Perspectives in Communication, Embedded-Systems and Signal-Processing (PiCES) – An International Journal
ISSN: 2566-932X, Vol. 4, Issue 4, July 2020
Part of the Proceedings of the 1st All India Paper writing Competition on Emerging Research - PaCER 2020

Abstraction of Functional Modules from a Legacy 'C' Program using Program Slicing

Padmapriya Patil
Assistant professor, Department of Electronics and communication Engineering, PDACE, Kalburgi, Karnataka, padmazapur@gmail.com

Dr. R N Kulkarni
Prof. and Head, Department of Computer Science and Engineering, BITM, Ballari, Karnataka, rn_kulkarni@rediffmail.com

Abstract: ***The present computation industry is growing very fast. Lot of improvements have taken place, as a result of which high speed computation with inbuilt parallel computation is the emerging technology for many of the parallel or multicore processors. Thus this technology is replacing the earlier technologies of sequential computation, leaving very little scope for the sequentially executing programs and their supporting software systems. These systems can neither utilize the modern computing resources nor scale upto their demands. Also, in contrary to the modern computational systems, the sequential executing legacy software systems are the systems which are a result of continuous developmental work of many developers embedding valuable mission critical functionalities. These systems have thus become the major working resources of the organizations but lack the capacity to adopt to the advanced hardware computation domains. Hence it is difficult for the organizations to disown these legacy systems or to maintain them. Thus a framework is required for bridging the gap between the advanced computation systems and the legacy systems. This paper proposes a methodology for abstraction of functionalities from the legacy 'C' program using the technique of program slicing.***

Keywords: Reengineering; Legacy Software System; Functionality; Program Slicing; Functional dependency; Computation Systems

I. INTRODUCTION

Legacy software system is the computing software system using programming language and also running application programs on a hardware domain, which are obsolete and restrict the systems from communicating with the current technologies. The legacy systems thus do not have the efficiency to allow for growth. Further, the legacy system can cause many problems, such as very high maintenance costs, they work with technologies that prevents integration between legacy and other systems and limited security. All these problems become more prominent, than the convenience of continuing the use of legacy systems.

But these legacy systems remain in use only because of their reliability, i.e. the systems continue to meet the needs they were earlier designed for. Hence it becomes difficult for organizations to abandon these legacy systems, which are required by the organizations to remain in business.

Enormous amount of such legacy software systems is prevailing in the software industries. Preserving these legacy systems may be by choice or necessity for the organizations as many of them are dependent on these for the execution of their mission critical application. Major reason for the organizations to still continue with legacy system is that it is steadily entrenched in company operations embedding decisive functionalities, and also the insight that, these system embodies a significant investment, and would be expensive to replace, further any change or enhancement can be expensive, time consuming, difficult to operate. But it is also required to ensure that mission critical legacy systems do become a major risk within the organization with the technology getting old, requiring specialized knowledge & skills to maintain them. It is therefore difficult for the legacy software systems to support increased throughput and efficient capacity and thus the growth of the company. As the industry grows, it demands the adaption to changing technology. Thus with the industry evolving constantly, the software must keep up with the advancing hardware computational technology, else the industry will have to settle operational business to the legacy system, slowly reducing the ability to growth.

The Computing hardware on the other hand is fast developing, on one end there are high performing single core processors with very high processing capabilities with increased transistor density. But at the same time this industry is facing serious problems due to the inability of the processors to scale.

The processor manufactures are also looking to increase the processing capabilities with multicore architectures and parallel programming models. In contrast to single core processing, the multiprocessing or multicore processing, prevent dependability on the software developer and controls task allocation using distributed approach for improved performance gains. With this insight into both legacy systems and advanced computation systems, it is required for the software developers to develop a framework which is capable of

execution of the legacy system on the present advanced computation systems.

This process of adaptation of the legacy system to advanced computation is significant and introduces new challenges. The legacy systems are operationally reliable but have through years failed to keep in pace with the performance of the advancing computation hardware. The continuous development on these systems to fulfil the then requirements over years and negligence in maintenance of these systems has complicated their readability and understandability. Hence, to evolve the legacy system, the major task of the software developer is to develop a framework for the reengineering of the legacy system for abstraction of functionality from the system and extract the functionally independent modules.

A methodology is required which reengineers the legacy system by initially restructuring it, analyze the legacy system to understand its behaviour and abstract the functionality.

The paper proposes a work which is in continuation to the earlier work on restructuring [4], wherein structuring of the legacy system makes suitable changes to the program without changing the underlying functionality, then abstraction of Information Flow Table; [5], this table is an informative representation of both the control flow and data flow information along the program. Further the information flow table is used to identify the functional dependency existing along the execution path of the program amongst the control structures and the data within these structures. The identification and abstraction of functional dependency relative to control and data dependency enables the extraction of relevant sets of attributes [6], which are used as criterion for recognizing the particular nodes in the program where parallelism can be induced and functional modules extracted using the technique of program slicing.

Thus the paper proposes a reengineering technique which not only works to make a legacy program compatible to execute on advanced single core platform but also is capable of addressing parallelization for execution of the sequentially executing program on multicore processors.

The work in this paper begins with considering of the relevant attribute sets as a criterion for identifying of those particular break points in the program which can be parallelized for execution on multi cores. Program slicing is performed at these break points using the relevant attribute set as the slicing criterion to abstract out the independently executing functional modules from the legacy program, which are applied for parallel execution onto multicore platforms.

II. LITERATURE SURVEY

Many researchers have worked on reengineering of legacy systems. Paper [1], the author has worked towards reengineering of legacy C programming system and has developed a tool for abstraction of control flow table and data flow graph from the input legacy ‘C’ program. Paper [4], work is done on restructuring of the legacy c program. A software tool is developed to read a input program and restructure the legacy c program by elimination or replacement of entities changing the structured sequence of program execution by an appropriate conditional or loop structure, at the very level of abstraction and in immediacy to the machine understanding. Paper [5], a tool is proposed here for the analysis of control flow and data flow of the legacy program and construct the Information Flow Table. Paper [6], work is done here to realize functional dependency within the control and data structures, and using the functional dependency relevant attribute sets are deduced which are used to define the appropriate nodes for modularization of the functions embedded in the legacy program. Paper [8] combines findings from selected literature identifying challenges encountered in the process of parallelizing sequential codebases to finally evolve the sequential code into parallel format. [9] Presents a dynamic approach for automatically identifying potential parallelism in sequential programs. It is based on the notion of computational units, which are small sections of code following the read-compute-write pattern that can form the atoms of concurrent scheduling. [7] discuss the relation between program slicing and data dependencies. Slicing is defined in the work and calculated parametrically on the chosen notion of dependency. [10] This work proposes an elegant way of achieving such a goal by targeting a task-based runtime which manages execution using a task dependency graph .A translator is developed for sequential JAVA code which generates a highly parallel version of the same program. This work in paper [12] presents a survey through different aspects considered over the time to detect and extract parallelism from sequential programs, and that could present any degree of applicability in modern multi-core embedded systems.

III. PROPOSED METHODOLOGY

The research work proposed and worked with in this paper is a continuation of the works of [6]. The research work begins with an objective of Reengineering of Legacy ‘C’ programming system by development of a framework which can be used for execution of the single core sequentially executing legacy ‘C’ program on multicore processor.

The work is carried out in phases with the output of every phase considered as the input to the next phase. The following describes the phases of work carried out so far and the ongoing work.

A. Phase 1

a) Restructuring of Legacy C program

The input legacy ‘C’ program is initially restructured to remove comment lines, blank lines, organize the program by converting multi-statement lines into single statements, and remove the unconditional goto statements with appropriate replacements to structure the program

flow. Finally line numbers are given to the structured program which is required for program analysis.[4]

b) Analysis of the Restructured Program

The Restructured Program is analyzed for the abstraction of the control flow and data flow information. The information obtained during the analysis is listed as a table of control constructs and variables [5]

B. Phase 2

a) The Information Flow Table (IFT)

The Information flow table, tabulates the flow sequence of control and data information along the program.[5]

b) Control Flow Information

Control flow information is extracted from the program and tabulated. The control flow table mainly involves the control constructs their start and end points and information relative to the transition.[1][5]

c) Data Flow Information

Data flows along the program in conjunction with the control constructs and the data is always in conjunction with the control flow. This flow of data along the program structure exhibits the behavior of the program with its flow. Accordingly the data variables are identified as Defined Variables and Referenced Variables [1]. The process begins in the control flow order where the information about the variables either defined or referenced or both is written along the line.[5]

C. Phase 3

a) Abstraction of Functional Dependency from the Information Flow Table.

Information embedded within a legacy software system represents the complete structure of the code design, data flow along the program and behavior of the program and control constructs with respect to the data at that instant.

The program and thus the legacy software system embed information which provides scope for the developers in the development of a framework for the legacy program execution in association with the advanced computation systems example multicore processing systems. [5]

b) Realization of the slicing criteria from the functional dependency table

The functional dependency table tabulates the control constructs and the type of dependency they share with each other. This table is generated with the information flow table and the restructured program as references.

By analyzing the information tabulated, relations are realized which includes the line number, respective control construct, data, the variable type and their interdependency.

The relation thus deduced is used as a slicing criterion for abstracting out the functionality embedded with the legacy program in the form of independently executing functional modules.[6]

D. Phase 4

a) Abstraction of functional modules using program slicing.

The slicing criterion deduced in the above phase is used for the abstraction of the functional modules.

The relations define the appropriate line number from where the slicing should begin; the control construct at which point node is identified for slicing, the data and its dependency is used to identify the scope of the module which is functionally independent for computation. Thus using the deduced relation, the embedded functionality can be abstracted form the legacy ‘C’ program. (Ref: case study)

b) Applying Program slicing technique for abstraction of functionality.

The static program slicing technique is applied for slicing out the functionally independent modules. The static program slicing technique is used, as the data in the flow considered is not real time and execution of the legacy ‘c program is neither required nor considered. The input legacy program is an executable program and is its working modules that can be executed independently are extracted out using the realized criteria. After identifying the nodes and slicing is performed to extract out the functional module.

IV. CASE STUDY

A. Restructured Input Program

```
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);
13 sum=a+b;
14 printf("Sumof two 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 }
```

Control structure at the entry point	Control structure	Conditional variables		Control constructs influenced by the entry point control structure	statements influenced by the control construct	Data Variables
		V_{cond}	V_{Ite}			
7	if	n		9,16,24, 31	-	-
9	for	n	i	-	11,12,13,14	a,b, sum
16	for	n	i	-	18,19,20,21, 22	a,b, add, avg
24	for	n	i	-	26,27,28,29	a, b, f
31	for	n	i	-	33,34,35,36	a, b, f
39	else			-	41	-

Table 1. Control and Data dependency Table

Inter Dependency	Explicit Intra dependency	Implicit Intra dependency	Data Dependency
$C_7 \leftarrow C_{39}$	$C_7 \leftarrow C_9, C_{16}, C_{24}, C_{31}$	$C_9 \leftarrow S_{11}, S_{12}, S_{13}, S_{14}$	Sum ← a , b
		$C_{16} \leftarrow S_{18}, S_{19}, S_{20}, S_{21}, S_{22}$	Add ← a , b Avg ← Add
		$C_{24} \leftarrow S_{26}, S_{27}, S_{28}, S_{29}$	f ← a, b
		$C_{31} \leftarrow S_{33}, S_{34}, S_{35}, S_{36}$	f ← a, b
	C_{39}	$C_{39} \leftarrow S_{34}$	-

Table 2. Functional Dependency Table

```
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("Diffof 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("Modof two nos=%d", func);
37 }
38 }
39 else
40 {
41 printf("No is not valid");
42 }
43 }
```

Tables 1 and 2 showcase generation of Functional Dependency Table.

The Functional dependency (FD) relations from this table are realized as shown below

$C_7 \leftarrow C_9, C_{16}, C_{24}, C_{31}$

$C_9 \leftarrow S_{11}, S_{12}, S_{14}, S_{15}$

Sum ← a, b

Similar to the above relations, other FD relations in the program may be used to realize the relations as criterion, as below

Criterion 1: (C_7, C_9, Sum, a, b).

Criterion 2: (C_7, C_{16}, Avg, a, b)

Criterion 3: (C_7, C_{24}, f, a, b)

Criterion 4: (C_7, C_{31}, f, a, b)

Criterion 2: (C_{39}, -, -, -, -)

Thus the slicing criterion for any module can be deduced as,

Ex: $C_7 \leftarrow C_9, C_{16}, C_{24}, C_{31}$

$C_9 \leftarrow S_{11}, S_{12}, S_{14}, S_{15}$

Sum ← a, b

Slicing criterion:

< S, V> = < (1:6, C7 (C9)), (a, b, sum)>

B. Results and Result Analysis

a) Control and Data dependency Table (Table 1)

The Table 1 is an abstraction of the complete behavioral, data flow and structural components of the program. It depicts the transition sequence of the control structures, dependency between them, binding structure and also the data flow through the constructs.

b) Functional Dependency Table (Table 2)

The table tabulates the complete sequence of the dependency existing between the control constructs and also data.

c) Analysis of Tables 1 and 2

The Table 1 begins with tabulation of the control structure at the entry point, followed by the conditional variable. This variable helps to trace whether the condition defined by the variable is influencing the execution of another construct within the scope of function Main ( ). If it influences any construct then the former and the latter exhibit interdependency. Further IFT (Information Flow Table) [5], is scanned to find any constructs executing within the entry point construct and if found they are tabulated as constructs influenced by the former. If the inner constructs also have constructs within, the process is repeated by considering this as new entry point. This exhibits intra control dependency. Simultaneously the data obtained along the control flow are also tabulated. The data is recorded at all the levels of program execution which may record redundant entities. This redundancy is minimized by establishing functional dependency.

In the table 2 Notation 'C' stands for control construct and 'S' stands for Statements affected by the control construct.

The control constructs in line 9,16,24,31 show dependency on the structure in line 7.This is the intra dependency

$C_7 \leftarrow C_9, C_{16}, C_{24}, C_{31}$, all these control constructs

i.e,. C_9, C_{16}, C_{24}, C_{31} are controls, intra dependent on C_7. Further all these constructs dependent on C_7 are scanned again to find more transitions, if transitions found then they are shown in the same way else if the loop is independent, then the statements depending on this construct are shown as ex., $C_9 \leftarrow S_{11}, S_{12}, S_{14}, S_{15}$. Finally the data dependency is abstracted as shown in the Table 2.

The Functional dependency (FD) relations from this table are realized for example as shown below

$C_7 \leftarrow C_9, C_{16}, C_{24}, C_{31}$

$C_9 \leftarrow S_{11}, S_{12}, S_{14}, S_{15}$

Sum $\leftarrow$ a, b

The above FD relations may be used to deduce the criterion.

This criterion given as an input for the slicing algorithm abstracts the functionality associated with the functional dependency relation, thus discovering the independently executing functional modules and inducing parallelization in the sequentially executing restructured input program.

Criterion 1: (C7, C9, Sum, a, b).
< (1:6, C7 (C9)), (a, b, sum)>
Input: Restructured Program
Output: Sliced functional module

```
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);
13 sum=a+b;
14 printf("Sum of two nos = %d", sum);
15 }
```

V. CONCLUSION:

In this paper a methodology is proposed for the abstraction of functionality or independently executing functional modules form a sequentially executing legacy 'C' program. In the earlier work carried out, work was done to generate the Information Flow Table which gave an analysis of the control flow and the data flow of the input program. With this as reference, the functional dependency table is developed, which defines transparently the dependency status with respect to all the control constructs, their inter dependency with each other and their dependency with the data and its flow along the program. This table is then analyzed to realize the slicing criteria, using which the independently executing functional modules are abstracted. Thus abstraction of functionality using Program Slicing is achieved.

REFERENCES

[1] Rajkumar N. Kulkarni, "Reengineering of the legacy 'C' systems, Ph.D. thesis, 2011.

[2] Chris Birchall, "Re-Engineering Legacy Software", Manning Publications, 2016

[3] Phillip A.Hausier, Mark G. Pleszkoch, Richard C.Linger and Alan R. Herner, "Using Function Abstraction to Understand Program Behaviour, IEEE, 1990.

[4] Dr Rajkumar N. Kulkarni, Padmapriya Patil, "Restructuring of Legacy 'C' Program to be Amenable for Multicore Architecture", ICRTEST Elsevier energy procedia proceedings, Oct- 2016.

[5] 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

[6] 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.

[7] Isabella Mastroeni, Damiano Zanardini, "Data Dependencies and Program Slicing: from Syntax to Abstract Semantics", PEPM'08, ACM, 2008.

[8] Anne Meade, Jim Buckley, J. J. Collins, "Challenges of Evolving Sequential to Parallel Code: An Exploratory Review", ACM, 2011.

[9] Zhen Li, Ali Jannesari, and Felix Wolf, "Discovery of Potential Parallelism in Sequential Programs", 2014

[10] Joao Rafael, Ivo Correia, Alcides Fonseca, "Dependency-Based Automatic Parallelization of Java Applications", LNCS 8806, Springer, 2014.

[11] 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.

[12] 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.