©2016 Published in 4th International Symposium on Innovative Technologies in Engineering and Science 3-5 November 2016 (ISITES2016 Alanya/Antalya - Turkey)

Parallelised Algorithm of Isolated Minterm Detection for Logic Function Simplification

*[1]Fatih Başçiftçi and [2]Hakan Akar
*[1]Faculty of Technology, Department of Computer Engineering, Selçuk University, Konya, Turkey
[2]Elmalı Vocational School, Akdeniz University, Antalya, Turkey

Abstract:

This study compares serial and parallel implementation of isolated minterm detection algorithms. Isolated minterms are detected from 10 different functions. Each function has different sized Karnaugh maps (Kmaps). Both serial and parallel implementations detected same isolated minterms but parallel implementation of algorithm runs faster than serial one. We presented "Finding isolated minterms in simplification of logic functions" in our previous work [1]. In this study, we developed the algorithm, parallelised this algorithm and compare computation times on different benchmarks. 10 different sized functions are tested with both algorithms. Results revealed that parallelised algorithms works faster than serial algorithm especially for big sized logic functions.

Key words: Isolated Minterms, Minterms, Isolation Level, Logic Functions, Switching Circuit Simplification, Minimization.

1. Introduction

As digital electronics are represented by logic functions, logic functions are used in many areas such as biology [2], computer hardware [3], cryptography [4], etc. [5]. Logic functions could be single or multiple valued functions. Significant enhance in digital technology results in more complicated logic functions. Thus, simplification of logic functions are very important for saving hardware, run time, memory, energy and decreasing fault tolerance.

Logic function minimization is important [6] as a common tool for many disciplines [7]. Many function simplification algorithms are developed such as Pomper and Armstrong [8], Quine-McClusky [9], Beslich [10], BOOM [11], Espresso [12], etc. As there exists different kind of logic functions, none of the algorithms are superior to others for all logic functions [13].

Logic functions consist of minterms and simplified results consist of Prime Implicants (PIs). Simplified results include less number of Prime Implicants than minterms in logic functions. In addition, costs of PIs are cheaper than minterms. Quality of logic function simplification is simplicity of results. One of the most important factors effecting quality is minterm ordering prior to simplification. Finding the right minterm to begin simplification is very important [14]. Simplification of minterm ordered logic functions is faster and result quality is higher than unordered simplification.

*Corresponding author: Address: Faculty of Technology, Department of Computer Engineering Selçuk University, 42003, Konya TURKEY. E-mail address: basciftci@selcek.edu.tr, Phone: +903322233353

2. Related Works

2.1. Isolated Minterms

Regarding the complexity of algorithms, best case and worst case conditions are affected by data set to be processed. For a more organized data set, algorithms work faster which leads to best case complexity [15]. This rule is same in logic function simplification. If the dates set, minterms in this case, are organized according to isolation level, logic function simplification algorithms work faster.

Relatively centred minterms are easy to cover for simplification. But as there are many options to cover centred minterms, result could be inaccurate. Simplification algorithm may not find the simplest result in this case. But different minterms, relatively farthest ones are difficult to cover. Also simplification results for these minterms are mostly accurate. Figure 1 is an example to clarify the situation. This Kmap includes 4 minterms represented as dots "•" and *A*, *B*, and *C* are the *PIs* covering these minterms.

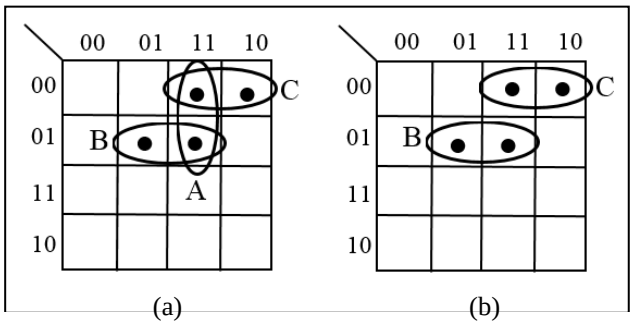

Figure 1. Worst and best SOP statements

Figure 1(a) represents the worst case statement. Simplification process starts from relatively centered minterm "0011", which leads unnecessary implicant *A*. Then minterm 0010 is covered by implicant *C* and minterm 0101 is covered by implicant *B*.

On the other hand, Figure 1(b) represents the best case statement. Simplification process starts from relatively farthest minterm 0010 which produces implicant *C*. Simplification process goes on minterm 0101 which produces implicant *B*. As a result unnecessary implicant *A* is not produced. Thus, simplified result is better and time complexity of simplification algorithm is lesser.

Above example explains how minterm selection effects simplification process. If we separate individual minterms from clustered ones, simplification process will be easier. These individual minterms could be located in the farthest corner of the map or they may have no neighbours be covered together. These minterms are called *isolated minterms*.

2.2. Isolated Minterm Detection

Organizing minterms for simplification effects both result quality and elapsed time. But organizing minterms also spends time. That's why some algorithms don't organize minterms prior to simplification such as Pomper and Armstrong algorithm. In this algorithm, minterm selection is random but a PI selection criterion is number of minterms covered. This algorithm finds all PIs covering target minterm and selects the biggest PI covering most minterms [16].

Besslich Algorithm assigns a *weight number* for each minterm. Clustered minterms has bigger *weight number* and isolated minterms has smaller *weight number*. Simplification process starts from the minterms which has smaller *weight number*. Beslich algorithm calculates *effectiveness factor* for PI selection procedure. *Effectiveness factor* is equal to number of covered minterm divided by cost of each PIs. Most effective PI is selected for covering procedure [10].

Dueck and Miller algorithm calculates the *isolation factor* for each minterm. The *isolation factor* is inversely proportional with sum of neighbour minterms and sum of neighbour minterm directions. Minterm ordering is started from high *isolation factored* minterms to *low isolation factored* minterms. In addition, Dueck and Miller algorithm calculates *Relative Break Count* (*RBC*) for each PIs. *RBC* is the level of simplification for remaining minterms. Thus, PI selection procedure is done according to *RBC level* [14].

BOOM algorithm selects minterms randomly. This algorithm finds all possible PIs and accumulates them in PI pool. Thus, recurring PIs are listed as only one which reduces complexity. PI selection procedure is done according to following rules [11]:
1) Pick the biggest PI covering most of the minterms,
2) Pick the PI covering difficult minterms.
3) Pick the PI that has lowest cost,
4) Pick the PI randomly.

Logic functions vary according to *ON* and *OFF* minterm numbers, function size, default ordering of minterms, etc. Some algorithms are efficient for some functions. The Pomper and Armstrong Algorithm find better results than the random selection. Dueck & Miller and Besslich algorithms find better results than Pomper and Armstrong Algorithms. According to the study by Tirumalai and Butler's; none of the algorithms cannot have a fully superiority against others for all functions [13].

3. Serial Detection Algorithm

Clustered minterms are close to each other while individual *isolated minterms* are away from these clustered minterms. Proposed detection algorithm calculates distance between each minterms and assigns them distance factor. X and Y coordinates of a minterm *A* are represented as A_x and A_y. These coordinates for minterm *B* are B_x and B_y. The distance factors of *A* and *B* minterms for x axis are calculated as below.

$$D_x = A_x - B_x \quad (1)$$

If distance factor (D_x) is smaller than half of Kmap size(S/2), then formula 1 is directly used to calculate D_x. But if D_x is bigger than half of Kmap size(S/2), then these two minterms are closer from the other side of the Kmap. In this case, D_x is calculated by subtraction from map size S. Let two minterms are given on 7x7 Kmap, then maximum Dx between these two minterms could be 3 units. If it's bigger than 3 units, these minterms are closer from other side of the Kmap. This equation is given on Formula 2.

$$D_x = \begin{cases} |\mathrm{A_x} - \mathrm{B_x}| & |\mathrm{A_x} - \mathrm{B_x}| \le \dfrac{S}{2} \\ S - |\mathrm{A_x} - \mathrm{B_x}| & |\mathrm{A_x} - \mathrm{B_x}| > \dfrac{S}{2} \end{cases} \quad (2)$$

The distance factor of y axis(D_y) is also calculated with same formulas stated in 1 and 2. After finding D_x and D_y, absolute distance factor D is calculated by sum of them.

$$D=D_x+D_y \quad (3)$$

Distance factor of each minterm is stored in a 2 dimensional array whose size is equal to S. First and second dimension of array represents x and y axis value of each minterm respectively. *Weightiness factor* (W) of two minterms is equal to Kmap size subtracted by distance factor. This formula is given in 4.

$$W = S - D \quad (4)$$

As seen from Figure 2, four nested loops are used to compare every minterms. First and second for loops represents x and y axis of minterm *A*, while third and fourth for loops represents x and y axis of minterm *B*.

Distance and Weightiness factors are calculated and stored in 2 dimensional arrays. Finally minterms are sorted according to their *W factor*.

```
for each x axis of minterm A do
    for each y axis of minterm A do
        for each x axis of minterm B do
            for each y axis of minterm B do
                Dx = Ax – Bx
                If (Dx > S/2) then
                    Dx = S – Dx
                Dy = Ay – By
                If (Dy > S/2) then
                    Dy = S – Dy
                W = S – D
                WA += Wx + Wy
            end
        end
    end
end
sort minterms by their W
```

Figure 2. Minterm detection nested loops

4. Parallelised Detection Algorithm

4.1. Parallel Computing

Computer technology has advanced significantly in last decade. Nowadays, server computers has 16, 32 cores in their CPUs, just as most of the PCs has multicore CPUs. To exploit the potential of computers, parallel computing algorithms should be utilized. There are many studies [17] about parallel computing [18]. As user interface remain unchanged for parallel computing, end users have no difficulty for working programs. Thus technical knowledge for parallel computing is not necessary for end users [19]. Parallelism is dividing a big task into smaller tasks and assigns them to different core of CPUs then integrate the results coming from each core [20]. Theoretically this approach should speed up a program up to number of core times. But smaller tasks should be independent from each other. These smaller tasks cannot communicate with each other. Also dividing and integration phase needs time. Thus practical results of parallelisation may not speed up as expected.

4.2. Parallel Detection Algorithm

One of the biggest obstacles to parallelise an algorithm is independence. In minterm detection algorithm, all threads are using same array for storing results. Also, parallelisation time makes this process unpractical for small functions.

Isolated minterm detection algorithm is parallelised by loops. This is called data parallelism. Every thread works for different minterms and stores results to the same array. Thus, threads don't need to hold on for each other. Also, parallelised loops could work for CPUs that has any number of cores. On the other hand, task parallelism requires distribution of tasks to number of CPU cores.

All the formulas given in serial detection algorithm is also applied for parallel detection algorithm. In this case, calculations for each minterm are done by different cores of the CPUs.

4.3. Implementation

We used *C#* language to implement serial and parallel detection algorithms. Users can create any size Kmap and then selects the minterms on this Kmap. Find button computes elapsed time for finding isolated minterms 3 times each for serial and parallel algorithms. Average of these three calculations is noted as elapsed time. Figure 3 represents the user interface of the program. This program is tested on a computer capable of i5 processor, 8 Gb. memory and running Windows 8 operating system.

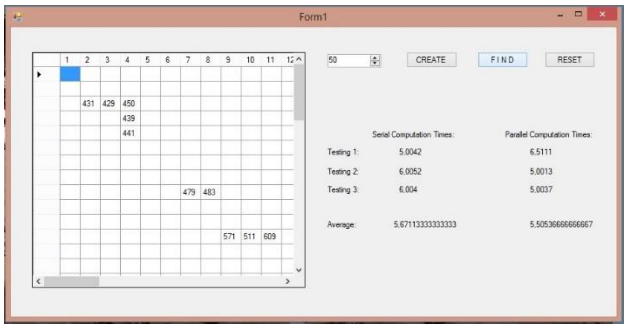

Figure 3. User interface of isolated minterm detection program

5. Experımental Evaluation and Applications

Conducted experimental results are presented in this section. 10 functions having 25 minterms are tested with serial and parallel detection algorithms. Kmap size of these functions varies from 50 to 500 cells. Comparison of results is given in Table 1.

Table 1. Serial and parallel implementation times (msec.)

No	Number of Rows	Serial	Parallel	Time Difference	Percentage
1	50	5.671	5.505	0.166	3.02%
2	100	22.682	19.679	3.003	15.26%
3	150	51.035	45.029	6.006	13.34%
4	200	92.396	78.407	13.989	17.84%
5	250	147.434	126.503	20.931	16.55%
6	300	212.477	180.452	32.025	17.75%
7	350	285.192	238.158	47.034	19.75%
8	400	372.917	324.549	48.368	14.90%
9	450	468.981	396.263	72.718	18.35%
10	500	583.39	494.663	88.727	17.94%

Results revealed that parallelised algorithm runs faster than serial one as expected. Minimum detection time is 5.5 milliseconds for function 1 which has 50 rows Kmap size. Maximum detection time is 583.4 for function 10 which has 500 rows Kmap size. Only function 1 has speed up less than 13%. Speed up of parallelisation for all the other functions are between 13.3% and 19.8%.

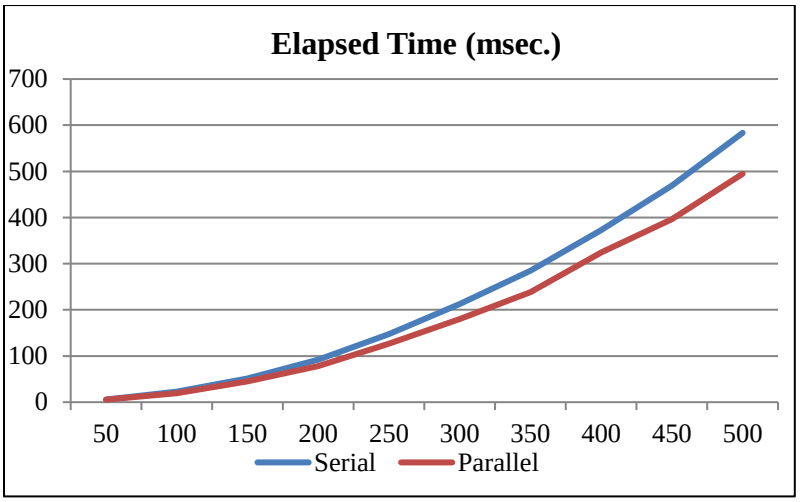

Figure 4. Elapsed time of serial and parallel algorithms

As seen from Figure 4, serial and parallel implementation times of isolated minterm detection algorithm is close to each other for smaller functions. There is almost no difference for a Kmap having 50 rows. On the other hand, there is a significant time saving for bigger functions. Figure 5 demonstrates that time gain of parallelisation is proportional to Kmap size. Time gain is bigger for big functions, whereas time gain is smaller for small functions.

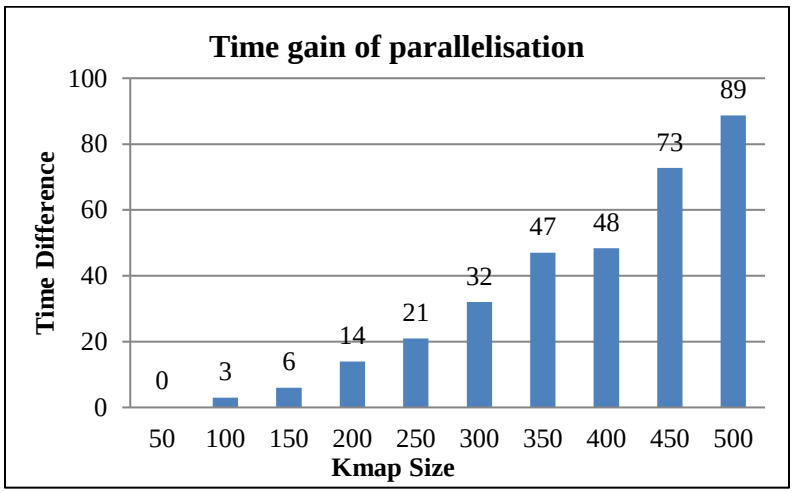

Figure 5. Time gain of parallelisation

As parallelisation procedure needs extra time, it is not convenient for smaller logic functions. Parallelisation is very handy for bigger logic functions as expected.

6. Conclusions and Future Work

The aim of this study is to reveal advantage of parallelised algorithms. We developed our previous work, finding isolated minterms algorithm [1]. In this study, we implemented serial and parallel minterm detection algorithms. 10 different sized functions are tested with both algorithms. Results revealed that parallelised algorithms save more time for big sized Kmaps than smaller ones. Results are similar with other studies [21]. As part of future work, we want to develop and implement parallel algorithms for every phase of logic function simplification.

Acknowledgements

This work has been supported by Scientific and Technological Research Council of Turkey, 1059B141500323 and the Coordinatorship of Selcuk University's Scientific Research Projects.

References

[1] Başçiftçi, F. and Akar, H., 2014, Finding isolated minterms in simplification of logic functions. International Conference on challenges in IT, Engineering and Technology (ICCIET'2014).
[2] Macia, J., Manzoni, R., Conde, N., Urrios, A., Nadal, E., Solé, R., & Posas, F., 2016, Implementation of complex biological logic circuits using spatially distributed multicellular consortia, PLOS Computational Biology.
[3] Hyduke, S. M., Hahanov V.I., Melnikova O.V., Hahanova I.V., 2005, Hardware emulation of large scale boolean equations systems, ISSN:1392-1215, Elektronika ir Elektrotechnika, 3(59).
[4] Datta, K. and Sengupta, I., 2013, Applications of reversible logic in cryptography and coding theory. 26th International Conference on VLSI Design.
[5] Amy, M., Maslov, D., Mosca M., and Roetteler M., 2013, A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits, Transactions on Computer - Aided Design of Integrated Circuits and Systems, 32(6).
[6] Altun M. and Riedel, M. D., 2012, Logic synthesis for switching lattices, Transactions on Computers, 61(11).
[7] Martins, M. G. A., Ribas, R. P., Reis, A. I. (2012): Functional composition: A new paradigm for performing logic synthesis. 13th International Symposium on Quality Electronic Design.
[8] Pomper, G. and Armstrong, J. R., 1981, Representation of multivalued functions using the direct cover method, IEEE Transactions On Computers, Vol. C-30, No. 9.
[9] McCluskey, E. J., 1956, Minimization of boolean functions. Bell System Technical Journal, 35, pp.1417–1444.
[10] Besslich, P. W., 1986, Heuristic minimization of MVL functions: A direct cover approach, IEEE Transactions on Computers, C-35(2).
[11] Bernasconi, A., Ciriani, V., Fišer, P. Trucco, G., 2012, Weighted don't cares. Proc. of 10th International Workshop on Boolean Problems. p. 123-130, ISBN 978-3-86012-438-3.
[12] Brayton, R. K., Hachtel, G. D., McMullen, C. T., Sangiovanni-Vincentelli, A. L. 1984, Logic minimization algorithms for VLSI synthesis, ISBN:0-89838-164-9, Kluwer Academic Publishers.

[13] Tirumalai, P. and Butler J. T., Analysis of minimization algorithms for multiple valued programmable logic arrays, IEEE Transactions on Computers, Vol. 40(2), pp.167-177., 1991.
[14] Dueck, G. W., 1988, Algorithms for the minimization of binary and multiple-valued logic functions, Graduate School of Manitoba.
[15] Oh, J, Choi, C-H, Park, M-K, Kim, BK, Hwang, K, Lee, S-H, et al., 2016, CLUSTOM-CLOUD: In-Memory data grid-based software for clustering 16s rrna sequence data in the cloud environment, PLoS ONE 11(3).
[16] Pomper, G. and Armstrong, J. R., 1981, Representation of multivalued functions using the direct cover method, IEEE Transactions On Computers, Vol. C-30, No. 9.
[17] Zhang, F., Hu, C., Wu, P., Zhang, H., Wong, M. D. F., 2015, Accelerating aerial image simulation using improved CPU/GPU collaborative computing, Computers & Electrical Engineering, 46, pp.176-189, ISSN 0045-7906.
[18] Subramaniam, K. and Balasubramanian, S. (2015): Application of parallel computing in image processing for grading of citrus fruits. Advanced Computing and Communication Systems, International Conference on, Coimbatore, pp. 1-6.
[19] Kumar, L. and Rath, S. K. (2015): Predicting object-oriented software maintainability using hybrid neural network with parallel computing concept. Proc. of the 8th India Software Engineering Conference, pp.100-109.
[20] Ercan, U., Akar, H., Koçer. A., (2013): Paralel programlamada kullanılan temel algoritmalar, Akademik Bilişim'13.
[21] Zhao, Zhendong, et al. (2015): A computationally efficient algorithm for learning topical collocation models. International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, 1.