# Parallel Computation of Fast Spectral Transforms of Logic Functions using the MPI Framework

Miloš Radmanović[1], Radomir S. Stanković[2], Dušan B. Gajić[3]

*Abstract* – **In many practical applications in signal processing, digital system design, logic design, pattern recognition, and related areas, it is often essentially important to be able to efficiently compute spectral transforms for logic functions. The corresponding Fast Fourier transform (FFT)-like algorithms for computation of various spectral transforms can be efficiently adapted to parallel computation platforms. In this paper, we investigate parallel implementation of two classes of FFT-like algorithms, the Cooley-Tukey algorithms, and the constant geometry algorithms, for computing the Reed-Muller, the Walsh, and the arithmetic transforms on multicore Central Processing Units (CPUs) using the Message Passing Interface (MPI) framework. The paper also discusses certain specific parallel implementation styles of programming FFT-like algorithms on multi-core CPU platforms. Performance of the MPI implementations is compared with the classical C++ implementations for the single-core CPUs. It is shown that parallel MPI programming provides significant speedups for one of the considered implementation styles of the Cooley-Tukey algorithms. Other MPI implementations have a negative impact on the performance of both Cooley-Tukey and constant geometry algorithms.**

***Keywords*** **– Logic functions, spectral transforms, FFT-like algorithms, multicore CPU, MPI.**

## I. INTRODUCTION

Spectral transforms have many applications in signal encoding and processing techniques, synthesis, verification, and testing of logic circuits [1], [2], and many other areas [9], [10]. Due to rapidly increasing complexity of logic circuits and systems, in recent years, there has been a renewed study in spectral transforms for logic functions. For practical applications, it is often necessary to be able to efficiently compute these transforms. There is a variety of algorithms for efficient calculation of these transforms: FFT-like algorithms using truth vector representations of functions [3], fast tabular techniques, calculation algorithms through reduced representations of logic functions, and binary decision diagrams [2], [9]. The FFT-like computation has been one of the most popular numerical methods applied in almost every field of science. The FFT-like computation can be executed much faster by using parallel processing [4], especially nowadays, as many supercomputing facilities are available to scientists and engineers across the world. Furthermore, the multicore desktop computers offer an inexpensive capability of parallel processing. Parallel computing on multicore CPUs enables parallel processing on commodity hardware. Only very recently the possibility of using multicore CPUs to solve complex problems in logic design has been explored by many researchers, for example in [5], [6].

Moreover, inspired by efficient execution of parallel problems in logic design and possibility of using multi-core CPUs platform, in this paper we investigate two classes of parallel FFT-like algorithms for computing spectral transforms of logic functions using the MPI framework. Particularly, in the case of the Cooley-Tukey algorithms, we investigate three implementation styles for the efficient parallel computation of the Reed-Muller, the Walsh, and the arithmetic transforms of logic functions using a multi-core CPU computation platform. Fast Reed-Muller, Walsh, and arithmetic transforms have the same time complexity of $O(N\log_2 N)$, where $N = 2^n$ is the size of the truth vector, and $n$ is the number of variables in the function. These spectral transforms have different transform matrices that are Kronecker product representable.

The paper also investigates mappings of two distinct FFT-like algorithms, the Cooley-Tukey class and the constant geometry class algorithms, to the multi-core CPU computing model. There are many approaches for implementation of parallel Cooley-Tukey class FFT-like algorithms and they can be categorized into three styles for parallelizing butterflies of FFT: block data mapping, cycling data mapping, and all butterflies parallel mapping [7].

Performances of the MPI implementations of two classes and three programming styles of FFT-like algorithms for the efficient parallel computation of Reed-Muller, Walsh, and the arithmetic transform of logic functions are compared with the classical C++ implementations on the single-core CPU. The idea behind the selection of these transforms is to compare the performance of their implementations since they have FFT butterflies of different computational complexity.

The paper is organized as follows. Section 2 shortly introduces the fast spectral transforms for logic functions and illustrates by examples FFT-like algorithms for efficient parallel computation of the Reed-Muller, the Walsh, and the arithmetic transforms. In section 3, parallelization of FFT-like algorithms for these transforms is discussed. Section 4

[1]Miloš Radmanović is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: milos.radmanovic@elfak.ni.ac.rs

[2]Radomir Stanković is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: radomir.stankovic@gmail.com

[3]Dušan Gajić is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: dusan.gajic@elfak.ni.ac.rs

presents experimental testing of mappings of these FFT-like algorithms with various data mapping styles to the multi-core CPU computing model using MPI framework. Section 5 offers some concluding remarks and directions for future work.

## II. Fast Spectral Transforms of Logic Functions

Spectral transforms of logic functions are an efficient tool in solving many tasks in logic design [2]. Spectral transforms defined by the Kronecker product representable transforms matrices have found many practical applications. The most common reason for this is existence of efficient calculation algorithms for these transforms.

In this paper, we discuss three different kinds of spectral transform of logic functions: the Reed-Muller, Walsh, and the arithmetic transforms. These transforms have different transform matrices that are Kronecker product representable.

The Reed-Muller transform [2] represents an important operator for obtaining AND-EXOR expressions of logic functions. The Reed-Muller transform matrix of order $n$, denoted by $R(n)$, is defined recursively as:

$$R(n) = \bigotimes_{i=1}^{n} R(1),\ R(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \quad (1)$$

The arithmetic transform [2], which is also known as the integer Reed-Muller transform, was initially introduced to represent multiple-output functions by a single polynomial for the equivalent integer functions. The arithmetic transform matrix of order $n$, denoted by $A(n)$, has a recursive structure analogous to that of the Reed-Muller transform and is defined as:

$$A(n) = \bigotimes_{i=1}^{n} A(1),\ A(1) = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}. \quad (2)$$

The Walsh transform [6] is based on a set of orthogonal functions defined by J. L. Walsh which are an extension of a set of functions defined by H. Rademacher. Analogously to previous transforms, the Walsh transform matrix of order $n$ in Hadamard ordering, denoted by $W(n)$, is defined as:

$$W(n) = \bigotimes_{i=1}^{n} W(1),\ W(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (3)$$

The spectrum of a logic function $f$ given by truth vector $F = [f(0), f(1)_{,\ldots,} f(2^n - 1)]^T$ is computed as:

$$S_f = T(n)F, \quad (4)$$

where $T(n)$ is any of the three matrices $R(n)$, $W(n)$, and $A(n)$, with computations performed in $GF(2)$ for the Reed-Muller transform, and in the set of rational numbers for the Walsh and the arithmetic transforms.

The FFT developed in signal processing for computing the Discrete Fourier transform (DFT) can be used to compute the coefficients in spectra of logic functions by varying just the basic kernels of the algorithms and ranges were the computations are performed [1], [2], [9]. The Reed-Muller, the arithmetic, and the Walsh transform matrices, expressed in (1), (2) and (3) respectively, can be factorized in different ways yielding different fast transform algorithms, the so-called FFT-like algorithms [8]. In this paper, we consider the Cooley-Tukey and the constant geometry algorithms for the Fast Reed-Muller transform (FRT), the Fast Arithmetic Transform (FAT) and the Fast Walsh Transform (FWT) algorithms. The selection was made in order to compare the performance of different parallel implementations on the multi-core CPU computation platform using MPI framework.

First, we consider the Cooley-Tukey (CT) algorithms, based on the Good-Thomas factorization which originates from the Kronecker product structure of the transform matrix [2]. Figure 1 shows the elementary butterflies operations (flow-graphs) for the Reed-Muller, the arithmetic and the Walsh basic transform matrices, respectively. In this figure, the solid and the dotted lines carry positive (+1) and negative (-1) weight, respectively. Note that, the operations for Walsh and arithmetic elementary butterfly are over integers, while the Reed-Muller butterfly uses $GF(2)$ operations.

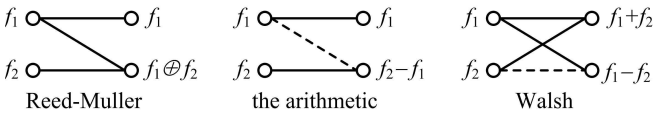


Fig. 1. The elementary butterfly operations for basic Reed-Muller, the arithmetic and Walsh transform matrices.

Figure 2 shows the flow graphs of the FFT-like Cooley-Tukey algorithm for the computation of the Reed-Muller spectrum of a three-variable function $f$ given by the truth vector $F = [f(0), f(1)_{,\ldots,} f(7)]^T$.

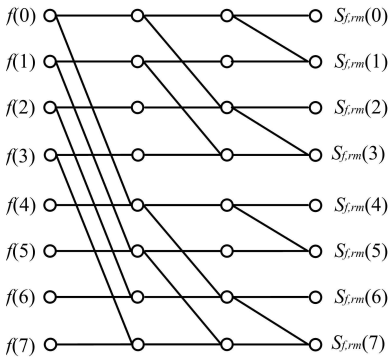


Fig. 2. The flow graphs of the FFT-like Cooley-Tukey algorithm for computing the Reed-Muller spectrum of a tree-variable function.

The constant geometry (CG) FFT-like algorithms are based on a factorization of the transform matrix into identical factor matrices [2]. Therefore, the indices of the butterfly operations are fixed for each step producing lower arithmetic complexity of algorithm. Because the results of butterfly

operations cannot be written in the same memory locations where the function is stored, implementations of this class of algorithms results in increased memory requirements compared to the Cooley-Tukey algorithms.

Figure 3 show the flow graphs of the FFT-like constant geometry algorithm for computing the Walsh spectrum of a three-variable function *f*.

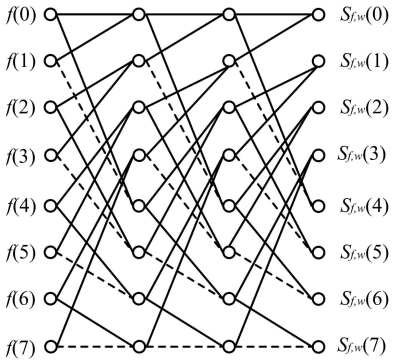


Fig. 3. The flow graphs of the FFT-like constant geometry algorithm for computing the Walsh spectrum of a three-variable function.

## III. Parallelization of the FFT-like Algorithms of Spectral Transforms

The FFT-like algorithms have a large degree of parallelism in each step of the flow graph and according to this, their implementation on parallel computers has been well studied. A fundamental step in parallelizing the FFT-like algorithms on multicore CPUs is the mapping of array addresses to cores.

There are many approaches for implementation of parallel Cooley-Tukey class FFT-like algorithms on multi-processing elements and they can be categorized into three styles for parallelizing butterflies of FFT: block data mapping, cycling data mapping, and all butterflies in parallel mapping [7]. Figure 4 shows the flow graphs of the FFT-like algorithm of Cooley-Tukey class with various data mapping styles for the computation of Reed-Muller spectrum of a three-variable function $f$ executed on two processing elements.

Block data mapping (BDM) style for parallelizing butterflies of Cooley-Tukey class of FFT-like algorithms (Figure 4a) uses an approach where the scope for parallelism increases with steps of the flow graph. In the first step, all the data is intertwined and all butterflies are performed on one processing element. In the second step, butterflies visibly break into two separate processing elements, and in the last stage there are two points of parallelization.

Cycling data mapping (CDM) style for parallelizing butterflies of Cooley-Tukey class of FFT-like algorithms (Figure 4b) uses an approach where the scope for parallelism decreases with steps of the flow graph. In the first step, all the butterflies are distributed individually to processing elements. In the second step, butterflies visibly break into two separate points of parallelization and, in the last step, there are four points of parallelization.

All butterflies in parallel mapping (BPM) style for parallelizing butterflies of Cooley-Tukey algorithms (Figure 4c) uses an approach where all butterfly computations can be performed in parallel. While this mapping style provides maximum scope of parallelism, there is a significant additional computation of memory addresses for each step producing higher overall arithmetic complexity of the algorithm.

Usually there is only one implementation style for parallel constant geometry class FFT-like algorithms, since the control flow is the same in each step.

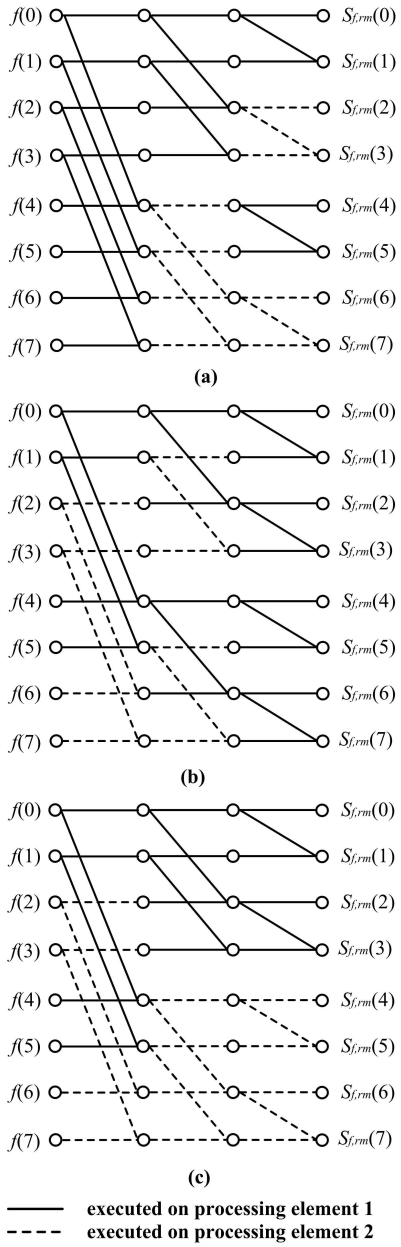


Fig. 4. The flow graphs of the parallel FFT-like algorithm of CT class for computation of the FRT of the tree-variable function executed on two processing elements: (a) BDM, (b) CDM, (c) BPM.

## IV. Experimental Results

For multi-core CPU architectures, the model of parallel processing is based on a large number of processor cores with the ability to directly address into a shared RAM memory. The MPI framework has become a widely used standard, though not necessarily the best framework for parallel programming. For comparison purposes, we developed referent C++ and MPI implementations of FFT-like algorithms using two classes of algorithms and three implementation styles of the fast Reed-Muller, the fast Walsh, and the fast arithmetic transforms. The computations are performed on an Intel desktop PC working at 3.66 GHz with 12 GBs of RAM. The quad-core CPU that is used is an Intel i7 with hyper-threading, yielding 8 logical cores (threads).

We compared the performance of a multi-core CPU accelerated MPI implementations to a single-core CPU C++ implementations for a sample set of random logic functions. Since the computations are performed over vectors, processing times are independent of function values and, therefore, are performed using randomly-generated function truth vectors.

Table 1 shows computation performance of FFT-like algorithms using various classes and implementation styles discussed in the previous section. All times in the table are given in seconds. The data in the table are horizontally sorted in the increasing order of the number of functions variables.

TABLE I
COMPUTATION TIMES OF THE REED-MULLER, THE WALSH, AND THE ARITHMETIC TRANSFORM USING VARIOUS CLASSES OF FFT-LIKE ALGORITHMS AND IMPLEMENTATION STYLES ON SINGLE-CORE CPU AND MULTI-CORE CPU PLATFORM

| FFT-like algorithm | | | | Computation time [s] | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | Number of variables *n* | | | | |
| Transform | Algorithm | Style | CPU cores | 25 | 26 | 27 | 28 | 29 |
| FRT | CT | | 1 | 0.6 | 1.3 | 2.0 | 3.9 | 10 |
| FRT | CG | | 1 | 0.9 | 1.8 | 4.0 | 6.5 | 13 |
| FRT | CT | BPM | 8 | 0.6 | 1.2 | 1.9 | 3.8 | - |
| FRT | CT | BDM | 8 | 0.1 | 0.3 | 0.4 | 0.7 | - |
| FRT | CT | CDM | 8 | 0.9 | 2.0 | 4.3 | 8.6 | - |
| FRT | CG | | 8 | 0.6 | 1.4 | 3.0 | - | - |
| FWT | CT | | 1 | 0.8 | 1.4 | 3.2 | 6.5 | 9.3 |
| FWT | CG | | 1 | 1.2 | 2.2 | 4.5 | 6.9 | 19 |
| FWT | CT | BPM | 8 | 0.8 | 1.3 | 2.0 | 4.0 | - |
| FWT | CT | BDM | 8 | 0.2 | 0.3 | 0.6 | 1.2 | - |
| FWT | CT | CDM | 8 | 1.2 | 2.4 | 4.8 | 9.7 | - |
| FWT | CG | | 8 | 0.7 | 1.4 | 2.9 | - | - |
| FAT | CT | | 1 | 0.5 | 1.0 | 1.7 | 3.8 | 7.2 |
| FAT | CG | | 1 | 1.0 | 1.9 | 3.8 | 7.9 | 16 |
| FAT | CT | BPM | 8 | 0.6 | 1.1 | 1.9 | 3.8 | - |
| FAT | CT | BDM | 8 | 0.1 | 0.2 | 0.4 | 0.7 | - |
| FAT | CT | CDM | 8 | 1.1 | 2.1 | 4.1 | 8.5 | - |
| FAT | CG | | 8 | 0.6 | 1.3 | 2.9 | - | - |

From the data in Table 1, it can be seen that, on this multi-core CPU platform, for all considered transforms, the BMD style for the Cooley-Tukey algorithms significantly reduces computation times when compared to single-core CPU implementation. In the case of CDM style for the Cooley-Tukey algorithms, computation times are increased for about 50 to 60%. Computation times of butterflies in BPM style are very close to the computation times in single-core CPU implementations. Table entries with dashes indicate that the implementation failed to complete for that particular benchmark because of running out of memory.

The constant geometry FFT-like algorithms are not suitable for an MPI parallel processing configuration, because they increase inter-shared memory communication delay.

## V. Conclusion

Computing power can be substantially increased through the exploitation of the inherent parallelism available in FFT-like calculations. However, an experimental performance analysis of the parallel FRT, FWT, and FAT algorithms has not been sufficiently investigated in a multi-core CPU environment. In this paper, we experimentally evaluated various implementations of fast algorithms for three different spectral transforms for logic functions. We analyzed the speedup obtained, by taking into account both the class of algorithms and the implementation styles.

It has been shown that the parallel MPI programming provides significant speedups only for block data mapping style of the Cooley-Tukey algorithms. Other implementation styles, for both Cooley-Tukey algorithms and constant geometry algorithms, have a negative impact on the performance of parallel program execution.

## References


[1] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD*, Springer, 2001.

[2] M. G. Karpovsky, R. S. Stanković, and J. T. Astola, *Spectral Logic and Its Applications for the Design of Digital Devices*, Wiley, 2008.

[3] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*, Bristol, Academic Press, 1985.

[4] M. Frigo, S. G. Johnson, "The Design and Implementation of FFTW3", Proc. of the IEEE, vol. 93, no. 2, pp. 216-231, 2005.

[5] C. Brunelli, R. Airoldi, and J. Nurmi, "Implementation and Benchmarking of FFT Algorithms on Multicore Platforms", Proc.Int. Symposium on System on Chip, pp. 59-62, 2010.

[6] Y. Zhou, J. Zhang, and D. Fan, "Software and Hardware Cooperate for 1-D FFT Algorithm Optimization on Multicore Processors", Proc. Int. Conf. on Computer and Inf. Technology, vol. 1, pp. 86-91, 2009.

[7] E. C. Chu, A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, CRC Press, 2000.

[8] J. Astola, R. S. Stanković, *Fundamentals of Switching Theory and Logic Design: A Hands on Approach*, USA, Springer, 2006.

[9] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier Transform – Algorithms and Applications*, Springer, 2010.

[10] A. Deb, S. Ghosh, *Power Electronic Systems – Walsh Analysis with MATLAB*, CRC Press, 2014.