Parallel Computation of Fast Fourier Transformation

Abstract: In this rapport, two parallelization algorithms of the Cooley-Tukey fast Fourier transformation algorithm, binary exchange algorithm and transpose algorithm, are presented and analyzed. Their advantages and disadvantages are discussed and the complexities are computed.

**Background**

The Fourier transformation is used in many engineering problems. The applications cover from the earphone designing all the way to the turbulence simulation. A fast and scalable algorithm to calculate the Fourier transformation is thus very important. There are several methods to calculate the Fourier transformation of the inputting function, one of the most commonly used is the ‎Cooley-Tukey Fast Fourier transformation algorithm (Shorten to the FFT in the following text). It is the method that is analyzed in this rapport.

The reason for developing parallelization of the FFT is that the size of the inputs grown dramatically because of the requirements on the size and accuracy of the application. Sequential algorithms are becoming slow or not even feasible. A parallel algorithm can compute larger problems faster.

**Sequential version of Cooley-Tukey Fast Fourier transformation**

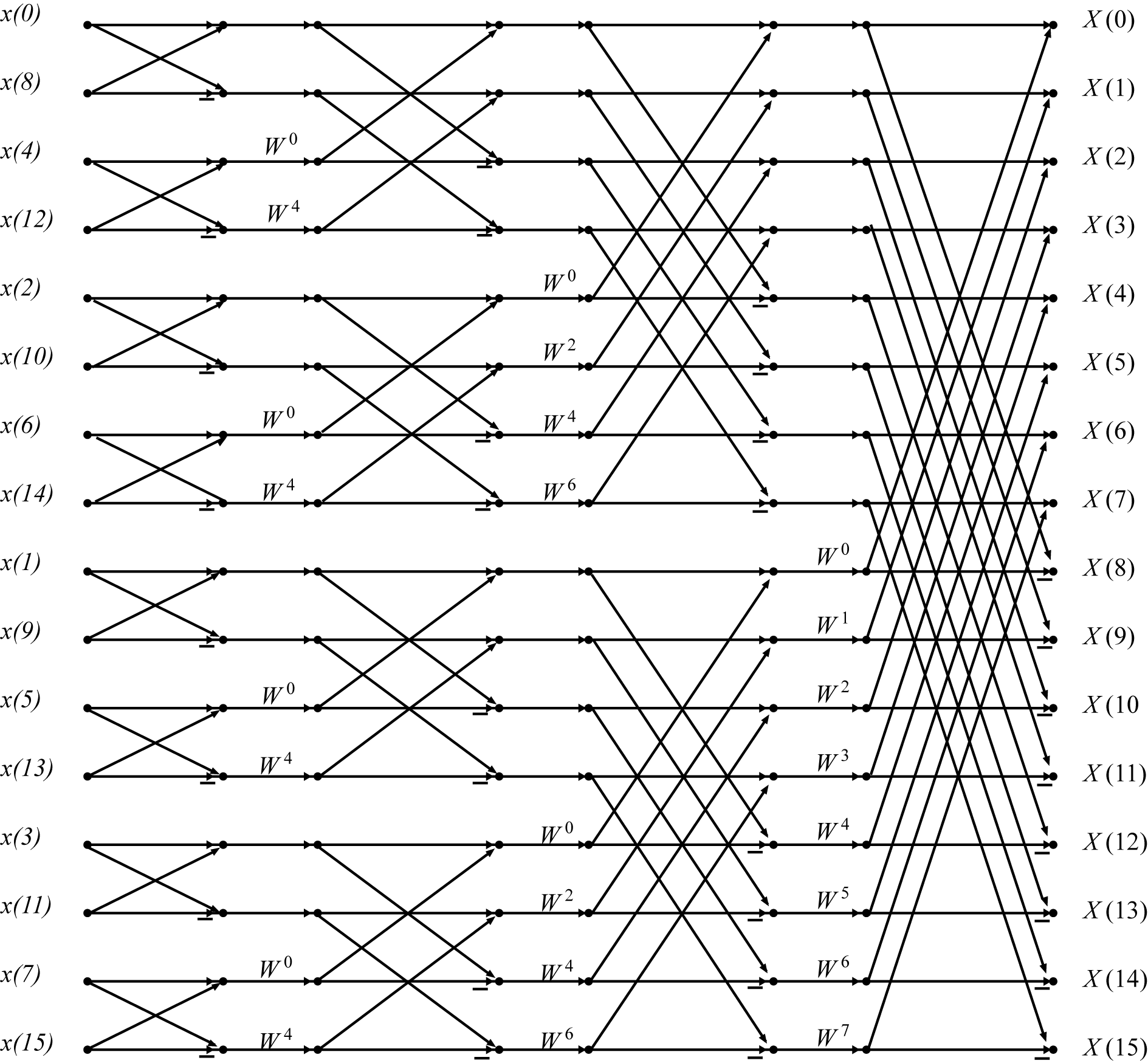
The FFT applies the basic idea of divide-and-conquer. It divides the inputs into smaller sets where the algorithm can be used recursively. For simplicity, assuming that the input function is a 1D function defined on region . The input to the FFT is given by values of function where . goes from to .

The discrete Fourier transformation (DFT) is defined by the following formula:

The FFT algorithm is based on the idea that rearrange the DFT of the function into two parts: a sum over the even-numbered indices and a sum over the odd-numbered indices .

Because of the periodicity of the complex exponential, can also be computed as:

Then, we get: . Thus, the computation can be divided into multiple stages of iterations of computing and . Each iteration includes computation of exponential and a summation independently. Each summation is done for a pair of elements so all summations can be done parallelized. A graph showing this summation is showed below.



This exchange algorithm is called binary exchange algorithm, also butterfly algorithm. The word binary comes from that the summation is always done between two elements that have their binary representations differs by one number.

From the graph we can see that the input is not sorted if we want the output to be sorted and this conclusion is also true verse vice. So how the data partition between processors is done greatly influences the amount of communication needed. One trivial solution is the trivial binary exchange algorithm that utilize the idea showed in the graph above. The other one is the transpose method. Both of these method shall be implemented and compared later in this rapport.

**Binary exchange algorithm**

We assume that there are input data and the number of processors is . To make the implementation simpler, we assume that and is dividable by . Thus, each process shall hold data. We can assume thus .

To minimize the communications, we would want each process initially contain those data that allow it to perform computation until the stage the communication is inevitable. For example, when and then the partition should be , ,, as the graph illustrate above. Then two times of communication would be required to get the final result.

The numbering of the input data is given by the binary reverse of the number from to , This means that the th input data, where has the binary representation shall be placed at the location where has the binary representation . After this permutation, the data is partitioned evenly to all processors according to this ordering.

The total amount of iterations needed is and of them are done within each processor. So, the total communication needed is . In each communication, data is transferred. To avoid dead lock, the exchanging is done with red-black algorithm thus the total communication time required is then where is the time required to transfer one data between two processors.

The time complexity of the computation at each node before communication is where is the time to perform the computation once on a stage because the serial algorithm has the complexity . The parallelized part has the computation time because totally we have communications. Thus, the total computation time is .

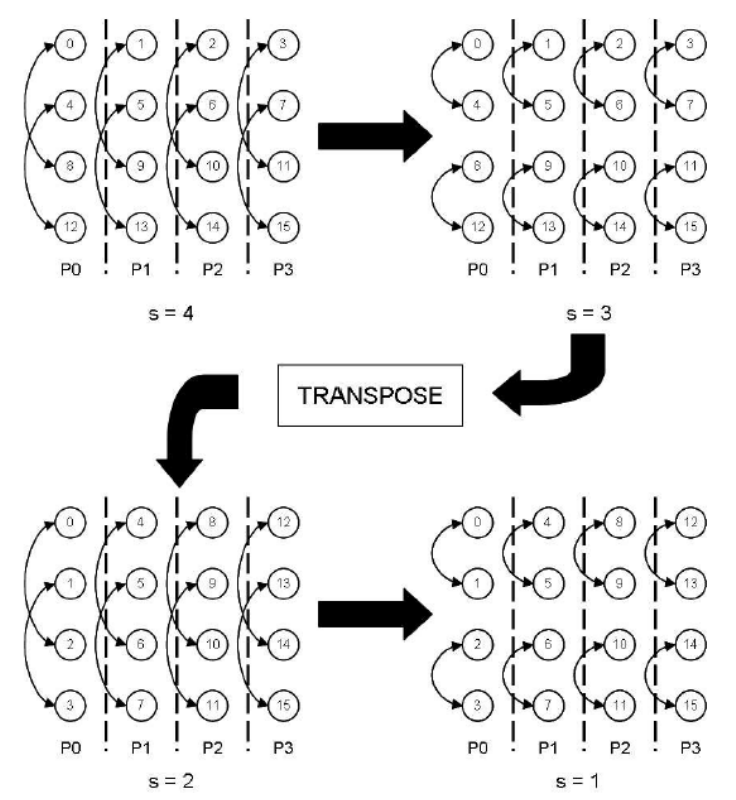
So, the total computation time is . The original sequential computation time is . So, the speed up is:

From the architecture of the computer we know that , then

If the communication is very slow compare to the computation then the speedup would be a lot less than because of the communication overhead.

**Parallel Transpose Algorithm**

To reduce the communication overhead, there is another partition algorithm – parallel transpose algorithm. The idea behind this algorithm is that to partite the data on processors in such a way that only one global communication is required. The global communication has a structure similar to transposing of a matrix, thus the algorithm is called transpose algorithm. It is described as following.



As the graph above showed. Assuming the number of input data is and here is an integer. The data is listed in a square matrix. The number of the processors is assumed to be and the data is partitioned into regions. Each region contains same number of columns from left to right. Then the FFT can be performed within each processor until external data needed. Then the processors perform a partition matrix transposition as the graph above shows. The resulting data will allow further FFT to get the final results.

The total number of the computation is the sequential time completely parallelized, i.e. . The communication is done only once with all data except those on the diagonal exchanged. The communication is done by pairs of processors so it can be fully parallelized. The communication time thus is . So, the total running time is and the speedup is:

If :

Thus, the speedup is improved roughly by a factor of which is not bad if is large.

**Implementation**

These two algorithms are both implemented with MPI in C. To compare the performance of them, both implementations shall be timed. The input will be a file which first contains an integer which means that the file contains data points, then follows by lines of double number giving all . The input data is generated with another program.

The timer starts when the reading from the input is done and end when the writing to the output is done. The time the computation took is also written in the file.

To make a rough estimation, several inputs are tested. There will be two different function to generate the input data, and where . There will be three sizes of the inputs , , . The program is tested on , , processors.

**Results**