International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-10, August 2019

Optimal C code Implementation of OWGWA-CSS Algorithm on TMS320C6713 DSK

Rachana Nagal, Pradeep Kumar, Poonam Bansal

Abstract: ***This paper presents the C code optimization of Oppositional Whale Grey Wolf Algorithm with Control Search Space (OWGWA-CSS) used for denoising EEG signal for its real-time implementation on floating point DSP processor TMS320C6713. In general, developers tried to speed up the program all-time which leads to increase in the code size so as increase the complexity as well as difficult readability. To make the code efficient it is important to optimize the C code. For optimizing the C code first, the MATLAB code is converted to C and then implemented on DSP processor. Here, multiple C code optimization techniques for the efficient implementation of OWGWA-CSS algorithm on DSP processor TMS320C6713 is presented. The aim of optimizing C code is to reduce the computational burden and speed up the process. The work done here is a step to minimize Millions Instruction Per Second (MIPS) used to execute the c- code of OWGWA-CSS EEG noise cancellation algorithm. The C code optimization is done by using Code Composer Studio (Integrated Development environment for TMS320C67xx series). CCS is used for optimization, profiling, debugging and implementation. After applying various C code optimization techniques 25% reduction in MIPS has been obtained.***

Index Terms: Code Optimization, profiling, Adaptive Noise Canceller, DSP Processor, Optimization algorithms, EEG Signal.

I. INTRODUCTION

Electroencephalograph (EEG) is a signal recorded from the brain, which is used for the diagnosis of diseases related to the brain [1-2]. EEG signals have very low signal to noise ratio and get contaminated by Event-related potential (ERPs). ERPs can be generated may be because of line noise or any kind of body movement (like the hand, eyeball, etc.) [3] of the patient during recording. It is very hard to believe but EEG signals also got contaminated by its ongoing EEG too [4]. The effectiveness of EEGs is judged by high SNR. Contaminated EEG signal can lead to the wrong diagnosis of the disease, so it becomes very important to remove the noise from the EEG signal. Therefore, since past decades extensive research has been done for developing some techniques for removing artifacts from the noise signal and increase the SNR of the EEG signal for better diagnosis [5]. In general, linear filtering is opted to remove the artifacts from the EEG signal but because of giving low SNR and depending upon the statistical property of EEG signal it was failed. Since then, the concept of adaptive filtering comes in the picture [6-7].

Manuscript published on 30 August 2019.
*****Correspondence Author(s)
Rachana Nagal, Ph. D., degree from AMITY University, Noida, India.
Pradeep Kumar, B.Sc. degree from CCS University, Meerut, India, in 1998, the M.Sc. degree from CCS University, Meerut, India,
Poonam Bansal, received, B.E. (Electrical) from Delhi College of Engineering, Delhi in 1989, M.E. (Computer Science) from Delhi College of Engineering, Delhi

Subsequently, dozens of algorithms have been developed for removing artifacts from EEG signals using adaptive filtering algorithms [8]. The most popular and most efficient algorithm that has been used for adaptive filtering are gradient-based algorithms [9]. Later there were so many optimization techniques like evolutionary-based [10], Swarm-based [11] and nature-inspired algorithms like Whale optimization algorithm [12-13] and Grey Wolf Optimization [14] comes in picture for obtaining higher SNR for EEG signals. To improve the performance of the adaptive noise canceller optimization algorithm, based on Oppositional Based Learning (OBL) [15], hybrid with the Whale Optimization Algorithm (WOA) and Grey wolf optimization (GWO) has been developed. In this case, Whale algorithm is utilized for updating the optimum control parameters of Grey Wolf algorithm. Opposition-Based Learning (OBL) method is breed to the WOA and will be utilized in the solution encoding stage to speed up the Whale Optimization Algorithm's convergence rate. The prime inspiration of hybrid OWGWA-CSS algorithm is to consolidate the benefits of whale optimization in grey wolf algorithms for conquering the static issue in GWO. To balance out the randomness of optimization strategies (meta-heuristic techniques) with their variations especially for the EEG, Control Search Space (CSS) [16] approach is implemented and executed to enhance optimized ANC, especially for the EEG Signal. The task of noise cancellation is carried out with real-time response, which brings more practical approach to the ANC implementation. Once the algorithm OWGWA-CSS is successfully able to remove the noise from the recorded EEG signal, real-time implementation of this algorithm on the DSP processor is being done. As the algorithm is written in MATLAB So first it is important to convert the MATLAB Code to C by using MATLAB Coder. Although MATLAB is considered as one of the best tools for mathematical computation and a graphical representation, the code written on MATLAB takes longer time to compile the programs than C. Conversion from MATLAB code to C code is widely spreading in the field of science as C language is mostly used for system implementation and realization of signal processing. In MATLAB there is a tool named MATLAB coder (MATLAB to C conversion) to convert M file to C so that the converted C file can run independently. This tool produces the MEX function and verifies the generated C code. The code generated by MATLAB coder is portable code for DSP processor. The reason behind converting MATLAB code to C is a) most of the signal processing algorithm is developed in MATLAB but for product development and porting they will be translated or converted in C, b) conversion of M file to C code is necessary because current

DSP compilers are not able to generate the efficient assembly code due to specific architecture. But it is possible to enhance the generate the C code for the target complier.
After converting the MATLAB file to C code, the algorithm is ready to implement on DSP processor. In general, developers tried to speed up the program all-time which leads to an increase in the code size so as increase the complexity as well as difficult readability. Reducing the implementation time is a very challenging task now. Programmer must make the code efficient through some approaches means it is important to do the code optimization. For making C code efficient some optimization techniques have been applied which reduces the time spent to get the output result.
Optimization techniques like loop optimization, function optimization, memory optimization algorithmic reduction, use of inline and leaf function, loop unrolling, efficient use of variables, memory optimization techniques, usage of processor specific intrinsic instructions etc. have been proposed in the literature [17-21].
In this work, C code for Adaptive Noise Canceller (OWGWA-CSS as an adaptive algorithm) is optimized using various optimization techniques for faster execution on floating-point digital signal processor TMS320C6713. Before implementing the OWGWA-CSS algorithm on the DSP processor TMS320C6713 in real-time it is very important to reduce the Millions Instruction Per Second (MIPS) it is taking to complete the process. So various C-code optimization techniques have been employed for faster processing of the algorithm during its real-time implementation.
This paper is organized as follows: the introduction part gives a brief overview of OWGWA-CSS algorithm along with the importance of using various optimization techniques. Section 2 describes the various optimization method applied to optimize C code. Next section gives the results obtained and the last section concludes the work presented.

II. C- CODE OPTIMIZATION TECHNIQUES

There are so many optimization techniques available, but the most important point before using optimization techniques are one should find the portion or module of the program which needs to be optimized. It becomes very important to search out the portion of the Program which is taking most of the time to execute or running very slow by using a lot of memory. Once each portion of that code is identified separately, it can be rewritten using optimization techniques. Now if every small portion of the program is optimized then the whole program by default be optimized. There is one more point, optimization needed to be done on those modules of the program as well that are run the most repeatedly. Here, in this section, many optimization techniques have been discussed.

A. Use of Intrinsic Function

The compiler associated with TMS320C6713 provides special Intrinsics function that directly uses all the instruction used to execute code based on a different generation of TMS320C6XX processors. It is used to optimize c code very quickly. When you call a function, Intrinsics are expressed with an underscore starting with (_) it. For example

Original Code (Addition without Intrinsics)

```
int addwi(int c, int d)
{
int ans;
ans=c+d;
if (((c ^ d) & 0x70000000) == 0)
{
if ((ans ^ c) & 0x70000000)
{
ans = (c < 0) ? 0x70000000 : 0x6fffffff;
}
}
return (ans);
}
```

Now this is a complicated code and this code can be simply replaced by using _addwi(). addition with **Intrinsics** is shown below.

Optimized Code

```
ans= _addwi(c, d);
```

There is a big list of **Intrinsics** which is used to execute the instruction based on C6000 series processor [22].

B. Pipelining

Pipelining of software is a method which is used to schedule the various codes from the loop so that while one instruction is executing rest of the instruction start with the different steps to reach the execution.

Let us understand it by taking an example. Say we have instructions A, B, C, and D. Instruction A has passes through the iteration A1, A2, A3, A4, A5 to reach the execution, Similarly B(B1, B2, B3, B4, B5), C(C1, C2, C3, C4, C5) and E(1,E2,E3,E4,E5). Now while A1, B1, C1, D1 are executing A2, B2, C2, D2 are in queue to execute as shown in the picture given below and so on for the other iteration as well.

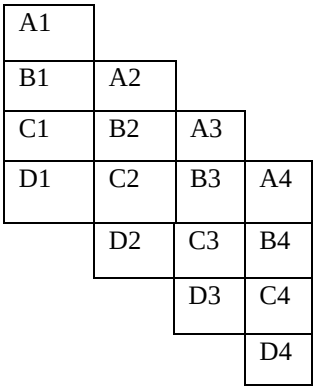

Figure 1: Pipelining process

Loops of any C code plays a very important role and most of the time will often be spent to execute loops. So, it is necessary to find time-critical loops. Loop optimization can be done by so many ways like loop unrolling, loop jamming, loop termination and so on. These loops present in the code leads for the poor performance of the code execution if not optimized properly.

Retrieval Number: J95580881019/19©BEIESP
DOI: 10.35940/ijitee.J9558.0881019
Journal Website: www.ijitee.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Let us now discuss some of the loop optimization method to optimize C code.

- Trip Count
- Loop Jamming
- Loops unrolling
- Early Loop Breaking

Trip Count: In one loop if n number of iterations are there so we call is as trip count. It is important to understand that if you write count up a loop, it is not well justified in terms of code optimization. Writing count down trip counter will take a smaller number of cycles to execute as shown below. Example

Original Code:

```
for (j = 0;   j < P ;   j++ )      /* j= Trip Counter, P= Trip Count */
```

Optimized Code:

```
for (j= P;   j  !=  0;    j-- )        /* Down Counting  Trip Counter   */
```

Loop Jamming: For writing an optimized code there is no need to use many loops if one will work. But sometimes you need to do a lot of work with one loop only, in that case instead of using one loop it is always better to use two different loops so that each can run faster. For Example

Original Code:

```
for (l=0;  l<100;  l++)  {
fun1();    }
for (l=0;  l<100;  l++)  {
 fun2();    }
```

Optimized Code:

```
for (l=0;  l<100;  l++)   {
fun1();
fun2();     }
```

Loop Unrolling: In C code optimization sometimes, the small loops must be unrolled although it comes with the increase in the code size. But still, when thinking in terms of optimization, this method takes a lesser number of instructions to execute a program. For loop unrolling a loop, a counter is needed. For example

Original code:

```
for (k=0; k<3; k++) {
   function1(k);   }
```

Optimized Code:

```
   function1(0);
   function1(1);
   function1(2);
```

Early Loop Breaking: Sometimes we need to search for an array and wants to break the loop at the place when we got it. For example, the below loop is in search of number 78 out of 1000.

Original Code:

```
detect=wrong
for (k=0;  k<1000;  k++) {
   If (list[k] = = 78)
{
   detect= right;  }
}
if(detect) printf (“Found 78”);
```

Now there is another way to terminate the search as soon as the required number has detected.

Optimized Code:

```
detect = wrong;
for (k=0;  k<1000;  k++)  {
If (list[k] = = 78)  {
detect= right;
break;    }
 }
if(detect)  printf (“Found 78”);
```

In the optimized code shown above, as soon as it will find 78, the loop stops and abort rest of 992 iterations.

C. Function Optimization Techniques

Below shown the three function optimization techniques

1. Leaf functions
2. Inline functions
3. function call overhead

Leaf function: There are so many functions in the code which are independent and not calling any other functions. In terms of compiling, leaf functions are compiled very easily. The number of instructions used for storing parameters, registers, and variable in the entry as well as the exit is also very small as compared to the other function. For better C code optimization, it is always advisable to try using as maximum as leaf function.

Inline function: There are some portion of the program which does not need to debug. Inline function can skip debugging of all those functions. For that keyword _inline to be use before the function. Using this keyword compiler skips the debugging of that function and made code faster. Example

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Original Code:

```
int add (int a, int b)
{
Return a+b;
}
```

Optimized Code:

```
_inline int add (int a, int b)
{
Return a+b;
}
```

Apart from making code faster, its disadvantage is making code size larger if the function used in so many places. Solution for this problem is to use inline function wisely and only those places where there is a critical need for this.

Function Call Overhead: Parameters passing in the function using registers is always a matter of concern. Passing more parameters whether it's integer or float or char or structure is not fine. The limit of passing those parameters should be maximum 4. Passing more than 4 parameters will take those to stack. Now if the parameters are in the stack CPU has gone to stack to call them, which waste CPU's so much time. Example

Code 1

```
int owgwa_css1 (float a, float b, float c, float d, float e
, float f)
{
  return a+ b +c+ d+ e+ f;
}
```

Code 2

```
int owgwa2_css (float a, float b, float c, float d)
{
  return a+ b +c+ d;
}
```

As shown in the above table code1 has to send 5th and 6th parameters to the stack, which makes CPU to excess more instruction to fetch them.

D. Dead Code Elimination

Dead code elimination is another very important optimization technique to make code faster. There is a big portion of the program which is not going to be executed for any kind of conditions, For example, if statement. There is a big possibility that the code written inside 'if' statement is never going to be true, which means that the line of code is never going to execute. So, if the compiler can eliminate this dead code it will save memory space. Example

Original Code:

```
#include <stdio.h>
int owgwa_css main ()  {
   int y;
if (y < 0  &&  y >0 )
{
     y=45;
pritnf ("It is found");
}
```

E. Pointers

Pointers, as we know, is used to access the information given in structure. Example

Original Code:

```
typedef  struct  {
int n, m, k ;
}
pt3;
typedef struct {
pt3 *po, *dir;
}
Object;
void gwo (object *p)
{
  p-> po-> n = 0;
  p-> po-> m = 0;
  p-> po-> k = 0;
}
```

Optimized Code:

```
void gwo (object *p)
{
    pt *po = p-> po;
    po-> n = 0;
    po-> m = 0;
    po-> k = 0;
}
```

Now let us analyse the original code and optimized code. IN original code it must have to reload p-> po for every time. The solution is given in optimized code by caching p-> po as a local variable.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

F. Variable:

Variables play a very important role in code optimization. While writing a code we frequently use data type **int**. Now as you know very well in your algorithm that the value you are putting inside variable **int** is never going to be negative so it is always better to use **unsigned int** instead of using **int**. Handling **unsigned int** will save some instruction cycles than **int**.

Global Variables: Storing the global variable always required more storage because the compiler is not able to cache it. Moreover, when global variables are using inside the loops, it becomes more critical. So, it is always advisable to use a global variable as a local variable so other consecutive functions cannot use them. Example

Original Code:

```
int weightu (void);
int feedbcker(void);
int error;
void weightu1(void)
{
    error+=weightu();
    error+=feedcker();
}
```

Optimized Code:

```
void weightu2(void)
{
int errorl= error;
errorl+ = weightu();
errorl+ = feedbcker();
error=errorl;
}
```

Analysing the above example weightu2 will take less time to execute than weightu1 and weightu2 is using errorl which is a local variable and needs only single instruction to execute.

III. RESULT AND DISCUSSION

In the last section, so many C code optimization techniques have been discussed those have implemented to optimized C code written for Oppositional Whale Grey Wolf Algorithm with Control Search Space (OWGWA-CSS). In this work, the code for OWGWA-CSS is implemented in MATLAB first and then converted to C using MATLAB coder. Then the code composer studio compiler has been used to make it ready for porting the code to TMS320C6713. Before the implementation of Adaptive Noise Canceller, C code optimization techniques have been applied. The various C code optimization methods have been applied after finding the exact place of optimization by using the profiling tool of code composer studio. The performance of the OWGWA-CSS algorithm is analysed by obtaining the number of cycles needed to execute the OWGWA-CSS algorithm code before and after optimization. Table 1 shows the results after obtaining the number of cycles to execute the instruction before and after optimization. There are two term used, **Cycle Total: Inclusive:** This mean, the amount of cycle shown in the Table 1 includes the cycles needed to called subroutine as well. **Cycle Total: Exclusive:** This mean the amount of cycle shown in the Table 1 excludes the cycle used to call subroutine. Analysing number of cycles given in Table 1 for C code OWGWA-CSS algorithm; it can be seen clearly that OWGWA-CSS non optimized code taking 1026589 MIPS which is reduced by 25.13 % with optimized code. If talk for MIPS excluding the subroutine the non-optimized code taking 84592 cycle while the optimized code is taking 63598 MIPS. The reduction achieved here is 24.81.%.

TABLE I: CYCLE COUNT OF OWGWA-CSS ALGORITHM BEFORE AND AFTER OPTIMIZATION

OWGWA_CSS Algorithm	No of Cycle Without Optimization		No of Cycle with Optimization	
Main Program name	Cycle Total: Inclusive	Cycle Total: Exclusive	Cycle Total: Inclusive	Cycle Total: Exclusive
void owgwa()	1026589	84592	768547	63598

IV. CONCLUSION

This paper presents the various C code optimization techniques for implementing Oppositional Whale Grey Wolf Optimization algorithm with Control Search Space (OWGWA-CSS) on DSP Processor TMS320C6713. To find out where to optimize profiling tool of Code Composer Studio has been used. After applying various C code optimization methods, the results have been analysed using two parameters; number of cycle inclusive and number of cycle exclusive. The first parameter number of cycles inclusive includes the MIPS used for executing subroutine as well, rather the second parameter number of cycles exclusive excludes the MIPS used for executing subroutine. After optimization both the parameters obtained around 25% reduction in MIPS (number of cycles needed to execute the code). Because of the reduction in MIPS, the optimized code will take lesser time to execute and efficient for its implementation in real time.

REFERENCES

1. Karthik, G. V. S., Fathima, S. Y., Rahman, M. Z. U., Ahamed, S. R., & Lay-Ekuakille, A., “Efficient Signal Conditioning Techniques for Brain Activity,” in Remote Health Monitoring Network. IEEE Sensors Journal, 13(9), 3276–3283, 2013.
2. Janos Szalal, Piroska Haller, Zita Marthi , “Adaptive Filtering of EEG Signals”, in proc. Interdisciplinarity in Engineering International Conference, 406-411, 2012.
3. Mannan, M. M. N., Kamran, M. A., & Jeong, M. Y. “Identification and Removal of Physiological Artifacts from Electroencephalogram Signals: A Review” IEEE Access, 6, 30630–30652, 2018.

4. Machado, S., Araújo, F., Paes, F., Velasques, B., Cunha, M., Budde, H. Budde, F. Basile, R. Anghinah, O. Arias-Carrio´ n, M. Cagy, R. Piedade, T.A. Graaf, A.T. Sack, and P. Ribeiro, "EEG-based Brain-Computer Interfaces: An Overview of Basic Concepts and Clinical Applications in Neurorehabilitation". In Reviews in the Neurosciences, 21(6), 451-468,2010.
5. B. Paulchamy , "Efficient Removal of Artifacts from EEG signal Using Enhanced Hybrid Learning Method," in Studies on Ethno-Medicine, 11:4, 359-365, 2017.
6. B. Widrow, J.R. Glover, J.M. McCool, J. Kaunitz, C.S. Williams, R. H. Hearn, J. R. Zeidler, Eugene Dong Jr., R. C. Goodlin , "Adaptive noise cancellation: Principles and applications," in Proc. IEEE 63 (12), 639–952, 1975.
7. Janos Szalal, Piroska Haller, Zita Marthi, "Adaptive Filtering of EEG Signals", in proc. Interdisciplinarity in Engineering International Conference, 406-411, 2012.
8. B. Widorw and M. Kamenetsky , "Statistical efficiency of adaptive algorithms," in IEEE Tran. On Neural Network, Neural Networks, 16(5-6), 735-44,2013.
9. J. M. Górriz, Javier Ramírez, S. Cruces-Alvarez, Carlos G. Puntonet, Elmar W. Lang, and Deniz Erdogmus, "A Novel LMS Algorithm Applied to Adaptive Noise Cancellation," in IEEE Signal Processing Letters, 16(1), 34–37, 2009.
10. A.Zhou, B.Y.Qu,H.Li, S.Z.Zhao, P.N.Suganthan, Q.Zhang, "Multi objective evolutionary algorithms: a survey of the state of the art," in Swarm and Evolutionary Computation. 1 (1) 32–49, 2011.
11. Mohd Nadhir, Ab Wahab, M. N., Nefti-Meziani, S., & Atyabi, A., "A Comprehensive Review of Swarm Optimization Algorithms," in PLOS ONE, 10(5), 2015.
12. S. Mirjalili & A. Lewis, "The whale optimization algorithm," in Advances in Engineering Software, 95, 51-67, 2016.
13. Gharehchopogh, F. S., & Gholizadeh, H., "A comprehensive survey: Whale Optimization Algorithm and its applications," in Swarm and Evolutionary Computation. doi:10.1016/j.swevo.2019.03.004, 2019.
14. Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis, "Grey Wolf Optimizer," in Advances in Engineering Software 69, 46-61,2014.
15. H. R. Tizhoosh, "Opposition-based reinforcement learning", in Journal of Advanced Computational Intelligence and Intelligent Informatics, 10(4), 578-585, 2006.
16. M.K. Ahirwal ; Anil Kumar ; G.K. Singh, "EEG/ERP Adaptive Noise Canceller Design with Controlled Search Space (CSS) Approach in Cuckoo and Other Optimization Algorithms, " IEEE/ACM Transactions on Computational Biology and Bioinformatics, 10(6), 1491–1504,2013.
17. Mark B. Kraeling, "Optimization of C code in a real time environment," WESCON, pp:574 – 580,1996.
18. Pratibha Singh, Smriti Sonker, Prabhat Verma, "Optimization Techniques in C – A Survey," in International Journal of Computer Applications Volume 115 – No. 20, April 2015.
19. Kin H. Yu and Yu Hen Hu, "Optimized Code Generation for Programmable Digital Signal Processors," in IEEE International Conference on acoustic speech and signal processing, 461-464, 1993.
20. Rachana Nagal, Pradeep Kumar, Poonam Bansal, "Profiling and Optimization of Variable Step Size Algorithm used for Adaptive Noise Cancellation," on 2nd International Conference on "Computing for Sustainable Global Development", Proceedings of the 9th IEEE Conference INDIACom, pp. 1848 - 1852, March 2015.
21. Qingfeng Zhuge, Zili Shao, Edwin H. -M. Sha, "Optimal Code Size Reduction for Software-Pipelined Loops on DSP Applications," in Proceedings of the International Conference on Parallel Processing, pp. 1-8, 2001.
22. Code Composer Studio IDE Getting Started Guide: User's Guide (spru509f)", Texas Instruments, 2005

AUTHOR'S PROFILE

Rachana Nagal received the B.Sc., degree in electronics from Rani Durgavati University Jabalpur, Madhya Pradesh, India in 2002, the M.Sc. degree from the School of Electronics, Devi Ahilya University, Indore, Madhya Pradesh, India, in 2004, and the MTech., degree from the Panjab University, Chandigarh, India, in 2007. Currently, she is pursuing her Ph. D., degree from AMITY University, Noida, India. Her Research interest is adaptive signal processing. Mail id: rachana.nagal@gmail.com

Pradeep Kumar received the B.Sc. degree from CCS University, Meerut, India, in 1998, the M.Sc. degree from CCS University, Meerut, India, in 2000, and the Ph. D. degree from Garwal University Uttaranchal, India, in 2006. His area of interest includes analog and digital filter designing, Noise cancellation, VLSI Design. Mail id: pkumar4@amity.edu

Poonam Bansal, received, B.E. (Electrical) from Delhi College of Engineering, Delhi in 1989, M.E. (Computer Science) from Delhi College of Engineering, Delhi in 2001, Ph.D. from School of Engineering and Technology, Guru Govind Singh Indraprastha University, New Delhi. Her area of interest includes speech enhancement and noise cancellation from the speech signal, speech signal processing. Mail id: pbansal89@yahoo.co.in