Proceedings of the Fourth International Conference on Inventive Systems and Control (ICISC 2020)
IEEE Xplore Part Number: CFP20J06-ART; ISBN: 978-1-7281-2813-9

Enhancing the Scope for Automated Code Generation and Parallelism by Optimizing Loops through Loop Unrolling

Anil Kumar S
EE II N, Examinations Wing
University of Kerala
Thiruvananthapuram, Kerala, India
aks.kerala.india@gmail.com

Abstract— **Solving a problem can have multiple methods, each having its own merits and demerits. The ultimate level of complexity of solution models is highly subjective in nature because a method easy for one person may be much difficult for others. Moreover, methods friendly for humans need not be suitable for automated systems like computers. Similar is the case among various automated systems too. All depend on the nature of the problem and the solver, availability of resources, optimization requirements and the like. For example, loops in source code reduce the complexity while programming but give additional overhead during execution. So, optimization of programs by reducing the number of loops can make considerable improvement in performance at runtime. There are several techniques for optimizing the loops. This study is based on the popular loop optimization technique known as Loop unrolling which is having own advantages and disadvantages, but able to open up the additional scope for enhanced parallelism and automated code generation.**

Keywords— ***loop optimization; loop unrolling; optimizing compiler; process models; automated code generation; software parallelism***

I. INTRODUCTION

Compilers use several optimization techniques to achieve better performance and better parallelism[2]. Optimizing compiler, the compiler which uses optimization techniques[5], are designated to maximize or minimize the significant attributes related to executing a computer program. These include attributes such as execution time, speed, memory requirements, communication overhead, ease of debugging, ease of maintenance and even power conception. The optimizations are done not only by the compiler but also by the programmers themselves for achieving specific optimization requirements. Optimizations involving high-level synthesis which are too complex for a human to perform can be made possible through specially designed software optimization systems [11] [12].

The scope of optimization techniques can affect any portion of the program ranging from a single statement to the entire program. It can be local, global or interestingly within the loop(s).

The techniques for optimization of loops, known as 'Loop Optimization Techniques (LOTs)', act upon a repeating group of statements which formulate the loop(s). Such techniques have a significant impact [14] because many programs spend a large portion of their execution time in loops, especially for managing the overhead due to the circular nature. Among the Loop Optimization techniques; 'Loop unrolling' is taken for this case study, as it opens up the additional scope for automated code generation and imparts better software parallelism.

The subsequent sections of this document describe the concept of Loop unrolling, its advantages and disadvantages with the help of meta-models from the real world as well as Computer Science and Software Engineering.

II. MATERIALS AND METHODOLOGIES

Materials used for this study are simple solution meta-models based on problems experienced during programming, simple gaming concepts, and observations from the real world. An underlying methodology is a Model-driven approach [3]. Also, this article includes the concepts related to problem-solving approaches of different persons while solving puzzles as well as the problems in work situations.

III. LOOP OPTIMIZATION THROUGH LOOP UNROLLING

Loop unrolling is a popular and widely used loop optimization technique [13]. It is also known as 'Loop Unwinding'. As the names indicate, it is like unrolling a rolled mat or unwinding a winded cable or opening a paper roll.

In the real-life, it is seen that while running in a circular, curved or zigzag path people get more tired, than while running in a straight and linear path. Similarly, the automated systems like computers have to put some extra effort and resources for managing the execution of conditional jumps, branching instructions, and maintenance of index variables required for implementing the loops.

The number of iterations, the more overhead will occur, even for checking the condition for terminating the loop as well as for updating of index variable.

Loop unrolling is implemented by eliminating the loops or reducing the number of iterations. This transformation can be undertaken by the programmer or an Optimizing compiler.

A simple example of Manual loop unrolling and a practical test with more iterations and repetitions are illustrated in the subsequent sections.

IV. Simple Example for Manual Loop unrolling

The problem is to update the content of an array keeping ages of 18 students of a post-graduate class with their ages after two years. The formula for calculating the age after two years is

$$\text{Age after two years} = \text{Present Age} + 2 \qquad (1)$$

Three different versions of source code for implementing the above equation(1) on all the elements of a single-dimensional array of size 18 are given below.

- Normal loop example without loop unrolling
- Loop with reduced iterations by applying partial loop unrolling
- After eliminating loop through complete loop unrolling

The three versions of the program are illustrated below with flow charts and sample source code.

A. Normal Loop Example without Loop unrolling

A flow chart illustrating the logic for implementing the first version, normal loop without any loop unrolling, is shown in Fig 1.

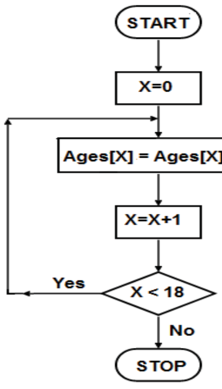


Fig. 1. Flow chart for the first version with NO Loop unrolling

A sample source code for implementing the above concept is as in Table I have shown below.

TABLE I. Source code for Normal loop example

```
for (intX = 0; intX< 18; intX++)
{
        arrAges[intX] = arrAges[intX] + 2;
}
```

This version is usually written by programmers in their programs; without applying any optimizations for the runtime.

B. Loop with reduced iterations by applying Partial Loop unrolling

A flow chart illustrating the second version, loop with a reduced number of iterations by applying partial Loop unrolling, is shown in Fig.2.

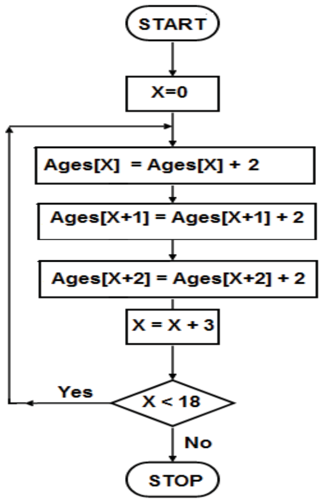


Fig. 2. Flow chart for the second version with Partial loop unrolling

A sample source code for implementing the above concept of partial loop unrolling is as in Table 2.

TABLE II. Source code for partial loop unrolling

```
for (intX = 0; intX< 18; intX+=3)
{
        arrAges[intX]   = arrAges[intX]   + 2;
        arrAges[intX+1] = arrAges[intX+1] + 2;
        arrAges[intX+2] = arrAges[intX+2] + 2;
}
```

Here, a set of three statements are executed within each iteration so as to reduce the number of loop iterations to one third.

C. After eliminating Loop through Complete Loop unrolling

A flow chart illustrating the third version, the source code after eliminating the loop though complete loop unrolling, is shown in Fig.3.

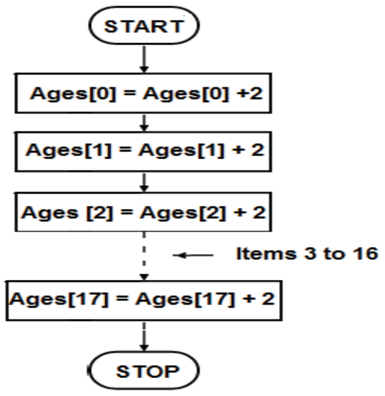


Fig. 3. Flow chart for third version with complete loop unrolling

A sample source code for implementing the above concept of complete loop unrolling is as in Table 3.

TABLE III. SOURCE CODE FOR COMPLETE LOOP UNROLLING

```
{
arrAges[0]  = arrAges[0]  + 2;
arrAges[1]  = arrAges[1]  + 2;
arrAges[2]  = arrAges[2]  + 2;
arrAges[3]  = arrAges[3]  + 2;
arrAges[4]  = arrAges[4]  + 2;
arrAges[5]  = arrAges[5]  + 2;
arrAges[6]  = arrAges[6]  + 2;
arrAges[7]  = arrAges[7]  + 2;
arrAges[8]  = arrAges[8]  + 2;
arrAges[9]  = arrAges[9]  + 2;
arrAges[10] = arrAges[10] + 2;
arrAges[11] = arrAges[11] + 2;
arrAges[12] = arrAges[12] + 2;
arrAges[13] = arrAges[13] + 2;
arrAges[14] = arrAges[14] + 2;
arrAges[15] = arrAges[15] + 2;
arrAges[16] = arrAges[16] + 2;
arrAges[17] = arrAges[17] + 2;
}
```

Here all the 18 statements are coded together so as to avoid the looping completely. The number of lines in the code is minimum in the normal loop, but it has to perform the condition check and incrementing of the index variables 18 times. This involves branching instructions, storage of temporary variables and jumps. But the partial loop unrolling helps to reduce the number of iterations and the complete loop unrolling helps to completely eliminate the iterations so as to save the time required during execution for performing the loop controlling activities.

The above example is too simple and just for understanding the concept of loop unrolling. There are only 18 statements to execute. In most of the modern computer systems, all of the above three variations will get executed within a fraction of millisecond, regardless of having a loop or not having a loop.

For seeing the difference among the methods with loops and without loop in respect of the execution time, testing [10] shall be done with sufficiently large number of steps for execution; i.e. more iterations in programs having loop(s) and more repetitions of statements in the equivalent programs without having loop(s).

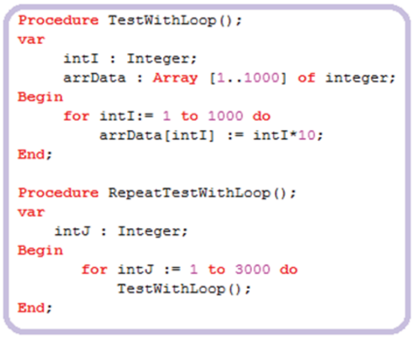
V. PRACTICAL TEST WITH MORE COMPLEXITY

A test with more complexity, in this case, means the one with a higher number of iterations and/or repetitions.

A practical test, conducted in a computer having an average speed, with a loop of simple operation of assigning a value (index multiplied by 10) into an array of size 1000 is illustrated below. The source code written in Pascal language with and without a loop is given. The execution time-analyzed by compiling and running with IDE of Delphi [8] (by switching off the code optimization options of the compiler).

Even the 1000 iterations will not be a sufficient sample size for experiencing the difference between the performance of the two methods (the one with loop and the one without loop). In order to impart sufficient complexity, the testing procedure is again executed several times, say 3000 times.

Applying the Rules of sums and products in Discrete Mathematics [9], a total of 1000x3000 = 30,00,000 (Thirty lakhs) statements are there for execution in both the methods.



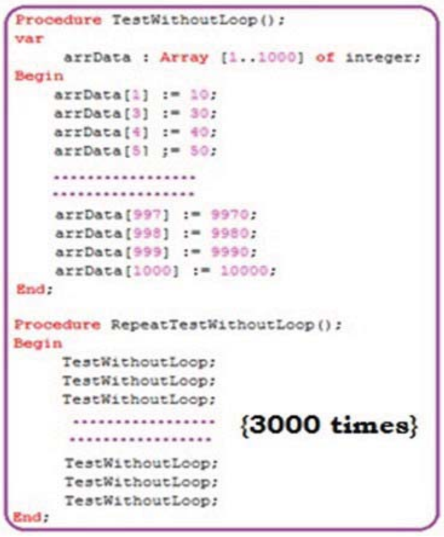
```
Procedure TestWithLoop();
var
     intI : Integer;
     arrData : Array [1..1000] of integer;
Begin
     for intI:= 1 to 1000 do
         arrData[intI] := intI*10;
End;

Procedure RepeatTestWithLoop();
var
    intJ : Integer;
Begin
       for intJ := 1 to 3000 do
           TestWithLoop();
End;
```

Fig. 4. Pascal code for testing WITH Loop (procedure having 1000 iterations and repeating the procedure 3000 times)

Applying loop unrolling, the above procedures are modified by replacing the loops with individual statements. Both the procedures (the one for testing purpose and the other for repeating the same) are modified accordingly.

The loop having 1000 iterations are replaced with 1000 individual statements. Similarly, the loop for repeating the testing procedure 3000 times has been replaced with 3000 individual procedure calls. The source code is shown in Fig.5.



```
Procedure TestWithoutLoop();
var
      arrData : Array [1..1000] of integer;
Begin
     arrData[1] := 10;
     arrData[3] := 30;
     arrData[4] := 40;
     arrData[5] ;= 50;

     ................
     ................
     arrData[997] := 9970;
     arrData[998] := 9980;
     arrData[999] := 9990;
     arrData[1000] := 10000;
End;

Procedure RepeatTestWithoutLoop();
Begin
      TestWithoutLoop;
      TestWithoutLoop;
      TestWithoutLoop;
       ................    {3000 times}
       ................
      TestWithoutLoop;
      TestWithoutLoop;
      TestWithoutLoop;
End;
```

Fig. 5. Pascal code for testing WITHOUT Loop (procedure having 1000 statements and repeating the procedure 3000 times)

An analysis of the results of the above program is given in the next section.

VI. ANALYSIS OF TEST RESULTS

A summary of the observations after executing both the versions (with loop and without loop) of the program mentioned above is given below.

TABLE IV. SUMMARY OF TESTING WITH STANDARD WORKLOAD

Parameter being observed	*Version with Loops*	*Version after Unrolling Loops*
Load of the actual task (number statements excluding those for controlling loops) in each epoch	1000 x 3000 =**30,00,000**	1000 x 3000 =**30,00,000**
Number of Testing Epochs	**100**	**100**
Range of execution time (in milliseconds)	**0 -- 16**	**0 – 16**
Average Execution time (in milliseconds)	**12.85**	**2.17**
Variance in Execution Time σ^2	**34.3712**	**29.2536**
Standard Deviation σ	**5.8627**	**5.4087**
Standard Error $\frac{\sigma}{\sqrt{N}}$	**0. 58627**	**0.54087**

The deviations from the mean values show that the execution time cannot be mathematically predicted, but can be statistically analyzed as in Stochastic Processes.

A graphical representation of process progress showing the upper limit (rounded towards the ceiling) of the execution time in milliseconds is shown in Fig6. In the graph, the red line represents execution time for procedures with loop and blue dots indicate the execution time for procedures without loops.

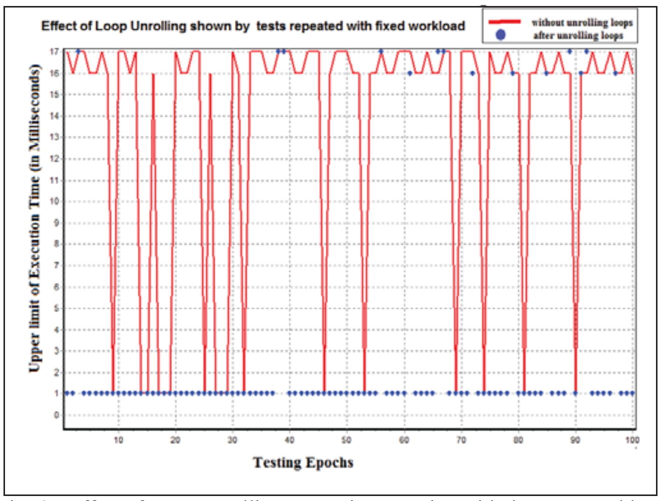


Fig. 6. Effect of Loop Unrolling at Runtime: Testing with the same workload

It is clear that the execution time is varying slightly for each testing epoch. This may be because execution time depends also on the process scheduling and optimization requirements of the operating system. Such external factors are having a significant role in determining the execution time of programs for each execution epochs as the process is tested in a multiprogramming environment. However, from the graph, it is obvious that the overall trend shows that the time for executing the code containing loops is higher than the time required for code after the elimination of loops through loop unrolling.

Moreover, for analyzing the difference, a more comprehensive test has been conducted with incrementing workload based on the durations for repeatedly executing the procedures 'TestWithLoop' and 'TestWithoutLoop' each having 1000 statements. By repeating them 100 times, facilitated for executing a load of 1 Lakh statements. 1000 such testing epochs were conducted in an increasing manner so as to find out the execution time for running 1 lakh statements, 2 lakhs statements, and so on up to 1000 lakhs statements. The execution times for each of the 'incrementing epochs' were plotted in a graph for both versions of code, 'TestWithLoop' and 'TestWithoutLoop'. The graphical output of the analysis is given below in Fig 7.

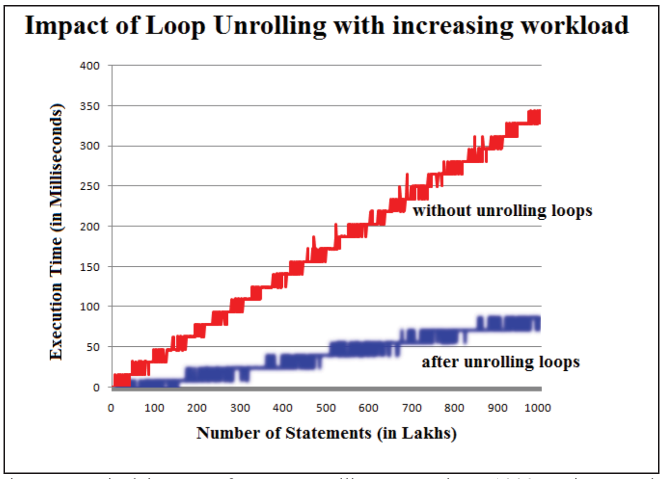


Fig. 7. Magical impact of Loop Unrolling at runtime: 1000 testing epochs with increasing workload by repeating statements in the range of multiples of lakh.

The magical effect of eliminating the loops by loop unrolling can be easily distinguished in the graph. The procedure with the loop takes too much time than the one without loops during execution at runtime. Also, while running programs with a higher number of statements will help to visualize the reduction in time complexity achievable through loop unrolling.

A statistical summary of the execution details of the above 1000 testing epochs with the increasing workload is given in the following chart.

TABLE V. SUMMARY OF TESTING WITH INCREMENTING WORKLOAD

Number of Statements (in Lakhs)	Execution Time (in Milliseconds)					
	Without Loop Unrolling			After Loop Unrolling		
Class Interval	Min	Max	Average	Min	Max	Averag e
1 - 100	**0**	**47**	**16.74**	**0**	**16**	**4.21**
101 - 200	**31**	**78**	**51.47**	**0**	**31**	**14.05**
201 - 300	**62**	**110**	**86.19**	**15**	**32**	**24.49**
301 - 400	**94**	**141**	**120.89**	**15**	**47**	**32.00**
401 - 500	**125**	**187**	**154.43**	**31**	**47**	**41.90**
501 - 600	**171**	**203**	**188.22**	**46**	**63**	**52.06**
601 - 700	**202**	**265**	**221.86**	**46**	**78**	**60.79**
701 - 800	**234**	**281**	**258.31**	**62**	**78**	**69.58**
801 - 900	**265**	**312**	**288.56**	**62**	**94**	**80.04**
901 - 1000	**296**	**344**	**324.34**	**78**	**94**	**87.75**

The above testing epochs for the two versions (with loop and without loop) of the same program were executed in the same computer system with common conditions of external factors such as development and runtime environments without altering the hardware configuration. The only difference was the versions of program codes, one without applying loop unrolling and other after effecting loop unrolling. Hence the above values are comparable. It may be noted that values related to execution time can be different while running in different computers or in the same computer by changing the run time environment (even other programs were working in parallel) or hardware configurations. However, the values, if generated under common external factors will still be comparable, i.e. for the purpose of analyzing the impact of loop unrolling. While experimenting in computers with higher speed, more workload may become essential to distinguish among the variations in execution time.

VII. Advantages and Disadvantages of Loop Unrolling

The suitable analogies from the real-life for explaining the advantages and disadvantages of optimizing loops through Loop unrolling are given below.

- ✓ Keeping a bundle of food items in the packed form may take extra time while serving. But unpacking them and moving them to easy to open containers will make it easy while serving. But it needs some extra effort for preprocessing (opening the bundle, moving items to containers). Also, maintaining the unpacked items may demand extra care and space than while in the packed form.
- ✓ Limiting the number of purchases (changing daily purchase to weekly activity) will help to save time and to reduce the overhead cost of repeated transportation. But purchasing items for one week at a time needs some extra effort and affordability.
- ✓ Avoiding a circular path and insisting for a linear path is not always beneficial. Consider the funny analogy of a Doctor advising a patient to come again after running 10 KMs daily for one month. If the patient keeps on running in a Linear path (without coming back) as in Loop Unrolling, the patient will reach 300 km away from the Doctor after one month. Similarly, extremely opposite results can occur if implemented without properly understanding the actual project requirements as well as the suitability of loop unrolling for the context.

A. Advantages

The major advantages of Loop unrolling are listed below.

- As in other loop optimization techniques, Loop unrolling prepares the program for improved performance at runtime.
- The overhead in "tight" loops often consists of simple instructions like incrementing a pointer or end of loop tests. If the compiler or assembler can pre-calculate the offsets to each individually referenced array variables, no additional arithmetical operations are required at run time
- Branch penalty is getting minimized
- Scope for applying *Machine Learning* techniques in complex analysis and high-level synthesis [16]

B. Disadvantages

The major disadvantages of loop unrolling are listed below.

- Increased Program code size which is not desirable in most of the contexts, especially in embedded applications
- Can also cause an increase in cache misses, which may adversely affect the performance
- Unless performed transparently by an optimizing compiler, the codes may become less readable and less manageable
- If there are subroutine calls within the loops, unrolling insist Inlining (substituting the subroutine call with its actual content) and will cause an excessive increase in code size.
- Possible increment in usage of Registers for maintaining temporary variables.
- Apart from very small and simple code lines, unrolled loops that contain branches may become slower even than recursions.
- Special exploration strategies have to be developed for estimating loop unrolling factors especially in the case of High-Level Synthesis [15].
- Above all, if done without proper intention and if not demanded by the optimization requirement of the project, Loop unrolling can give totally negative outcomes, as mentioned above in the analogy of patient running daily in the style of loop unrolling.

VIII. Scope for Achieving Additional Advantages

Apart from the advantages and disadvantages mentioned above, when being done manually and intentionally, the loop unrolling can provide two additional advantages at the implementation level.

- ✓ Enhanced scope for Software Parallelism
- ✓ Scope for automated systems for code generation

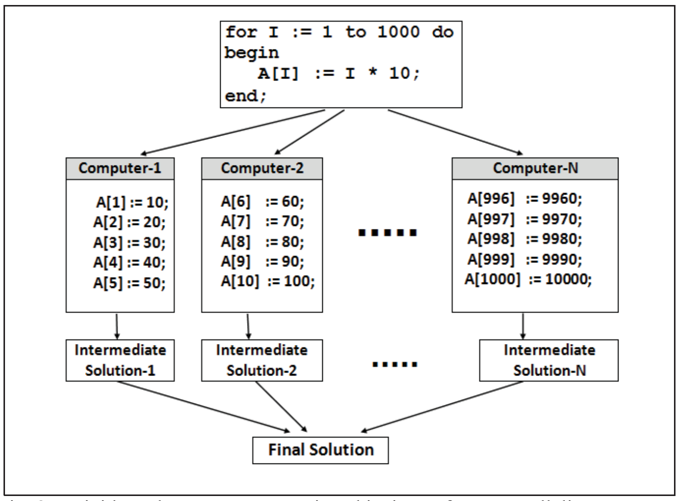
These two concepts are explained below.

A. Enhanced scope for Software Parallelism

As the program statements obtained through loop unrolling are mostly independent of each other, they exhibit better software parallelism [7] than the statements in Loops. They can be executed in parallel applying the *Divide and conquer approach* [4]. Even if there is a large number of statements they can be processed by splitting into different groups. Also, it facilitates to combine intermediate solutions from each group to get the ultimate solution as in.

$$A \oplus B \oplus C \oplus D \oplus E = (A \oplus B) \oplus (C \oplus D) \oplus E \qquad (2)$$

This Associative nature imparts the scalability [6] also. Even if there are a hundred or thousand or even infinitely large numbers of statements, the same Divide and conquer approach can be made applicable [1] as shown in Fig-8.



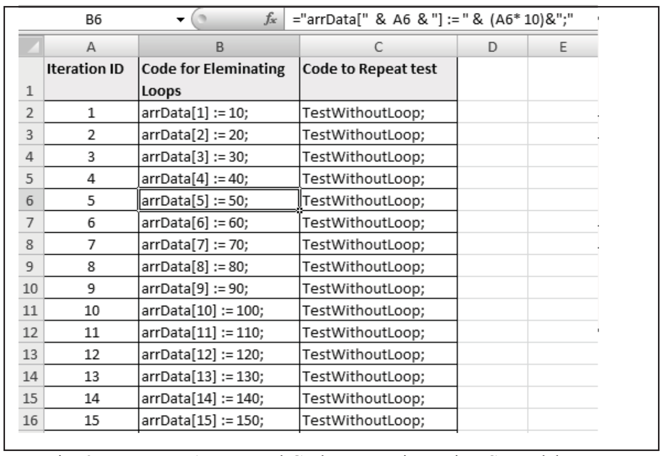
```
for I := 1 to 1000 do
begin
   A[I] := I * 10;
end;
```

Fig. 8. Divide and conquer strategy in achieving software parallelism.

B. *Possibility for Automatic Code generation*

The program statements getting generated through Loop unrolling are mostly of similar structure with minor differences. Hence they can be generated automatically, as in Model-Driven Development[3], using another program or even with a spreadsheet package like Microsoft Excel as shown in Fig.9. The formula in the cell B6 for generating the source code for assigning value to an array element (Fifth Item) is as follows

$$= \text{"arrData["} \ \& \ A6 \ \& \ \text{"] := "} \ \& \ A6*10 \ \& \ \text{";"} \qquad (3)$$



B6 fx ="arrData[" & A6 &"] :=" & (A6* 10)&";"

	A	B	C	D	E
1	Iteration ID	Code for Eleminating Loops	Code to Repeat test		
2	1	arrData[1] := 10;	TestWithoutLoop;		
3	2	arrData[2] := 20;	TestWithoutLoop;		
4	3	arrData[3] := 30;	TestWithoutLoop;		
5	4	arrData[4] := 40;	TestWithoutLoop;		
6	5	arrData[5] := 50;	TestWithoutLoop;		
7	6	arrData[6] := 60;	TestWithoutLoop;		
8	7	arrData[7] := 70;	TestWithoutLoop;		
9	8	arrData[8] := 80;	TestWithoutLoop;		
10	9	arrData[9] := 90;	TestWithoutLoop;		
11	10	arrData[10] := 100;	TestWithoutLoop;		
12	11	arrData[11] := 110;	TestWithoutLoop;		
13	12	arrData[12] := 120;	TestWithoutLoop;		
14	13	arrData[13] := 130;	TestWithoutLoop;		
15	14	arrData[14] := 140;	TestWithoutLoop;		
16	15	arrData[15] := 150;	TestWithoutLoop;		

Fig. 9. Automated Code generation using Spreadsheet

This technique can be considered as a meta-model while designing software robots for developing software programs automatically from high-level designs and specifications.

IX. CONCLUSION

Loop unrolling is a Loop optimization technique for reducing circular paths in program execution through substituting with their linear alternatives. This can bring considerable improvement in the performance of the software at runtime, obviously by optimizing the overall execution time.

Apart from the usual merits of loop optimization techniques, if implemented intentionally in a wise manner, and if the nature of the project's permits, loop unrolling technique can also impart enhanced software parallelism and open the scope for automated code generation. So it can be considered as a meta-model for architecting software robots capable of performing many complex tasks in the real world as well as research problems. As far as the software industry is concerned, such robotic software models can become useful for performing many of the software development activities automatically in a parallel and distributed manner.

REFERENCES

[1] S. A. Kumar, "Achieving software parallelism through alternative process models," 2017 International Conference on Inventive Systems and Control (ICISC), Coimbatore, 2017, pp. 1-4.

[2] David B. Kirk and Wen-mei W. Hwu, "Programming Massively Parallel Processors A Hands-on Approach", Morgan Kaufmann by Elsevier, 2010

[3] Thomas Stahl and Markus Völter, "Model-driven software development : technology, engineering, management ," John Wiley & Sons Ltd, West Sussex, England ,2006

[4] Q. Yang and C. Yu, "Parallelization by the divide-and-conquer method," [Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics, Chicago, IL, 1992, pp. 1265-1270 vol.2.

[5] A. Rogers and K. Pingali, "Compiling for distributed memory architectures," in IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 3, pp. 281-298, Mar 1994.

[6] R. Yang and J. Xu, "Computing at Massive Scale: Scalability and Dependability Challenges," 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, 2016, pp. 386-397

[7] G. W. Cobb, "The impact of parallelism on software," 1975 IEEE 3rd Symposium on Computer Arithmetic (ARITH), Dallas, TX, USA, 1975, pp. 220-222.

[8] M. Cantũ, "Mastering Delphi 5, "Mastering Series, Sybex, 1999.

[9] Sudarshan Iyengar,"Motivation for Counting", NPTEL e-Learning course in Discrete Mathematics

[10] Rajib Mall,"Software Testing", NPTEL e-Learning course in Software Testing

[11] Teixeira, Thiago SFX, Corinne Ancourt, David Padua, and William Gropp. "Locus: a system and a language for program optimization." In Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization, pp. 217-228. IEEE Press, 2019.

[12] Knaust, Marius, Florian Mayer, and Thomas Steinke. "OpenMP to FPGA offloading prototype using OpenCL SDK." In 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 387-390. IEEE, 2019.

[13] J. C. Huang and T. Leng, "Generalized loop-unrolling: a method for program speedup," Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122), Richardson, TX, USA, 1999, pp. 244-248.

[14] Y. Dong, J. Zhou, Y. Dou, L. Deng and J. Zhao, "Impact of Loop Unrolling on Area, Throughput and Clock Frequency for Window Operations Based on a Data Schedule Method," 2008 Congress on Image and Signal Processing, Sanya, Hainan, 2008, pp. 641-645.

[15] P. R. Panda, N. Sharma, S. Kurra, K. A. Bhartia and N. K. Singh, "Exploration of Loop Unroll Factors in High Level Synthesis," 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), Pune, 2018, pp. 465-466.

[16] G. Zacharopoulos, A. Barbon, G. Ansaloni and L. Pozzi, "Machine Learning Approach for Loop Unrolling Factor Prediction in High Level Synthesis," 2018 International Conference on High Performance Computing & Simulation (HPCS), Orleans, 2018, pp. 91-97