Realization of Parallelism in a Sequential Legacy 'C' Program

Padmapriya Patil, R. N. Kulkarni

***Abstract:** In the present era of high speed computation with the multicore and other parallel processors in the computational field, there are still some organizations which rely on their old software systems developed years ago, which over the time have been subjected to continous development by different developers. Even though these softwares persist with the old and little in use technology, they still work to satisfy the operational demands of the organizations and have kept them going in the competetive industry. These systems which have with time grown into legacy, embed the major business functionalities of the organization, which is but effort of years. Hence a methodology is required to rebuild the legacy system to make them suitable for execution on to the present computation systems. The paper discusses a research work, wherein work is done to realize points of latent parallelism in a sequentially executing legacy 'C' program which is initially restructured and the design information abstracted. A technique using finite state machine is proposed to identify tasks, events, processes and jobs in the program, which helps to locate functionally independent computational units in the program. Furthur using the slicing technique, slicing is performed to extract out the appropriate lines of codes defined by the slicing criteria, which assembled together form a functionality that can be executed in parallel with other extracted functional modules or computational units on any parallel computational platform.*

***Keywords:** Restructuring, Legacy Software System, Functional Modules, Program Slicing, Functional Dependency, Parallel Computation Systems*

I. INTRODUCTION

The Information Technology is one of the fastest growing industries today. A lot of newer software technologies as well as enhanced hardware computation systems have come up in the industry, making it fully advanced with respect to speed of operation, increased throughput and efficiency with reduced delay of any types. Before the advent of multicore or the multiprocessor systems, the Single core processors were considered to be the high performing systems. Their capacity of executing sequentially at very high speed, increased with increase of the chip density and these single core systems were outperforming in every field of computation.

But this increased density on a single chip is burdening the chip operation by creating drawbacks with excess utilization of energy, and high power dissipation. Adding to this, these single core processors do not provide for scaling. This has reduced the reliability of the single core processors.

In recent years introduction of parallel computation and parallel programming models in the IT industry has given a new alternative to the single core processors.

The parallelization technique has led to the introduction of multicore processors which work with multiple processors or cores embedded on a single die/chip. In contrary to this, there are many organizations in the industry, which own legacy software systems and have continued to be in operation relying on these legacy systems. The Legacy software systems are the in-operational software systems developed by the organizations in the beginning and which have gradually grown to be company's reliable software systems embedding all the crucial functionalities of business of the organization.These legacy systems may have been developed with the programming languages and on computational platform which were in use back then and may have been continuously worked upon by various developers. But with lot of emerging programming languages and enhanced computational platforms, the legacy software systems may appear obsolete and also show slower pace of operation, in comparison with the present technologies. On the other hand, even with all the prevailing problems, it may be harder for the organizations to think of replacing the legacy systems as these have been operational through years and also were reliable. And the major setback in the replacement is the business significant functionalities embedded in the legacy systems. This makes the organizations re-think about discarding the legacy systems. Under such conditions, the concept of reengineering can be used to rebuild the legacy systems allowing it to handshake with the advanced systems. The paper discuss the work which is in continuance to the earlier work published in [2][3] on the concept of reengineering of legacy 'C programming system to provide them a framework using which these legacy systems can transform their execution in accordance with the advanced technologies. Paper [2][3] discuss work related to reengineering of the legacy system by restructuring it and then abstraction of the control flow and data flow relative entities and tabulation of these as Information Flow Table(IFT). The generation of the Information Flow Table is discussed in paper [3] along with discussion on establishing of functional dependency within the derived attributes. The paper presents work done to identify the possible nodes for parallelization within the restructured program using the IFT and functional dependency.

Manuscript received on May 25, 2020.
Revised Manuscript received on June 29, 2020.
Manuscript published on July 30, 2020.
***** Correspondence Author

Padmapriya Patil*, Assistant Professor, Department of Electronics and Communication Engineering, P. D. A. College of Engineering, Kalaburagi, India. E-mail: padmazapur@gmail.com
Dr. R. N. Kulkarni Prof. & Head, Department of Computer Science & Engineering, BITM, Ballari, India. E-mail: rn_kulkarni@rediffmail.com

II. LITERATURE SURVEY

Reengineering is a vast and emerging field. A lot of research has been done to rebuild, reuse the existing systems and for many other requirements. Paper [1], the author has worked to reengineer a legacy ‘C’ program by successful abstraction of the design in the form of control flow and data flow. Paper[2][3][4], work has been done to restructure the legacy ‘C’ program to abstract out the structural and behavioural aspects in the form of control flow and data flow tables, paper[3] further shows development of the Information Flow Table(IFT) which represents the complete design information of the program. The attributes tabulated in the IFT are used to continue the work shown in paper [4] where functional dependency amongst the attributes is established. Paper [5][6][7], discuss work on parallelization techniques and evolving a sequentially executing program to parallel execution. Paper [8], discuss discovering the points of potential parallelization in any program. Paper [9][10], shows work carried out for implementation of parallelization driven techniques.

III. RESEARCH WORK CARRIED OUT

The research work proposed and discussed in the paper depends on the work developed and published previously. The research work mainly intends to reengineer a single core executing sequential legacy ‘C’ program such that it can be made amenable to a multicore executing platform. But this reengineering work is carried out at the very level of proximity relative to the original program, without manipulating the underlying or embedded functionality of the input program. The phases of the reengineering work are as discussed below.

A. Restructuring the Input Legacy ‘C’ Program

An executable ‘C’ program is considered as input to the work. A program is usually developed by one or more developers to execute a particular application. During the course of application development, requirements are defined initially and program developed accordingly. With continuous usage of this application program new requirements may arise and thus the need to change the program accordingly. Developers many times leave comment lines to increase understandability, but as the program is worked upon based on requirements of different developers, it may become non maintainable. This reduces the value of the non executable information included by developers earlier. There may also be much information like blank lines; multi-statement lines etc which fail to carry any executable data. Thus before beginning the work of reengineering, restructuring of the original program becomes necessary to increase readability of the input program for further work on it. Thus the input program is subjected to restructuring. Restructuring is a basic process defined in the work, wherein a program is written to parse the input program to identify comment lines, blank lines and remove them. The multistatement lines are converted into single statements and redundancy is identified and removed. Work is also done to eliminate some unconditional structures or loop which tend to change the continuous structure of the program like the *goto* statement, which is an unconditional jump construct, which take the control to an unknown location and is replaced with appropriate replacements without disturbing the functionality. Thus to mould the program for in-depth analysis of it, restructuring is required. Finally line numbers are given to the structured program which is required for program flow analysis [2].

B. Restructured Program Flow Analysis

The program flow analysis of the restructured program is required to obtain the structural i.e the syntactic information recorded as control flow information and the semantic i.e the behavioral information, recorded as the data flow information.[3]

C. Generation of Information Flow Table (IFT)

The information flow table is generated with the help of the extracted information during program flow analysis [3]. To generate the IFT, the control flow and data flow along the program are traced and tabulated. The control flow table (Ref [2]), mainly involves mapping of the control constructs along with their start and end line numbers. The controls are also mapped with respect to their transitions.

The Data flows in program, depending upon the requirements of the execution of control structures. Data Flow represents the flow of signals which set-off the functions for operation. The Data flow is traced and recorded along with the control constructs. Together the tabular column generated for mapping of control flow and data flow is termed as Information Flow Table (Ref [3])

IV. PROPOSED METHODOLOGY

A. Establishing Functional Dependency to deduce Relevant Attribute Set.

The information mapped out by tracing the legacy programming system represents the absolute constitution of the design of the program. This data is analyzed to understand the functionality ingrained across the program and can also be used to study the dependencies that exist between the functionalities. On establishing mutual dependency relation amongst these functional entities and analysis of these, relevant set of attributes can be deduced which can be used as a criterion for slicing that respective functional computational unit, involving the attributes of the set. The nodes where parallelization can be induced are identified [4].

B. Identification of Nodes for Parallelization using a Finite State Machine (FSM)

A programming systems working to execute major application of an organization, embeds many functionality spread along the system. These functionalities may work as subprograms of the main application program. The functionalities may be dependent on each other operationally or may be independently executing functions.

Thus by identifying such functions or their functional points, parallelism can be induced in the sequentially executing program by extracting out such functional modules and executing them in parallel.

The Slicing criterion defined in the earlier section provides for abstraction of functional modules, relative to the attributes. To assist the slicing technique a Finite State Machine is designed with its states defined to identify the nodes in the program which define control structures, their execution status, and transfer of control, control dependence and data dependence.
The states of the FSM also detect the inline preceding controls structures dependent on the former control and the following statements under the scope of the control construct executing with or without expression of functionality. These nodes or points of controls or data are identified as Events at first encounter, as Process if controls found in succession, with dependency on the former control structure and if any statements with data flow are encountered, they are identified as Jobs respectively. Then the Finite State Machine (FSM), designed using Mealy Machine , is run considering the input restructured program as one task, which on execution scans the input program to identify the presence of events, process and jobs. On the basis of the dependency shown by these, the parallelizable modules are identified and using the slicing technique with the criteria defined these functionally independent computation modules are sliced which can be executed in parallel on a parallel computational system like a multicore processor.

V. CASE STUDY

The work carried out so far are Restructuring of Input program [2], Generation of Information Flow Table [3], Deriving control and Data dependency, Establishing of functional Dependency between control flow and data flow attributes, Realizing of the Relevant attribute set, Deducing of the Slicing Criteria[4].

Design of Finite State Machine (using the Mealy Machine) to represent the state of the Input program

Input: Restructured Legacy ‘C’ Program

Task ➔ Input Program

Events ➔ Control structure encountered for the first time, during parsing of input program (IFT as lookup table)

Control Structures: C_i 🡐 {C_1, C_2, C_3, ……….C_n }, Where i= line number where the control was encounters in succession in the restructured program.

Process ➔ control structures encountered within the Event

Process P_i 🡐 {P_1,P_2,P_3…..P_n}

Jobs ➔ If no more controls are implicitly found, then the statements within or outside the scope of controls, defined by data flow and defining the behavior of any function under the influence of the data are termed as jobs.

Jobs J_i 🡐 {J_1, J_2, J_3,……..J }

Consider design of a 3 bit Mealy machine,

For every change of state, one condition is assigned

000 Event - ‘N’ (N=> No) // this state implies no event has occurred

001 Event - ‘Y’(Y=> yes) // An event has occurred

010 Event- ‘Y’ and condition ‘F’// Event occurred but condition is false i.e. there may be transition of control.

011 Event - ‘Y’, condition ‘T’ // Condition is true, no transition of control takes place

100 Event- ‘Y’, condition ‘F’ -> No process -> No jobs, control exists but no statements defining process or jobs exists.

101 Event- Y, condition F -> process -> jobs; Controls, implicit controls and statements defining process and jobs exists.

110 Event- Y, condition T -> No process -> No jobs

111 Event -Y, condition T -> process -> jobs

State	Status (Event)	Condition T/F	Process Y/N	Jobs Y/N
000	N	-	-	-
001	Y	-	-	-
010	Y	F	-	-
011	Y	T	-	-
100	Y	F	N	N
101	Y	F	Y	Y
110	Y	T	N	N
111	Y	T	Y	Y

With every change in the state, the FSM identifies the occurrence of Task followed by Event, if event occurs, it continues to identify whether condition of the event is true or false, if condition is true, then it identifies whether the processes are present or not and further jobs are present or not. Thus the main function of the FSM is to check the occurrence of Events representing main control structure, Processes representing sub or successive control structures with the scope of the former structure and jobs representing the executable statements, again within the scope of the control. If Statements are found in continuous outside the scope of any control structures, then they are independent jobs.

Input Restructured program (Source [3])

```
1 void main ()
2 {
3 int a, b, i, n, sum, add, func;
4 printf("Enter n value");
5 scanf("%d", &n);
6 clrscr();
7 if (n>0)
8 {
9 for(i=0; i<n; i++)
10 {
11 printf("Enter two nos");
12 scanf("%d%d", &a, &b);
13sum=a+b;
14 printf("Sum of 2 nos = %d", sum);
15 }
16 for(i=0; i<n; i++)
17 {
18 printf("Enter two nos a,b");
19 scanf("%d%d", &a, &b);
20 add=a+b;
21 printf("Sum of two
nos=%d", add);
22 avg=add/2;
23 }
24 for(i=0; i<n; i++)
25 {
26 printf("Enter two nos");
27 scanf("%d%d", &a,
&b);
28 f=a-b;
29 printf("Diff of two nos=%d", func);
30 }
31 for (i=0; i<n; i++)
32 {
33 printf("Enter two nos");
34 scanf("%d%d", &a, &b);
35 f=a%b;
36 printf("Modulus of two nos=%d", func);
37}
38 }
39 else
40 {
41 printf("No is not valid");
42 }
43 }
```

Output Program and Analysis:

The FSM identifies the input program as one complete task **Task T**

No event has occurred; hence it considers the statements encountered as jobs.

1 J_1:void main ()
2 {
3 J_2:int a,b,i,n,sum,add,func;
4 J_3: printf ("Enter n value");
5 J_4: scanf("%d", &n);
6 J_5: clrscr();

//Break Point 1 at line 7: Control C_7 (Ref: [3]);
//Encountering of control statement
//State: S_1 or Event E_1: (T/F)

As the restructured program is scanned down for more events, process and jobs and if they occur, then it is defined as the break point or the node which can be considered as potential point for inducing parallelism.

7 if (n>0) // Event E_1; continue to check for process
8 {

//Break Point 2 at line 9: Process P_1=C_9 (control structure at line 9)

C9: 9 J_1: for(i=0; i<n; i++)
10 {

No more control structures found, hence no sub processes. Executable statements are encountered, these are identified as jobs.

11 J_2: printf("Enter two nos");
12 J_3: scanf("%d%d", &a, &b);
13 J_4: sum=a+b;
14 J_5: printf("Sum of nos = %d", sum);
15 }

//Break Point 3: Process P_2=C_{16}

C_{16}:16 J_1:for(i=0; i<n; i++)
17 {
18 J_2 :printf("Enter two nos a,b");
19 J_3:scanf("%d%d", &a, &b);
20 J_4:add=a+b;
21 J_5:printf("Sum of nos=%d", add);
22 J_6: avg=add/2;
23 }

//Break Point 4: Process P_3=C_{24}

C_{24}:24 J_1:for(i=0; i<n; i++)
25 {
26 J_2:printf("Enter two nos");
27 J_3:scanf("%d%d", &a, &b);
28 J_4:f=a-b;
29 J_5:printf("Diff of nos=%d", func);
30 }

//Break Point 5: Process P_4=C_{31}

C_{31}:31 J1:for(i=0; i<n; i++)
32 {
33 J_2:printf("Enter two nos");
34 J_3:scanf("%d%d", &a, &b);
35 J_4:f=a%b;
36 J_5:printf("Mod of nos=%d", func);
37 }
38 }

// If Event E_1: False (transition of control)
Process P_5

C_{39}: 39 J_1: else
40 {
41 J_2: printf ("No is not valid");
42 }
43 }

The control structures encountered are the break points of parallelization and these control structures at the break points are termed as events or processes depending upon the dependency they exhibit with their former or latter control structures. Thus these are considered for inducing of parallelism. Further with these points as reference and the slicing criteria deduced[4], the functionally independent computational units can be sliced or extracted out for parallel execution.

VI. CONCLUSION

Parallelization, parallel computation and parallel programming models are the newest technologies in the Information Technology industry presently. The paper presents a research work, wherein, work is carried out to parallelize the single core executing sequential programming systems. The research attempts to rebuild these legacy systems by reengineering, such that they can be transformed to adapt to the enhanced parallel computation systems. The work of reengineering of legacy systems is carried out in phases, which begins with restructuring of the legacy program, abstraction and mapping out of the absolute design in the form of information flow.

Then using the concept of functional dependency a slicing criteria is deduced, the paper discuss a work wherein finite state machine is used , the states of the which are used to define the points in the program, where the program can be parallelized, thus inducing parallelism in the sequential legacy program.

REFERENCES

1. Rajkumar N. Kulkarni, "Reengineering of the legacy 'C' systems, Ph.D. thesis, 2011.
2. Dr Rajkumar N. Kulkarni, Padmapriya Patil, "Restructuring of Legacy 'C' Program to be Amenable for Multicore Architecture", ICRTEST Elsevier energy procedia proceedings, Oct- 2016.
3. Dr Rajkumar N. Kulkarni, Padmapriya Patil, "Abstraction of Information Flow Table from a Restructured Legacy 'C' program to be amenable for Multicore Architecture" LNCS Springer 2018
4. Dr R.N Kulkarni, Padmapriya Patil, "Abstraction of Functional Dependency and Information Flow from a Restructured Legacy 'C' program for Parallelization" IEEE digital Library-2018.
5. Alcides Fonseca, Bruno Cabral, Joao Rafael, Ivo Correia , Automatic Parallelization: Executing Sequential Programs on a Task-Based Parallel Runtime, International Journal of Parallel Programming, 2016
6. B. Bugeryaa, E. S. Kimb, and M. A. Solovev, Parallelization of Implementations of Purely Sequential Algorithms, ISSN 0361-7688, Programming and Computer Software, 2019
7. Anne Meade, Jim Buckley, J. J. Collins, "Challenges of Evolving Sequential to Parallel Code: An Exploratory Review", ACM, 2011.
8. Zhen Li, Ali Jannesari, and Felix Wolf, "Discovery of Potential Parallelism in Sequential Programs", 2014
9. Joao Rafael, Ivo Correia, Alcides Fonseca, "Dependency-Based Automatic Parallelization of Java Applications", LNCS 8806, Springer, 2014.
10. S. Rul, H. Vandierendonck, and K. De Bosschere, "Function level parallelism driven by data dependencies," SIGARCH Comput. Archit. News, vol. 35, no. 1, pp. 55–62, Mar. 2007.
11. Miguel Angel Aguilar , Juan Fernando Eusse, Projjol Ray , Rainer Leupers , Gerd Ascheid , Weihua Sheng , Prashant Sharma ," Parallelism Extraction in Embedded Software for Android Devices",2015 IEEE.
12. Djordje Kovacevic, Mladen StanojevićVladimir Marinkovic, Miroslav Popovic, A Solution for Automatic Parallelization of Sequential Assembly Code, DOI: 10.2298/SJEE1301091K, Feb 2013
13. Alexandros V. Gerbessiotis, Using parallelism techniques to improve sequential and multi-core sorting performance, Aug 2016
14. Chen Tian · Min Feng · Vijay Nagarajan , Rajiv Gupta, Speculative Parallelization of Sequential Loops on Multicores, springer Int J Parallel Prog (2009)

AUTHORS PROFILE

Dr. R. N. Kulkarni holds a Ph.D in software Engineering from Visvesvaraya Technological University, Belagavi, Karnataka, India. He has more than 30 years of teaching experience and published nearly 65 publications in International journals/conferences and also national conferences. His area of research work are Software Engineering, DBMS, Soft Computing and Object Technology. He is presently working as Prof. & Head of Computer Science & Engineering Department at Ballari Institute of Technology & Management, Ballari, Karnataka, India.

Padmapriya Patil completed M.Tech. in Computer Network Engineering from Visvesvaraya Technological University, Belagavi, Karnataka, India. She has more than 17 years of teaching experience. Her areas of interest are Computer Communication and Networking, Software Engineering and Processors. She is the program coordinator for many Industry Institute Interaction Programs and Faculty Development Programs. She is pursuing her research in Software Engineering. At present she is working as Assistant Professor in Department of Electronics and Communication Engineering, PDA College of Engineering, Kalaburagi.