Multicore GPU and CPU Platforms for Software Operations

Miloš M. Radmanovic
Faculty of Electronic Engineering
Aleksandra Medvedeva 14
18000 Niš, Serbia
milos.radmanovic@gmail.com

ABSTRACT: *Many scientific applications are depending on kernel for which we have used the Reed-Muller transform in this work. The function of the multicore GPU and CPU are used to enhance the capability of the software operations. We have introduced the parallel computing for the correct use of fixed polarity Reed Muller transforms of the Boolean operations. The search algorithms have impact on it and the best algorithm is implemented with sequential code and also by using the Message Parsing Interface framework. The final performance outcome is studied with the other computational outcome of the multicore CPU using time involved in computation.*

Keywords: Reed-Muller Transform, Parallel FFT-like Algorithms, Multicore CPU, MPI

Received: 15 March 2022, Revised 3 July 2022, Accepted 20 July 2022

DOI: 10.6025/pms/2022/11/2/41- 47

1. Introduction

Reed-Muller (RM) transform represents an important class of AND-EXOR expressions and it has been efficiently applied in many areas such as signal processing [1], [2], coding technique [3], [4], and computer aided design [5], [6], including synthesis, verification, and testing. Fixed polarity RM form is the Reed-Miller polynomial expression in which each variable has the same form [7]. Any Boolean function can be expressed by fixed polarity RM forms. There are 2n polarities for an n-variable Boolean function and the number of XOR terms depends on these polarities. Finding the best or optimal polarity can be very CPU time consuming, in order to search for the best polarity which will lead to the minimum number of XOR terms for a particular function. Therefore, there have been numerous algorithms developed to reduce the search time for finding the best polarity of RM forms. The most popular search criterion of the best polarity RM form is obtained by the exhaustive search of the all possible polarity vectors. An efficient exhaustive search of the all possible polarity vectors can be constructed by using the dual polarity search route method [8], [9], [10], [11], and transeunt triangle method [12], [13]. Instead of generating all of the polarity vectors and searching for the best polarity, many algorithms will find the non-exact optimal polarity using: the separation and sparse techniques [14], [15], quantum genetic and propagation of signal probability techniques [16], and technique based on Ordered functional decision diagrams (OFDDs) [17], [18], [19]. Furthermore, in [20] it has been developed non-exhaustive exact method for finding best fixed polarity Reed-Muller form directly from Walsh spectral coefficient, but only for 3-variable Boolean functions.

In general no efficient algorithm has been found that is able to obtain exact best polarity vector without constructing XOR terms for all polarities.

To solve large-scale problems parallel computing has been used efficiently either by distributing computational loads among processors or by utilizing the large memory in parallel networked workstations. During recent years, a large number of scientific algorithms and specific applications have been successfully ported to multicore CPUs and manycore GPUs platforms. Implementations on these platforms are recognized as having the potential to considerably speedup or accelerate compute intensive algorithms over their equivalent single CPU core implementations [21], [22]. 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 [23], [24].

Moreover, inspired by efficient execution of parallel problems in logic design and possibility of using multicore CPUs platform, in this paper it is proposed parallel implementation for computing the best fixed polarity Reed- Muller transforms of Boolean functions in order to execute it efficiently on multicore-CPU platform. The computation of best transform is based on exhaustive search algorithms and it is implemented both with sequential code and through MPI (Message Parsing Interface) framework for parallel algorithms development. The proposed implementation exploits the various points of parallelism that can be found in exhaustive search based algorithm for best RM transform and made an efficient mapping of the points to the multicore CPU architecture. The experimental results confirm that the application of the proposed MPI implementation of best RM transform on multicore CPU leads to significant computational speedups over traditional C/C++ implementations processed on single CPU. This paper is organized as follows: Section 2 shortly introduces the fixed polarity RM transform of Boolean function and illustrates polarity influence on the size of the resulting RM form. In section 3, multicore CPU platform is discussed. In Section 4, it is described a proposed MPI parallel implementation to be executed on a multicore CPU platform. The features of proposed implementation for the computation of the best RM transform were experimentally tested in section 5. Section 6 offers some concluding remarks and directions for future work.

2. Reed-muller Transform

A positive polarity Reed-Muller form (PPRM) is an exclusive-OR of AND product terms, where each variable only appeared in un-complemented form. Any n-variable Boolean function given by truth vector $F = [f_0, f_{1,\ldots,} f_{2^n-1}]^T$ can be represented by the PRM form in matrix notation defined as [6]:

$$f(x_1, x_2, \ldots, x_n) = X(n)A(n)F \tag{1}$$

where

$$X(n) = \bigotimes_{i=1}^{n} [1 \quad x_i] \tag{2}$$

and

$$A(n) = \bigotimes_{i=1}^{n} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{3}$$

where ⊗ denotes the Kronecker product, addition and multiplication are in modulo 2, A(n) represents the positive Reed-Muller transform matrix of order n, and A(1) is the basic positive Reed-Muller transform matrix.

A fixed polarity Reed-Muller form (FPRM) is an exclusive- OR of AND product terms, where each variable only appeared in complemented or un-complemented form, but not both. FPRMs are canonical representation of Boolean functions defined as [6]:

$$f(x_1, x_2, \ldots, x_n) = X_H(n)A_H(n)F \tag{4}$$

where

$$X_H(n) = \overset{n}{\underset{i=1}{\otimes}} \begin{bmatrix} 1 & x_i^{h_i} \end{bmatrix}$$
$$x_i^{h_i} = \begin{cases} x_i, h_i = 0 \\ \bar{x}_i, h_i = 1 \end{cases} \tag{5}$$

and

$$A_H(n) = \overset{n}{\underset{i=1}{\otimes}} A_i^{h_i}(1)$$
$$A_i^{h_i}(1) = \begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, h_i = 0 \\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, h_i = 1 \end{cases}, \tag{6}$$

where FPRMs are specified by the polarity vector:

$$H = [h_1, h_2, \ldots, h_n], \tag{7}$$

where the component $h_i \in \{0,1\}$ specifies the polarity for the variable x_i. If $h_i = 1$, then the *i*-th variable is represented by the complemented literal x_i, and when $h_i = 0$, by the uncomplemented literal x_i.

An FPRM transform can be given by the FPRM spectrum S_f^H calculated as:

$$S_f^H = A_H(n)F. \tag{8}$$

Example 1: The FPRM of a two-variable Boolean function *f*, given by the truthvector $F = [1,0,0,0]^T$, for a polarity vector H = [0,0] is given by:

$$f(x_1, x_2) = ([1 \quad x_1] \otimes [1 \quad x_2])(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix})F = \, = 1 \oplus x_1 \oplus x_2 \oplus x_1 x_2$$

The corresponding FPRM for $H = [1,1]$is given by:

$$f(x_1, x_2) = ([1 \quad \bar{x}_1] \otimes [1 \quad \bar{x}_2])(\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix})F = \bar{x}_1 \bar{x}_2$$

Fast Fourier transform (FFT) is an algorithm for efficient calculation in terms of space and time of the Discrete Fourier transform (DFT). Extension of FFT to FPRM transform is straightforward, thanks to the so-called Good–Thomas factorization for Reed-Muller matrices [6], [7]. This feature is highly exploited in computing RM transform in the proposed implementation of best RM transform on multicore CPU.

3. Multicore CPU Platform

During the past few years, the major chip manufacturers have realized that growing the processor's clock speed is no longer practical because of its physical limitations. Therefore, they are now interesting on increasing their processors performance by integrating multiple processing cores. Particularly, parallel architectures, such as the multicore CPU and manycore GPU, have much attention for high performance computing on consumer level. Thus, many software applications which were developed for processing on single-core processors are now being reimplemented in order to efficiently exploit the multicore CPU hardware resources. To solve scientific problems we need not only fast algorithms but also a combination of good tools and fast computers. The model of the parallel processing that is developed in multicore CPU architectures is based on a large number of processor cores with the ability to directly address into a shared RAM memory. Those architectures provide parallel single instruction multiple datastream (SIMD) computing units, up to several processing cores, huge memory bandwidth and caches, and rather simple branching circuits. In SIMD computing, a large number of threads execute in parallel a single data-parallel function.

There are several application programming interfaces which are available for the development of multicore CPU programs, like MPI, OpenMPI, OpenMP or TBB (Treading Building Blocks). MPI has become a widely used standard, though not necessarily the best language for parallel programming. Among the parallel programming standards, the MPI is very popular because of its rich interface. Also, the implementations available manage to bridge the performance gap between the hardware and the applications [25]. During an MPI application, data are exchanged among the various participating processes. The MPI programming paradigm is: each process may communicate with any other in the application. Process communication using a network is slower than process communication using shared memory [26].

4. Reed-muller Transform On Multicore Cpu Platform

In this section, it is presented both implementations for FPRM transform with sequential code and through MPI framework. An outline of the sequential program for computation of FPRM transform on singlecore CPU platform is given as Algorithm 1.

Algorithm 1

```
1: Set the total number of non-zero coefficients in FPRM spectrum minC to maximum value.

2: Determine the next polarity vector H, of the FPRM form according to the exhaustive rule.

3: Compute the FPRM spectrum of polarity H based on the FFT-like calculation from truth vector of Boolean function.

4: Calculate the total number of non-zero coefficients C in FPRM spectrum.

5: If C < minC then minC = C.

6: Stop if all polarities have been treated. Otherwise go to the step 2.

7: Obtain polarity H having minimal number of non-zero coefficients (minC).
```

The sequential program accepts a Boolean function, represented by its truth vector and obtain best polarity vector as having the influence in optimal size of the resulting FPRM. However, it should be noticed that finding the best polarity can be very CPU time consuming, since the methods for computing the FPRM transform (step 3) are exponential in the number of variables in the function.

An outline of the parallel program for computation of FPRM transform on multicore CPU platform is given as Algorithm 2. This organization of computations allows to have a large number of processes performing the same operations on different data simultaneously which is a good match to the multicore CPU hardware. In considered mappings of sequential program, the input truth function vector is shared between processes and the FPRM spectrums of procesess are stored in the RAM memory. It should be noticed that very CPU time consuming part (computing the FPRM transform in step 3) is split up into cores tasks and each task is assigned one set of polarity vectores. Note that each task allocates only as much storage as needed for its arrays. However, this implementation is tailored for a specific vendor hardware and is thus not suitable for a generic case.

Algorithm 2

1: Set the total number of non-zero coefficients in FPRM spectrum minC to maximum value.

2: Initialize the MPI execution environment using MPI_Init command.

3: Get the number of processes (cores), and the id of this process using MPI_Comm_size and MPI_Comm_rank commands.

4: Allocate memory for message passing of truth vector of Boolean function using MPI_Alloc_mem command.

5: Use the id of this process and number of cores to work out which iterations to perform on determining the next polarity vector H.

6: Determine the next polarity vector H, of the FPRM form according to the exhaustive rule.

7: Compute the FPRM spectrum of polarity H based on the FFT-like calculation from truth vector of Boolean function.

8: Calculate the total number of non-zero coefficients C

9: If C < minC then minC = C.

10: Stop if all polarities have been treated. Otherwise go to the step 6.

11: Determine the minimal minC an find the id of process with minimal minC using MPI_Reduce command.

12: If id of process = id of process with minimal minC then obtain polarity H having minimal number of non-zero coefficients (minC).

13: Terminates MPI execution environment using MPI_Finalize command.

5. Experimental Results

In this section we compared the performance of our multicore CPU accelerated to a single-core CPU implementation for a sample set of random functions. Below we give Table 1 of computation performance using algorithms from previous section. The data in table are sorted in the increasing order of the number of functions variables.

Num. of function variables	computation time [s]	
	CPU	multicore CPU
10	0.001	0.000
11	0.125	0.019
12	0.530	0.081
13	2.294	0.364
14	7.021	1.132
15	38.564	6.536
16	135.346	25.064

Table 1. Computation Times of Best Reed-muller Transform on Single CPU And Multicore CPU Platform

The computations are testing on a PC Pentium IV on 2.66 GHz with 4 GB of RAM. Multicore CPU that is used is an Intel i7 with and its 4 physical cores and hyper-threading yields 8 logical cores on desktop PCs. The MPI environment are developed using the Microsoft HPC Server 2008 [27]. All times in all tables are given in seconds. As it can be seen from Table 1, the MPI implementation of the best RM transform clearly outperforms the referent sequential CPU implementation. The speedup, in terms of number of variables of the function, varies from the factor of 6.5x to 5.5x.

6. Conclusion

This paper proposes an efficient implementation of best fixed polarity Reed-Muller transform computation based on exhaustive search of the all possible polarity vectors on multicore CPU platform. The ultimate goal is to exceed the computation performance for finding the best polarity, since the computation time for the FPRM transforms are exponential in the number of variables in the function. The proposed implementation exploits the parallel mapping of exhaustive search for best FPRM transform performing the same operations on different data simultaneously which is a good match to the multicore CPU hardware.

The experimental results confirm that the application of the proposed implementation on multicore CPU platform leads to significant computational speedups. It is also confirmed that this implementation is especially efficient for functions with large number of variables. From results, it is evident that standard MPI framework can be efficiently used in implementation on parallel computation of best FPRM transform.

Future work will be on extension of the proposed method and the algorithm to various other polynomial transform for Boolean functions.

References

[1] Blahut, R.E. (2010). Fast Algorithms for Signal Processing. Cambridge University Press: Cambridge.

[2] Ahmed, N. & Rao, K.R. (1975). Orthogonal Transforms for Digital Signal Processing. Springer-Verlag.

[3] Chitode, J.S. (2007) Information coding techniques. Technical Publications.

[4] Sweeney, P. (2002). Error Control Coding: From Theory to Practice. Wiley: Chichester.

[5] Thornton, M.A., Drechsler, R. & Miller, D.M. (2001). Spectral Techniques in VLSI CAD. Springer: Berlin.

[6] Karpovsky, M.G., Stankoviæ, R.S. & Astola, J.T. (2008). Spectral Logic and Its Applications for the Design of Digital Devices. Wiley: Chichester.

[7] Sasao, T. & Fujita, M. (1996). Representations of Discrete Functions. Kluwer Academic Publishers: Boston.

[8] Tan, E.C. & Yang, H. (2000) Optimization of fixed-polarity reed- Muller circuits using dual-polarity property. *Circuits Systems and Signal Processing*, 19, 535–548.

[9] Falkowski, B.J. & Chang, C.-H. (2000) Generalised k-variable-mixedpolarity Reed-Muller Expansions for System of Boolean Functions and their Minimization. IEE Proceedings – Circuits, *Devices and Systems*, 147, 201–210.

[10] Falkowski, B.J., Rahardja, S. & Lozano, C.C. (2004) Efficient calculation of fixed polarity Reed-Muller expansions over GF(5) using extended dual polarity property. *Proceedings of the of the 47th Midwest Symposium on Circuits and Systems*, Vol. 2, pp. 221–224.

[11] Jankoviæ, D., Stankoviæ, R.S. & Moraga, C. (2009) Optimization of polynomial expressions by using the extended dual polarity. *IEEE Transactions on Computers*, 58, 1710–1725.

[12] Dueck, G.W., Maslov, D., Butler, J.T., Shmerko, V.P. & Yanushkevich, S.N. (2001) A method to find the best mixed polarity Reed-Muller expression using transeunt triangle. Proceedings of the of the 5th Int. Reed-Muller Workshop. Starkville, MA, USA, pp. 82–92.

[13] Faraj, K. (2005). Combinational Logic Synthesis Based on the Dual Form of Reed-Muller Representation [PhD Thesis]. Edinburgh Napier University.

[14] Faraj, K. & Almaini, A.E.A. (2008) Optimal polarity for dual reed- Muller expressions. Proceedings of the of the 7th WSEAS international conference on Microelectronics. *Journal of Nanoelectronics and Optoelectronics*. Istanbul, Turkey, pp. 45–52.

[15] Wu, W., Wang, P., Zhang, X. & Wang, L. (2008) Search for the best polarity of fixed polarity Reed Muller expression base on QGA. Proceedings of the of 11th *IEEE international conference on Communication Technology*, pp. 343–346.

[16] Butler, J.T., Dueck, G.W., Yanushkevich, S.N. & Shmerko, V.P. (2009) On the use of transeunt triangles to synthesize fixed-polarity Reed-Muller expansions of symmetric functions. Proceedings of the of the Reed-Muller Workshop. Naha, Okinawa, Japan, pp. 119–126.

[17] Drechsler, R., Theobald, M. & Becker, B. (1996) Fast OFDD-based minimization of fixed polarity Reed-Muller expressions. IEEE Transactions on Computers, 45, 1294–1299.

[18] Drechsler, R. & Becker, B. (1997) Sympathy: Fast exact minimization of fixed polarity Reed-Muller expressions for symmetric functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16, 1–50.

[19] Butler, J.T., Dueck, G.W., Shmerko, V.P. & Yanuskevich, S. (2000) Comments on ‘sympathy: fast exact minimization of fixed polarity Reed-Muller expansion for symmetric functions, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (trans. IEEE)*, 19, 1386–1388.

[20] Falkowski, B.J. & Yan, S. (2004) Walsh-Hadamard optimization of fixed polarity Reed-Muller transform. IEICE Electronics Express, 1, 39–45.

[21] Karniadakis, G.E. & Kirby, R.M. (2003). Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation. Cambridge University Press: Cambridge.

[22] Aamodt, T. (2009) Architecting graphics processors for non-graphics compute acceleration. Proceedings of the of the IEEE Pacific Rim Conference on Communications, *Computers and Signal Processing*, pp. 963–968.

[23] Brunelli, C., Airoldi, R. & Nurmi, J. (2010) *Implementation and benchmarking of FFT algorithms on multicore platforms international Symposium on System on Chip* (SoC), Vol. 62, p.59.

[24] Zhou, Y., Zhang, J. & Fan, D. (’09) Software and hardware cooperate for 1-D FFT algorithm optimization on multicore processors Ninth IEEE international conference on Computer and Information Technology, *CIT*, Vol. 1, p. 91, 2009, p. 86.

[25] Hughes, C. & Hughes, T. (2011). Professional Multicore Programming: Design and Implementation for C++ Developers. Wiley - Interscience: Chichester.

[26] Gropp, W., Lusk, E. & Skjellum, A. (1999). Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press: Cambridge, USA.

[27] Microsoft Corporation (2008) Windows HPC server 2008 R2 technical library. Microsoft Website. technet.microsoft.com/sr-latnrs/hpc/. retrieved April 2013.