Random Generation of Bent Functions on Multicore CPU Platform

Miloš Radmanović[1] and Radomir S. Stanković[2]

Abstract – **Bent functions are an important class of Boolean functions since they are used in different kinds of cryptographic applications. Nevertheless, Bent functions are rare and difficult to determine, especially in the case of functions with many variables. During recent years, in order to improve its performance, scientific software has been ported to multicore CPU (Central Processing Unit) and GPU (Graphics Processing Unit) platforms. Thus, this paper proposes a parallel implementation of a well renowned algorithm for generation of Bent functions on multicore CPU platform using the MPI (Message Parsing Interface) framework. The algorithm is based on random discovering of Bent functions on the reduced search space in the Reed-Muller spectral domain. The experimental results show that the random generation of Bent function on multicore CPU platform is quite efficient in terms of the computation time.**

Keywords **– cryptography, Boolean function, Bent function, random generation, multicore CPU platform, MPI.**

I. INTRODUCTION

Bent functions were introduced by Rothaus in 1976 as Boolean functions with maximal nonlinearity [1]. They are important due to the applications in many areas such as combinatorics [2], coding theory [3], cryptography [4], and logic synthesis [5].

It is well known that all Walsh spectral coefficients of Bent functions have the same absolute value $2^{n/2}$, where n is number of variables of the function. They have the maximum possible value of nonlinearity equal to $(2^{n-1} \pm 2^{n/2-1})$, and they only exist for even number of variables. Extensive work on Bent functions has been done and various interesting results of researches have been brought out in respect oftheir generalization, construction, classification, etc.

Constructing of all Bent functions for a given number of variables is possible just for a small number of variables (less than 10) [6]. While general Bent functions are difficult to discover, specific Bent functions can be easily described [7]. It is also well known that from given Bent functions, new Bent functions with the same or greater number of variables can be constructed [8]. All known methods for constructing Bent functions are deterministic, and they do not provide any, for example cryptographic, quality to the generated function.

There are different methods for random discovering of Bent functions, and most of them are based on the Reed-Muller expressions of Boolean functions [9]. A reason for determining Bent functions in the Reed-Muller domain is the efficiency of related algorithms in terms of time.

The algorithm for generation of Bent functions in Reed-Muller domain takes as its input the minimum and maximum number of non-zero coefficients in the Reed-Muller spectrum of orders that the Bent functions are allowed to have. Since the order of Bent functions is less or equal to $n/2$, where n is the number of variables, possible search space for random generation in the Reed-Muller domain is reduced.

Performing the related algorithm is a CPU time consuming task, so we have developed two independent implementations for a comparison, a single-core implementation using C++ and a multicore CPU implementation using MPI framework. Implementations on multicore CPU platforms are recognized as having the potential to considerably speedup or accelerate computing intensive algorithms over their equivalent single CPU core implementations [10]. The proposed multicore MPI implementation exploits possibilities for parallelism that can be found in the algorithm for generation of Bent functions in the Reed-Muller domain.

In the case of the random generation of Bent functions, we investigate this algorithm in the Reed-Muller domain for different minimum and maximum numbers of non-zero coefficients and for random possible orders of Bent functions. In this paper, we studied how the algorithm performances change when the multicore approach is applied. Restrictions in the Reed-Muller domain greatly influence the performance of the random generation of Bent functions. As the Boolean function size increases, the number of calculations extremely increases. For this reason we have experimented with small sizes of Boolean function with strong restrictions in the Reed-Muller domain.

The experimental results obtained on a multicore CPU platform with 8 cores show performance speedups for some benchmarks of maximum 4 times. This results confirm that the application of the proposed implementation using MPI framework leads to significant computational speedups over traditional C++ implementations processed on single CPU.

This paper is organized as follows: Section 2 shortly introduces theoretical background. In Section 3, the algorithm for discovering Bent functions in the Reed-Muller domain is discussed. Section 4 offers some details for multicore CPU implementation of this algorithm using MPI framework. The experimental results are presented and discussed in Section 5. The closing Section 6 summarizes the results of the research reported in this paper.

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

II. THEORETICAL BACKGOUND

Definition 1 For a Boolean function f defined by the truth-vector $F=[f(0),(1),\ldots,f(2^n-1)]^T$, the Walsh spectrum $S_{f,W}=[S_{f,w}(0),S_{f,w}(1),\ldots,S_{f,w}(2^n-1)]^T$ is defined as [11]:

$$S_{f,W}=W(n)F\,, \tag{1}$$

where,

$$W(n)=\bigotimes_{i=1}^{n}W(1) \tag{2}$$

where $\otimes$ denotes the Kronecker product, and

$$W(1)=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}, \tag{3}$$

is the basic Walsh matrix.

The Walsh transform is a self-inverse transform up to the constant 2^{-n} that is used as the normalization factor when defining the Walsh transform and its inverse. It is an important mathematical tool for the analysis of Boolean functions. It can be shown that with $(1,-1)$ encoding of function values, the Walsh coefficients are even integers in the range 2^{-n} to 2^n.

Definition 2 A Boolean function f in $(1,-1)$ encoding is called Bent if all Walsh spectral coefficients S_f have the same absolute value $2^{n/2}$.

Note that a Bent functions can be characterized by the positive polarity Reed-Muller form, leading to a correspondence with the upper bound of the algebraic degree of that form.

A positive polarity Reed-Muller form (PPRM) is an exclusive-OR of AND product terms, where each variable appears uncomplemented. Any Boolean function f can be represented by the PPRM form in matrix notation defined as [11]:

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

where

$$X(n)=\bigotimes_{i=1}^{n}\begin{bmatrix}1 & x_i\end{bmatrix} \tag{5}$$

and

$$R(n)=\bigotimes_{i=1}^{n}\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} \tag{6}$$

where addition and multiplication are modulo 2, $R(n)$ is the positive Reed-Muller transform matrix of order n, and $R(1)$ is the basic positive Reed-Muller transform matrix.

The PPRM spectrum $S_{f,RM}$ is calculated as [10]:

$$S_{f,RM}=R(n)F\,. \tag{7}$$

The elements of $S_{f,RM}$ are coefficients in the PPRM espressions for any Boolean function [11]:

$$f(x)=a_0\oplus\sum_{i=1}^{n}a_ix_i\oplus\sum_{1\le i<j\le n}a_{ij}x_ix_j\oplus\ldots\oplus a_{12\ldots n}x_1x_2\ldots x_n \tag{8}$$

where Σ denotes modulo 2 summation.

The algebraic degree or the order of nonlinearity of a Boolean function [11] f is a maximum number of variables in a product term with non-zero coefficient a_k, where k is a subset of $\{1,2,3,\ldots,n\}$. When k is an empty set, the coefficient is denoted as a_0 and is called the zero order coefficient. Coefficients of order 1 are $a_1,a_2,\ldots a_n$, coefficients of order 2 are $a_{12},a_{13},\ldots a_{(n-1)n}$, coefficient of order n is $a_{12\ldots n}$. The number of all coefficients of order i is $\binom{n}{i}$. The PPRM coefficients are divided into order groupings according to the number of ones in the binary representation of its index in the spectrum.

Algebraic degree of Bent functions is at most $n/2$ for $n\ge 4$ [6]. Thus, the maximal number of PPRM coefficent of Bent functions is: $\sum_{i=0}^{n/2}\binom{n}{i}$.

The number of Bent functions of a given number of variables is not known. Therefore, the number of Bent functions is at most $2^{2^{n-1}+\frac{1}{2}\binom{n}{n/2}}$ which is much less than all Boolean functions 2^{2^n} [6].

For the PPRM transform, we need an inverse transform to get back from the Reed-Muller domain. Since the Reed-Muller transform matrix $R(n)$ is a self-inverse matrix over $GF(2)$, the forward and inverse transform are given by the same matrix [11].

The Walsh and Reed-Muller the transform matrices, expressed in (3), and (6) respectively, can be factorized in different ways yielding different fast transform algorithms, the so-called FFT-like algorithms [11].

Figure 1 shows the elementary butterflies operations (flow-graphs) for the Reed-Muller, and the Walsh basic transform matrices, respectively.

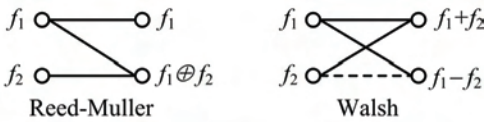

Fig. 1. The elementary butterfly operations for the basic Reed-Muller and Walsh transform matrices

The Cooley-Tukey class of algorithms is based on the Good-Thomas factorization which originates from the Kronecker product structure of the transform matrix [11].

Figure 2 shows the flow graphs of the fast Cooley-Tukey spectral transform algorithm for the computation of the Walsh spectrum of a three-variable logic function f given by the

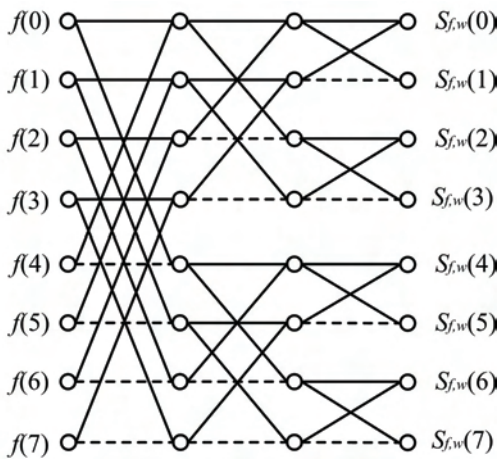

Fig. 2. The flow graphs of the Cooley-Tukey spectral transform algorithm for computing the Walsh spectrum of a tree-variable logic function.

truth- vector $F = [f(0), f(1), \ldots, f(7)]^T$. This algorithm is highly exploited in computing the inverse PPRM transform for discovering Bent functions in the Reed-Muller domain.

Fast spectral transform algorithm reduces computational complexity of a spectral transform from $O(N^2)$ to $O(N\log_2 N)$, it is an extremely effective tool in scientific computing.

III. Random Generation of Bent Functions in Reed-Muller Domain

The algorithm for the generation of Bent functions in the Reed-Muller domain takes as its input the number of function variables and the minimum and maximum number of non-zero Reed-Muller coefficients of any order that is allowed. Since the order of Bent functions is less or equal to $n/2$, the number of non-zero PPRM coefficents is limited and the positions of the coefficients in the PPRM spectrum are restricted. For example, Table 1 gives the limitation of number of the non-zero PPRM coefficents in relation to the total number of coefficients of Bent functions for the number of variables ranging from 8 to 14.

TABLE I
LIMITATION OF THE NUMBER OF THE NON-ZERO PPRM COEFFICIENTS OF BENT FUNCTIONS

Num. of function variables	Limitation / total PPRM coefficients
8	163 / 256
10	638 / 1024
12	2510 / 4096
14	9908 / 16384

These restrictions ensure generation possibility, since they certainly reduce the possible search space for random generation in the Reed-Muller domain. As the Boolean function size increases, the possible search space (see Table 1) also extremely increases.

An outline of the algorithm for the generation of Bent functions in Reed-Muller domain is given as Algorithm 1.

Algorithm 1

1: Set the number of function variables and the minimum and the maximum number of the non-zero Reed-Muller coefficients (the maximum number of coefficeinets should be less than limitations).

2: Random generation of the number of the non-zero coeficients in Reed-Muller spectrum (between minimum and maximum).

3: Random generation of the positions of the non-zero coeficients in Reed-Muller spectrum. Positions of coefficient are related with the order of coefficients and their generation is determinated by the number of ones in the binary representation of the positions in the spectrum (the number of ones is less or equal to n/2).

4: The computation of the truth-vector of a Boolean function is done by using the flow graph of the inverse fast PPRM transform of Reed-Muller spectrum.

5: (1,−1) encoding of a Boolean function.

6: Fast testing if the first Walsh coefficient has the absolute value $2^{n/2}$. Otherwise go to the step 2.

7: Fast testing if the second Walsh coefficient have the absolute value $2^{n/2}$. Otherwise go to the step 2.

8: Fast testing if the last Walsh coefficient have the absolute value $2^{n/2}$. Otherwise go to the step 2.

9: Testing if all values of Walsh spectrum have the same absolute value $2^{n/2}$. The computation of Walsh spectrum is using the flow graph of the fast Walsh transform. Otherwise go to the step 2.

10: Obtain random Bent Boolean function having extreme nonlinearity properties.

IV. Implementation of Random Generation of Bent Functions on Multicore CPU Platform

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. This organization of computations allows to have a large number of processes performing the same operations on different data simultaneously. Process of communication using a network is much slower than the process of communication using the shared memory [12].

The algorithm for random generation of Bent function in the Reed-Muller domain has a large degree of parallelism. Steps 2 to 9 of the Algorithm 1 are computationally independent and according to this, its implementation on multicore CPU platform is convinient. A fundamental step in parallelizing of this algorithm on multicore CPU is the mapping to processor cores of arrays representing Reed-Muller spectrums, that are used for random generation of functions as well as Bent detection. It should be also noticed

that random generation of Bent function can be very CPU time consuming, since the computation of the inverse fast PPRM transform (step 4 of the Algorithm 1) are exponential in the number of variables in the function.

V. Experimental Results

The MPI standard has become a widely used standard for parallel programming framework [13]. For comparison purposes, we developed referent single-core C++ and MPI implementation on multi-core CPU platform of algorithm for random generation of Bent function. Note that, for the largest Boolean functions, computations were not performed on multicore CPU, due to the computation time limitations of 30 minutes.

The computations are performed on an Intel i7 CPU at 3.66 GHz with 12 GBs of RAM. The quad-core CPU that is used is hyper-threading, yielding 8 logical cores with 8 MB of smart cache memory.

Table 1 shows computation performance of algorithm for random generation of Bent function using referent single-core C++ and MPI implementation on multi-core CPU platform.

The presented computational times represent average values for ten executions of implementations for each number of function variables and min. and max. of non-zero Reed-Muller coefficients. From data in Table 1, it can be seen that, on this multi-core CPU platform, for all the computations, MPI implementation of the algorithm reduces computation times when compared to the single-core C++ implementation.

VI. Conclusion

Methods for generating Bent functions are deterministic, and they do not provide any, for example cryptographic, quality to the generated function. The approach for finding a non-deterministic Bent functions is most often based on random discovering Bent function in Reed-Muller domain. It should be noticed that finding the Bent function can be very CPU time consuming, since the search space in Reed-Muller domain are extrimely exponential in the number of variables of the function. However, computing power can be substantially increased through the exploitation of the parallelism on multi-core CPU platform.

In this paper, we investigated parallelization of algorithm for random generation of Bent function on multi-core CPU platform. The parallel MPI implementation of this algorithm is convinient, since the algorihm has a large degree of parallelism. Experimental results confirm that exploiting multi-core CPU platform can help improve the computation performances of this algorithm.

We can conclude that, when processing time is a critical parameter, the algorithm for random generation of Bent functions in Reed-Muller domain should be performed on the multi-core CPU platform. Future work will be on extension of the proposed technique to various other multi-processing platforms.

TABLE I
COMPUTATION PERFORMANCE OF ALGORITHM FOR RANDOM GENERATION OF BENT FUNCTIONS USING REFERENT C++ AND MPI IMPLEMENTATION

Num. of function variables	Min. and max non-zero RM coefficients	Avg. computation time [s]	
		CPU	mCPU
8	1 - 100	1.443	0,288
8	1 - 163	1,792	0,509
10	1 - 100	57,201	32,175
10	1 - 200	32.103	11.108
10	1 - 300	27.883	10.002
10	1 - 400	39.751	10.145
12	1 - 100	> 30 min.	> 30 min.

Acknowledgements

The research reported in this paper is partly supported by the Ministry of Education and Science of the Republic of Serbia, projects ON174026 (2011-2016) and III44006 (2011-2016).

References

[1] O. Rothaus, “On Bent Functions”, Jourbal of Combin. Theory Ser. A, vol. 20, pp. 300–305, 1976.
[2] T. Helleseth and A. Kholosha, “Bent Functions and Their Connections to Combinatorics”, in S. Blackburn, S. Gerke, and M. Wildon, editors, *Surveys in Combinatorics 2013*, pp. 91-126, Cambridge University Press, 2013.
[3] O. Logachev, A. Salnikov, and V Yashchenko, *Boolean Functions in Coding Theory and Cryptography*, American Mathematical Society, 2012.
[4] N. Tokareva, *Bent Functions, Results and Applications to Cryptography*, Academic Press, 2015.
[5] T. Sasao, J. Butler, and M. Thornton, *Progress in Applications of Boolean Functions*, Morgan and Claypool Publishers, 2010.
[6] P. Langevin and G. Leander, “Counting all Bent Functions in Dimension Eight 99270589265934370305785861242880”, Designs, Codes and Cryptography, vol. 59, pp. 193-201, 2011.
[7] A. M. Youssef, and G. Gong, “Hyper-bent Functions”, Advances in Cryptology-EUROCRYPT 2001, Lecture Notes in Computer Science, vol. 2045, Springer, pp. 406–419, 2001.
[8] J. Seberry and X. Zhang, “Constructions of Bent Functions from Two Known Bent Functions”, Australasian Journal of Combinatorics, vol. 9, pp. 21-34, 1994.
[9] N. Y. Yu, and G. Gong, “Constructions of Quadratic Bent Functions in Polynomial Forms”, IEEE Transactions on Information Theory, vol. 52, no. 7, pp. 3291-3299, 2006.
[10] G. E. Karniadakis, R. M. Kirby, *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation*, Cambridge University Press, 2003.
[11] T. Sasao, and M. Fujita, *Representations of Discrete Functions*, Boston: Kluwer Academic Publishers, 1996.
[12] C. Hughes, T. Hughes, *Professional Multicore Programming: Design and Implementation for C++ Developers*, Wiley-Interscience, 2011.
[13] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1999.