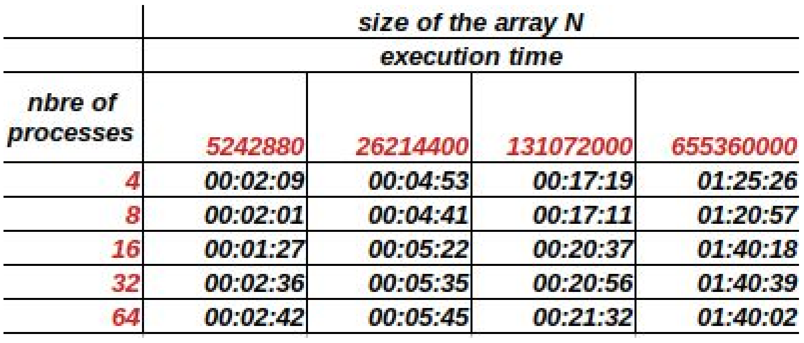
we alternate between different sizes of arrays and different numbers of processes, we take for this evaluation:

- As a size of arrays :[32768, 65536, 131072, 262144,524288, 1048576] for small arrays, and [5242880, 26214400, 131072000] for large arrays.
- As a number of processes: 4 , 8 , 16 , 32 , 64 .

From what we can see, the execution time of the parallel algorithm compared to the sequential one is pretty good, this good performance can be seen on large scale when we use large arrays for example for an array with size of 5242880 on average in a cluster of 8 processes the parallel algorithm run in 1.99 s, while the sequential one run in 2 minutes and 21.72 s on a single machine.

we conclude then that using parallel algorithm for large arrays is beneficial in term of performances.

The problem was when we use different processes in parallel algorithm, we can see that if we use 4 or 8 or 16 processes we have less execution time, but when we use 32 or 64 processes the execution time become higher .



	size of the array N			
	execution time			
nbre of processes	5242880	26214400	131072000	655360000
4	00:02:09	00:04:53	00:17:19	01:25:26
8	00:02:01	00:04:41	00:17:11	01:20:57
16	00:01:27	00:05:22	00:20:37	01:40:18
32	00:02:36	00:05:35	00:20:56	01:40:39
64	00:02:42	00:05:45	00:21:32	01:40:02

table: execution time in different processes for large arrays

This can be explained by the number of nodes that we use, we already said that we use 18 cores per node, so for 4/8/16 processes we are in the same machine so if the number of processes become higher the execution time decrease, but when we use 32/64 we have then different nodes so the cost of the communication increase, this leads to an increase in the execution time also.