International Journal of Combinatorial Optimization Problems and Informatics, 11(3), Sep-Dec 2020, 130-135. ISSN: 2007-1558.

www.editada.org

Time optimization for polynomial series using a long For in C

Gustavo Medina Ángel[1, 2], *Marco Cruz Chávez*[1], *Gennadiy Burlak*[1]

[1] Centro de Investigación en Ingeniería y Ciencias Aplicadas, Universidad Autónoma del estado de Morelos, México.
[2] Facultad de Contaduría, Administración e Informática, Universidad Autónoma del estado de Morelos, México.
gustavo.isc@hotmail.com, mcruz@uaem.mx, gburlak@uaem.mx

Abstract. We use a For loop in the programming language C to optimize lines of code and time in the resolution of polynomial series, and we make a comparison between the conventional For and a long For to optimize the execution time, the results show a saving of lines of code and great saving of time for the resolution of polynomials when there is a greater number of terms in the function.

Keywords: For, Long, Code, Series, Polynomials.

Article Info
Received June 11, 2019
Accepted Dec 11, 2019

1 Introduction

The C language is one of the most popular programming languages in the field of research and analysis of algorithms [1] [2] [3] [4] due to the ease of programming and the manipulation of its sentences and repetition structures.

Due to this reason, we study the behaviour of a For cycle when we incorporate the initialization of variables and increments within the structure of the cycle, unlike conventional methods in which the cycle is only composed of the initialization of the main variable of the loop, a condition that ends the loop cycle and an increment that will control the jumps of the cycle [5].

For the long For cycle, we incorporate within its structure the variable initializations and all increments within the same body that can be read in a single line of code.

For our case study, we optimize the following polynomial equation series (**1**) [6] [7] [8] [9] [10] [11] [12] [13] [14] [15], where we program a series in a conventional way and another applying a long For.

$$f(x,y,z,n) = \frac{y^{1+x}}{x-(1-z)} - \frac{y^{3+x}}{x^4+(6!-z^5)} + \frac{y^{5+x}}{x^7-(11!-z^9)} - \frac{y^{7+x}}{x^{10}+(16!-z^{13})} + \frac{y^{9+x}}{x^{13}-(21!-z^{17})} \ldots \quad (1)$$

Where *x*, *y* and *z* are variables for the terms of the polynomial series and n is the number of terms that the series will contain, we can observe that the function is composed of fractional terms with an exponent in the variable y which increases by two units in each term grouped in the series more the value of the variable x, the denominator is composed by the variable *x* +/- a factorial with increments of 5 units minus the *z* value as an exponent with increments of 4 units for each grouped term. We can observe that for each term that forming this series, the next value will be subtracted or added according to the term of the sequence in which it is found.

2 Conventional Programming

For the conventional programming, the For repetition structure was used and within its block, the instructions necessary to calculate the polynomial series of equation (**1**) were introduced [16] [17]. Below is the code that was used for conventional programming without reducing.

```
1.    #include<stdio.h>
2.    #include<Math.h>
3.    #include <time.h>
4.    int main(){
5.    clock_t start = clock();
6.    double
x,y,z,n,incx,incy,incz,numfac,ry,rx,rz,vg,a,resfac,signo,signo2,resultado,resultado2;
7.    printf("Enter value for  x:  ");
8.    scanf("%lf",&x);
9.    printf("Enter value for  y:  ");
10.   scanf("%lf",&y);
11.   printf("Enter value for  z:  ");
12.   scanf("%lf",&z);
13.   printf("Enter value for  n:  ");
14.   scanf("%lf",&n);
15.   incy=1;
16.   incz=1;
17.   numfac=1;
18.   incx=1;
19.   signo=1;
20.   signo2=-1;
21.   resultado=0;
22.   resfac=1;
23.   resultado2=0;
24.   for(vg=1; vg<=n; vg++){
25.   ry=pow(y,(incy+x));
26.   incy=incy+2;
27.   rx=pow(x,incx);
28.   resfac=1;
29.   for(a=1; a<=numfac; a++)
30.   resfac=resfac*a;
31.   rz=pow(z,incz);
32.   resultado=(ry/(rx+(signo2*(resfac-rz))))*signo;
33.   incz=incz+4;
34.   incx=incx+3;
35.   numfac=numfac+5;
36.   signo=signo*-1;
37.   signo2=signo2*-1;
38.   resultado2=resultado2+resultado;}
39.   printf("R=%e\n",resultado2);
40.   printf("Calculation time: %f", (((double)clock() - start)) / CLOCKS_PER_SEC);}
```

Lines of code 1 to 3 include the libraries corresponding to the inputs and outputs, the mathematical library and the library to obtain the time. On line 6 the variables x, y, z and n corresponding to equation (**1**) are declared, the variables incx, incy, incz are the increments of the exponents for each term that are grouped in the function. Variables rx, ry and rz save the result of the corresponding exponents for each term x, y and z raised to its exponent. Variables a and vg are variables that belong to cycle. Variable resfac saves the result obtained from the factorial of the denominator for each term. Variables signo and signo2 will control the change in the +/- sign for the polynomial series in each term, the variable result saves the value of the result of the series, while result2 variable saves the value of the current result plus the value of the result of the previous term of the accumulated series.

In the lines of code 7 to 14, input data corresponding to the values for x, y, z and n are obtained. In lines 15 to 23, the values of all the variables are initialized.

In line 24 we can observe the first For cycle that ends in line 38 that will be executed n times depending on how many terms are calculated. On lines 25 and 27, values of the exponents for resx and resy are calculated, and an increase of 2 is made for each value of the exponent of y on line 27.

On line 28 we assign the value of resfact =1 to clean old values and prepare the calculation of a new factorial. On line 29 an internal for begins, on line 30 result of the factorial of the denominator of the terms of the series is calculated, on line 31, result of the exponent is calculated for z value and is stored in rz variable , on line 32 result of the current term of the series that is saved in resultado variable is obtained, on lines 33 and 34 increments of 3 and 4 times are made for following terms of the exponents z and x correspondingly, while on line 35 value for the factorial of following term is increased. Lines 36 and 37 exchange the sign between one and another term grouped in the series. In line 38 the result of the last of the series is added with the summation value of the terms previously calculated. The final result of the series is printed on line 39. Finally, on line 40 the elapsed time it took to calculate the polynomial series is recorded and printed.

3 Reduced Programming implementing a modified For loop.

We implement a second form of programming by reducing code and integrating all instructions within the internal body of the for cycle (Instructions to execute) [18]**,** within cycle the necessary variables are initialized, and the instructions are execute that were executed before between { } of the For cycle, the code for this program is shown below.

```
1. #include<stdio.h>
2. #include<Math.h>
3. #include <time.h>
4. int main(){
5. clock_t start = clock();
6. double
   x,y,z,n,incx,incy,incz,numfac,ry,rx,rz,vg,a,resfac,signo,signo2,resultado,resultado2
   ;
7. printf("Enter value for  x:  ");
8. scanf("%lf",&x);
9. printf("Enter value for  y:  ");
10.      scanf("%lf",&y);
11.      printf("Enter value for  z:  ");
12.      scanf("%lf",&z);
13.      printf("Enter value for  n:  ");
14.      scanf("%lf",&n);
15.      for(incy=1,incz=1,numfac=1,incx=1,signo=1,signo2=-
   1,resultado=0,resfac=1,resultado2,vg=1;
   ry=pow(y,(incy+x)),incy=incy+2,rx=pow(x,incx),resfac=1,rz=pow(z,incz),incz=incz+4,in
   cx=incx+3,resultado2=resultado2+resultado,vg<=n;numfac=numfac+5,signo=signo*-
   1,signo2=signo2*-1,vg++)
16.      for(a=1; resfac=resfac*a,a<=numfac; a++)
17.      resultado=((ry/(rx+(signo2*(resfac-rz))))*signo);
18.      printf("R=%e\n",resultado2);
19. printf("Calculation time: %f", (((double)clock() - start)) / CLOCKS_PER_SEC);}
```

Second code we can observe a considerable decrease of source code, the initializations of the variables and operations were implemented within the internal body of the For cycle and the corresponding increases were also introduced forming a long For. Elements that are introduced into the body of the For cycle must be separated by commas (,). For both codes method

(((double)clock() - start)) / CLOCKS_PER_SEC was used from the library **time.h** to measure time and establish the difference between both cases [19].

4 Results

Below are some examples that show the results and time between conventional and the reduced form using a long For cycle.

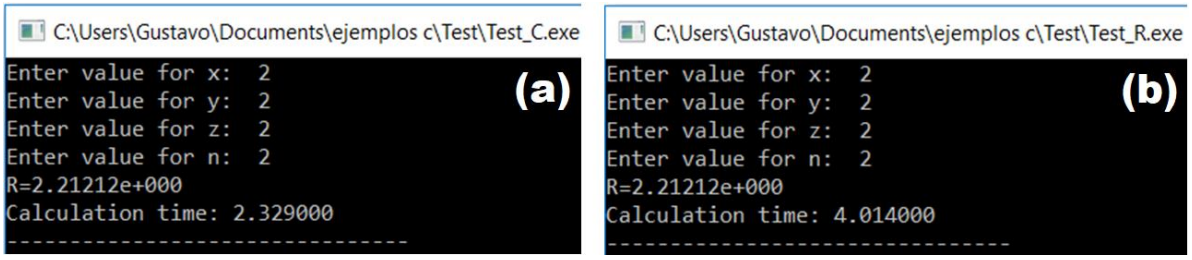

Fig. 1. Test 1, with values for x=y=z=2 and number of times
n=2 for the conventional (a) and reduced (b) programming respectively.

Fig. 1 (a) and Fig. 1 (b) show results for the first case with the input parameters (x=y=z=n=2), we can observe the same result (R) for the polynomial series but with different calculation time between the reduced form and the conventional form. The difference is 1.685 milliseconds.

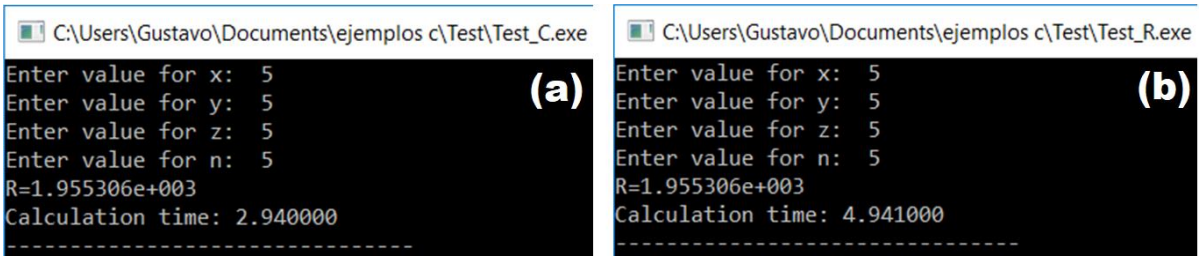

Fig. 2. Test 2, same as in Fig.1 but with x=y=z=n=5.

In Fig. 2 (a) and Fig. 2 (b) we can observe the same case as **Fig. 1**, but with a different number of terms n=5 and with the variables x=y=z= 5, with a time difference of 2.001 milliseconds while for values of Fig. 3(a) and Fig. 3(b) we have a value of n = 30, x = y = z = 30 and a time difference of 4.642 milliseconds. We can observe that for tests 1 and 2, time is shorter for the case of conventional programming.

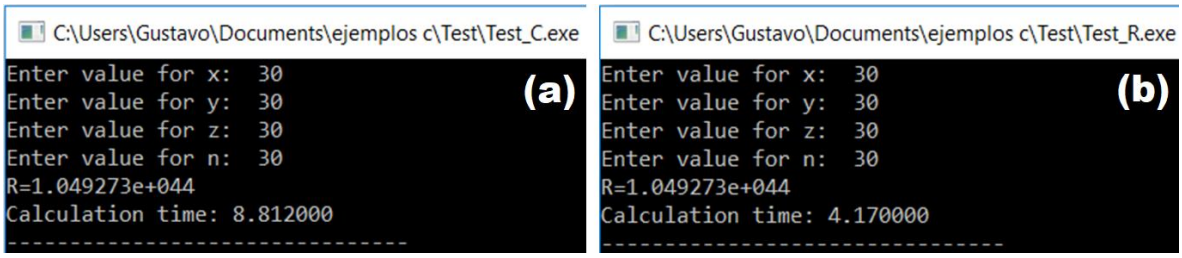

Fig. 3. Test 3, same as in Fig. 1 but with x=y=z=n=30.

In Fig. 4(a) and Fig. 4(b) we can observe different values in input parameters x = 15, y = 27, z = 41 and n = 50 obtaining a calculation time of 6.035 milliseconds between both cases.

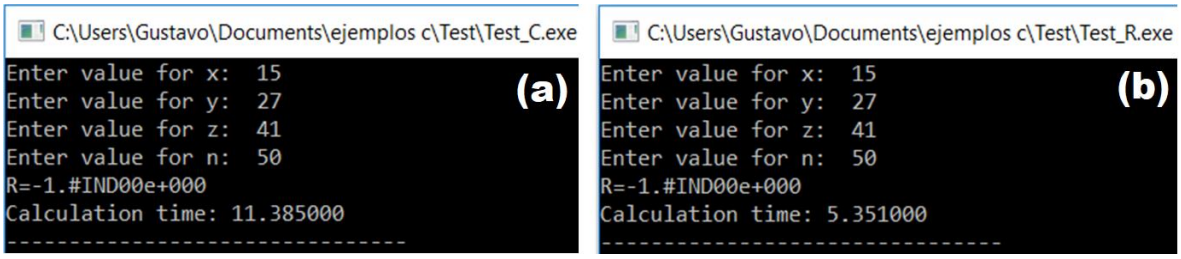

Fig. 4. Test 4, the same as in Fig. 1 but with values for x=15, y=27, z=41 and number of times n=50.

We can observe that for the last two tests (Test 3 and test 4) time is many minors in the case of reduced programming.
Table 1 shows the processing times between conventional and reduced program, as well as the difference between them for each of the cases of figures 1-4.

Table 1 Times for tests in Figures 1-4 and the time difference between them.

Parameters	Time for Conventional Programming	Time for Reduced Programming	Difference
x=y=z=n=2	2.329	4.014	1.685
x=y=z=n=5	2.940	4.941	2.001
x=y=z=n=30	8.812	4.170	4.642
x=15, y=27, z=41, n=50	11.385	5.351	6.035

We can observe from table 1 that the higher the value assigned in the parameters, the greater the calculation time.
We perform several tests by modifying the value of n=1 to n=100 with the input variables x=y=z=2. We can observe the results of these tests in the graph of Fig. 5.

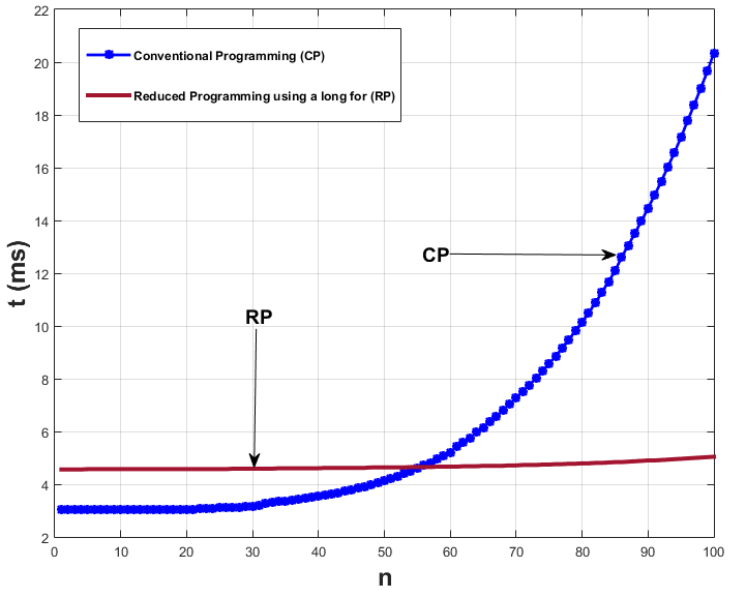

Fig. 5. The behaviour of conventional programming vs reduced programming using a long For, where *n* is a number of polynomials in the series and *t* is the time in milliseconds (ms).

In Figure 5, we can observe a considerable increase in time for the conventional method (CP) after the number of terms in polynomial exceeds the value of $n = 30$, while for the reduced form (RP) using a long For, the time remains almost constant as a function of time.

Tests for numerical calculations were performed on a computer with Acer Intel (R) Atom (TM) processor 32-bit N2800 CPU at 1.86 GHz with 2 GB of RAM. Results may vary depending on the characteristics of each computer.

5 Conclusions

According to the results obtained, we conclude that reduced programming using a long For is more efficient in terms of time and source code, since in a conventional way we program 40 instructions, while in the reduced form we program 19.

References

1. Etter, D. M. (2012). *Engineering Problem Solving with C*. Prentice Hall Press.
2. Luis, J., Fernandez, C., & Ignacio, Z. (2005). Programación en C: Metodología, algoritmos y estructura de datos. *Me Graw Hill. Ed*, *2*.
3. Aguilar, L. J., Azuela, M. F., & Baena, L. R. (1988). *Fundamentos de programación*. McGraw Hill.
4. Battistutti, O. C. (2006). *Fundamentos de Programacion Piensa en C*. Pearson educacion.
5. Deitel, H. M., & Deitel, P. J. (1995). *Cómo programar en C/C+*. Pearson Educación.
6. Anjos, M. F., & Lasserre, J. B. (Eds.). (2011). *Handbook on semidefinite, conic and polynomial optimization* (Vol. 166). Springer Science & Business Media.
7. Laurent, M. (2009). Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry* (pp. 157-270). Springer, New York, NY.
8. Renner, P., & Schmedders, K. (2015). A polynomial optimization approach to principal–agent problems. *Econometrica*, *83*(2), 729-769.
9. Nataraj, P. S., & Arounassalame, M. (2007). A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *International journal of automation and computing*, *4*(4), 342-352.
10. Peña, J., Vera, J. C., & Zuluaga, L. F. (2015). Completely positive reformulations for polynomial optimization. *Mathematical Programming*, *151*(2), 405-431.
11. Li, H. L., & Chang, C. T. (1998). An approximate approach of global optimization for polynomial programming problems. *European Journal of Operational Research*, *107*(3), 625-632.
12. Takano, Y., & Sotirov, R. (2012). A polynomial optimization approach to constant rebalanced portfolio selection. *Computational Optimization and Applications*, *52*(3), 645-666.
13. Sherali, H. D. (1998). Global optimization of nonconvex polynomial programming problems having rational exponents. *Journal of Global optimization*, *12*(3), 267-283.
14. Bleylevens, I., Peeters, R., & Hanzon, B. (2007). Efficiency improvement in an nD systems approach to polynomial optimization. *Journal of Symbolic Computation*, *42*(1-2), 30-53.
15. Schildt, H., C.: Manual de Referencia. Cuarta Edición. McGraw-Hill, Madrid (2001)
16. Cairo, B. (2005). Metodología de la Programación: Algoritmos, diagramas de flujo y programas. 3ra edición. Alfaomega.
17. Kernighan, B. W., & Ritchie, D. M. (1991). *El lenguaje de programación C*. Pearson Educación.
18. Plauger, P. J. (1991). The standard C library. Prentice-Hall PTR, New Jersey.