The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology

and high-performance computing”

# NUMERICAL METHODS FOR SOLVING DIFFERENTIAL EQUATIONS

*Practice 4. Fast Fourier Transform for the Problem of Heat Diffusion in a Plate*

Nizhni Novgorod

2014

## OBJECTIVES

The purpose of this practice is to study the Fast Fourier Transform-based method for numerical solution of differential equations as illustrated by the problem of solving Poisson's equation with homogeneous boundary conditions and approaches to its parallelization.

## ABSTRACT

The practice deals with the stationary problem of heat diffusion in a square plate with zero temperature conditions at its edges (Dirichlet problem). It defines the problem and derives a difference scheme to solve it. Solving the difference equation is based on expansion in basic functions with Fourier coefficients. The computational method consists in multiple use of the tridiagonal matrix algorithm and Fourier transform. This practice involves studying Fourier transform implementation in Intel MKL and FFTW. You will develop a sequential implementation of the proposed problem solution algorithm and evaluate efficiency of the Intel MKL and FFTW sequential solvers. After this, you will develop a parallel version of the program and analyze its scalability.

## GUIDELINES

The practice deals with the stationary problem of heat diffusion in a square plate with zero temperature conditions at its edges: $\Delta U = U_{xx} + U_{yy} = -f(x,y),\ (x,y): x \in [0,l], y \in [0,l]$, $U(0,y) = \mu_1(y),\ U(l,y) = \mu_2(y),\ U(x,0) = \mu_3(x),\ U(x,l) = \mu_4(x)$. Its numerical solution involves differential equation approximation by a difference scheme within a uniform grid with the step $h$ in the $x$ and $y$ variables ($h = \frac{l}{n}$ ). The difference scheme is based on a five-point stencil.

For numerical solution of this problem, it is proposed to consider the grid functions $v_{ij}$ and $f_{ij}$ as single-dimension grid functions of $j$. Then, for each $i$ they can be expanded in eigenfunctions of the auxiliary problem as their eigenvalues using the Fourier coefficients. The problem is solved in three stages:

- finding the right-hand part Fourier coefficients $i = \overline{1, n-1}$ using the Fourier transform;
- finding the Fourier coefficients $c_k(i)$ of $v_{ij}$ having solved the tridiagonal linear system for each $k = \overline{1, n-1}$;
- restoring the solution $v_{ij}$ by the Fourier coefficients $c_k(i)$ using the Fourier transform for each $i = \overline{1, n-1}$.

The complexity of this algorithm is $O(n^2 \log_2 n)$ .

The practice describes the use of Intel MKL [2] и FFTW [5] libraries to compute the Fast Fourier Transform (FFT) and discusses the issues of libraries building and linking.

The use of MKL FFT requires calling the following sequence of functions:

1. **`DftiCreateDescriptor()`** – function to create the FFT descriptor. The function has 5 input parameters:
   - pointer to the descriptor object where the transform handle will be stored;
   - precision of the transform (single or double);
   - forward domain of the transform (complex or real);
   - dimension of the transform;
   - length of the transform for a one-dimensional transform. Lengths of each dimension for a multi-dimensional transform.

2. **DftiCommitDescriptor()** – descriptor initialization. The function input is the descriptor object.
3. **DftiComputeForward**/**DftiComputeBackward()** – forward/backward Fourier transform. The function parameters are the descriptor object and pointer to the output array.
4. **DftiFreeDescriptor()** – descriptor release.

The descriptor object type is **DFTI_DESCRIPTOR_HANDLE.** Additional transformation parameters can be determined using **DftiSetValue()**.

The Fastest Fourier Transform in the West (FFTW) library is a set of open C and Fortran modules for FFT computation. A typical application scheme of FFT being a part of the FFTW library is based on the following sequence of functions:

1. Creation of a so-called FFT computation schedule, i. e. setting input and output data as well as transform parameters. This is done by means of either

   **fftw_plan_dft[_<dim>]()** in case of complex input and output signals; or

   **fftw_plan_dft_<type>[_<dim>]()**, in other cases.

   Here, **<dim>** is the FFT dimension and **<type>** is the transform type.

   The inputs of these functions are the number of elements for each transform dimension, pointers to the memory allocated for the input and output data sequences, transform types and a special flag. The function returns a computation schedule of the **fftw_plan** type.
2. Execution of this schedule is by calling **fftw_execute()**. The input of this function is the computation schedule.
3. Release of the schedule and all the related auxiliary data by **fftw_destroy_plan()**.

This will be followed by implementing a sequential version of the proposed algorithm. The practice describes a general implementation scheme and use of Intel MKL and FFTW libraries to compute Fourier coefficients of the problem in question.

**ComputeDecision()** contains the complete differential equation solution path. To ensure implementation flexibility, **SinFT**, a pointer to the function whose inputs are two pointers to **double** type arrays of a length **n**, is introduced. The function computes factorization of the first array and stores the respective coefficients in the second one. **SinFT** points to a certain FFT implementation depending on the command line parameter value. **SinFT_MKL()** and **SinFT_FFTW()** compute sums of the form $s_j = \sum_{k=1}^{n-1} z_k sin\frac{\pi kj}{n}, j = \overline{1, n-1}$ using the technique described in [**Ошибка! Источник ссылки не найден.**]. It consists in reducing the vector of coefficients $z_k$ to the vector that enables determining the coefficients $s_j$ as a result of FFT. For this purpose, it is proposed to extend the array $z_k, k = \overline{1, n-1}$ to reach the length of $2n$ with $z_0 = z_n = z_{n+1} = \cdots = z_{2n-1} = 0$. The Fast Fourier Transform will result in the array $v_j, j = \overline{0,2n-1}$ with $Imv_j = s_j, j = \overline{1, n-1}$. If $n = 2^m, m \in \mathbb{N}$, all computations will take $O(n\log_2 n)$. The bodies of **SinFT_MKL()** and **SinFT_FFTW()** are subject to creation of an extended array $z_k, k = \overline{1,2n-1}$, call of the corresponding library FFT computation function and forming of the sought array $s_j, j = \overline{1, n-1}$.

The tridiagonal linear system is solved using the MKL library. This is done by **dgttrfb()**, a function that computes $L_1UL_2$ factorization of the tridiagonal matrix and **ddttrsb()**, a function that solves the linear system based on the known $L_1UL_2$ factorization of the tridiagonal matrix.

The proposed method is parallelized separately for each computation stage. In this case, parallelization is based on OpenMP. Iterations inside the loops that compute Fourier coefficients are data-independent, so they can be distributed among independent threads. For this purpose, add `pragma omp parallel for` to `ComputeDecision()` before each problem solution stage performed within the loop. In this case, the memory for the arrays storing the auxiliary tridiagonal linear system will be allocated individually to each thread.

The practice compares the results computed by means of MKL and FFTW-based Fourier transform and those computed using own implementation from "Development, Optimization and Parallelization of the Fast Fourier Transform as Applied to the Filtration Problem". Experiments were performed within grids ranging from $128 \times 128$ through $8192 \times 8192$. Experimental results showed that the MKL-based implementation is the most efficient, the FFTW-based one is 1.4 times less efficient, while own implementation is 1.8 times less efficient for the largest available grid sized $8192 \times 8192$ .

The parallel program version for the own FFT implementation is compared to the MKL-based implementation for 1 through 8 threads. For the own implementation the highest speed up equals to 5.4 was obtained in case of 8-thread computation within a $8192 \times 8192$ grid. In this case, monotonic speed up growth in relation to the number of threads is observed within grids whose dimensions exceed $1024 \times 1024$ . For the MKL implementation the highest speed up equals to 4.8 was obtained in case of 7-thread computation within a $512 \times 512$ grid. For 8-thread computation, the highest speed up is 4.3 for a $2048 \times 2048$ grid. In this case, monotonic speed up growth in relation to the number of threads is observed within grids whose dimensions exceed $2048 \times 2048$ . For 8 threads, the gap between the own and the MKL implementation reduced to 1.3 times.

## RECOMMENDATIONS FOR STUDENTS

See [2, 3] for the methods for solving differential equations by the finite difference method. Study [1] for the methods for solving linear systems. Visit [4-6] for the documents related to libraries used in this practice.

## REFERENCES


1. Golub G.H., Van Loan Ch. F. Matrix Computations. The John Hopkins University Press, 1996.
2. Hoffman J.D. Numerical Methods for Engineers and Scientists, 2nd Edition. New York: CRC Press, 2001.
3. Kincaid D.R., Cheney E.W. Numerical Analysis: Mathematics of Scientific Computing, 3rd Edition. Pacific Grove: Brooks Cole, 2001.
4. Intel® Math Kernel Library documentation [http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/].
5. Fastest Fourier Transform in the West (FFTW) official page [http://www.fftw.org/].
6. Fastest Fourier Transform in the West documentation [http://www.fftw.org/fftw3.pdf].


## PRACTICE

1. Develop an implementation that uses ordinary single-dimension arrays instead if pointer arrays. Evaluate its efficiency.
2. Modify the developed program for a $l_1 \times l_2$ rectangular plate.
3. Solve the initial problem subject to inhomogeneous boundary conditions.

4. Develop your own FFT implementation that will work for a random length input sequence.
5. Implement the FFT algorithm to base 4. Evaluate the efficiency of this implementation as compared to implementations described as part of this practice. Optimize and parallelize the algorithm.

## TEST

1. What is the complexity of computation of $n$ Fourier coefficients using FFT?
    a. $O(\log_2 n)$
    b. $O(n^2 \log_2 n)$
    c. + $O(n \log_2 n)$
2. How many times is the Fourier Transform performed in the described algorithm for a $n \times n$ grid?
    a. $n$ time
    b. $2n$ time
    c. + $2(n-1)$ times
3. What Fourier Transform type is supported by MKL and FFTW?
    a. + Fourier transform in both complex and real domains
    b. Fourier transform in the real domain
    c. Fourier transform in the complex domain
4. What is NOT an input parameter of **`DftiCreateDescriptor()`** of MKL?
    a. + descriptor object
    b. transform dimension
    c. number of sequence elements
5. What invariable cannot be set by **`DftiSetValue()`** of MKL?
    **a.** + **`DFTI_PRECISION`**
    b. **`DFTI_FORWARD_SCALE`**
    c. **`DFTI_PLACEMENT`**
6. What is the name of the FFTW static library file containing Fourier transform implementation for the double precision elements?
    a. **libfftw3r-3.lib**
    b. **+ libfftw3l-3.lib**
    c. **libfftw3d-3.lib**
7. What is returned by **`fftw_plan_dft_1d()`** from the FFTW library?
    a. nothing
    b. error code
    c. + computation schedule
8. What does **ddttrsb()** of MKL do?
    a. $L_1UL_2$ -factorization of tridiagonal matrices
    b. + solving a linear system with the tridiagonal matrix for which $L_1UL_2$-factorization has been found
    c. solving a linear system with the tridiagonal matrix for which LU-factorization has been found
9. What is the size of the input array for the Fourier transform computation in this problem?
    a. + $2 * n$

b. $n$
c. $2 * (n - 1)$

10. How is the memory to store the tridiagonal matrix allocated among the threads in the proposed parallel algorithm implementation?
    a. Each thread receives a pointer to the general matrix
    b. + Each thread receives a pointer to its own matrix
    c. Each thread receives a pointer to its own fragment of the general matrix.