International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5837
***Article Received**: 27 September 2022 **Revised**: 09 November 2022 **Accepted**: 28 November 2022*

A Machine Learning Technique for Abstraction of Modules in Legacy System and Assigning them on Multicore Machines Using and Controlling p-threads

Vinay T R[1], Ajeet A Chikkamannur[2]
[1]Dept. of Artificial Intelligence and Data Science, Ramaiah Institute of Technology,
Bengaluru, India.
e-mail: tr.vinay@gmail.com
[2]Dept. of Computer Science, Nagarjuna College of Engineering and Technology,
Bengaluru, India
Email: ac.ajeet@gmail.com

Abstract—Hardware and Software technology has undergone a sea-of-change in recent past. Hardware technology has moved from single-core to multi-core machine, thus capable of executing multi-task at the same time. But traditional software's (Legacy system) are still in use today in business world. It is not easy to replace them with new software system as they carry loads of knowledge, business value with them. Also, to build new software system by taking the requirements afresh involves lot of resources in terms of skilled human resources, time and financial resources. At last the customer may not have confidence in this new software. Instead of building a new software, an attempt is made to develop a semi-automated methodology by learning about the program itself (machine learning about the program) to abstract the independent modules present in the same abstraction level (implementation level) and recode the legacy program (single threaded program) into multi-threaded parallel program. A case study program is considered and execution time is noted and analyzed for both the original program and reengineered program on a multi-core machine.

Keywords-Multicore Machines, Legacy Systems, Reverse Engineering, Multi-threads

I. INTRODUCTION

Legacy software's are the software's designed and implemented in the last decade but still in use in the business world. The software designers, architects, engineers and others involved in building these systems would have designed as a single threaded system executing all the tasks/modules sequentially. The programming language used in yester years would only support sequential execution on a single-core machine. But software and hardware development has undergone a sea change. It is very common now to have a multi-core machines on hardware side and from software side, programming languages supporting parallel executions doing multi-tasking at the same time. Thus, accelerating the rate at which programs will be executed.

The legacy software's which are still in use today, would have undergone many modifications, updating's as per the business requirements. Even though these software's are slow compared to modern software systems, they carry rich knowledge throughout their life-cycle and may have very high business value. In sense, customers/ clients using this legacy software's will not be ready to take risk in replacing them with modern system because of the business value they are associated with it. Also, if replaced by a modern system, these systems cannot guarantee the correctness and completeness of the legacy system and it may take time to get the confidence of the users. In this scenario, the alternative option is to migrate this legacy system onto modern hardware machines in the same abstraction level (implementation level). Rather than following forward engineering approach, Reverse engineering approach should be employed as shown in figure 1.

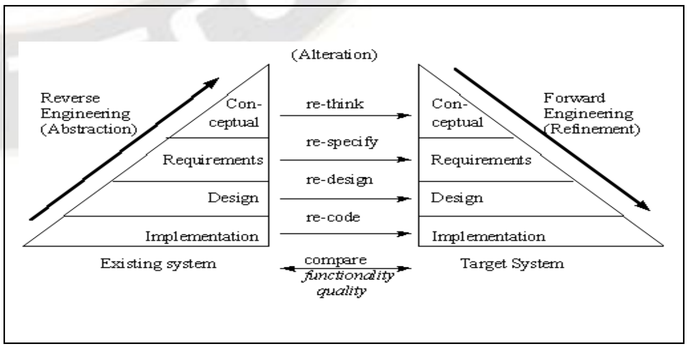

Figure 1: Forward Engineering and Reverse Engineering

In figure 1, the abstraction of modules from legacy system is done at the implementation level and re-coding it on to the multi-core system, there by converting a single threaded sequential program into multi-threaded parallel program [14].

To achieve this, the basis is to first understand the legacy program which contains hidden knowledge, rich statistical values as well [Hindle et al., 2012]. The present-day software engineers will find it difficult to understand legacy software implemented language. The programming language also has undergone a tremendous change. The associated documentation of legacy software's will also be of little use, as many modifications would have taken place in its life cycle and little importance would have given to documentation.
Analyzing the program and representing it in intermediate form [12] such as control flow graph, control flow table **[11]** and Data flow table [8,9] is done. Many researchers designed automated tools, machine learning algorithms to analyze the properties of code for finding bugs or security vulnerabilities, to design test-cases and compiler optimization. Here an algorithm is presented to abstract the modules of legacy system by analyzing the program and building aHypergraph **[14, 16]** of the program. There by identifying any dependent or independent modules present within the program. Thereafter these independent modules can be executed in-parallel on a multi-core machine using p-threads.
A standards-based thread API for C/C++ is provided by the POSIX thread libraries [15]. One is able to start a fresh concurrent process flow thanks to it. It works best on systems with several processors or cores since the process flow may be scheduled to occur on a different processor, resulting in faster parallel or distributed processing. The POSIX thread library is used in applications to accelerate program execution.The detailed step-by-step methodology is explained in section 3.

II. Review on Migration Techniques

[1] In this paper the author proposed to take risk analysis before taking up migration project and proposed a framework where the forward and reverse engineering are amalgamated as per the requirement. For this framework, detailed software requirement specification (SRS), associated documentation of the legacy system is required. Here the migration team first understands the legacy system through the study of SRS and documentation of the software analyzes the risk involved and plan accordingly for migration of tasks that has no risk. If any tasks are having high risk, the author proposes to design and build the task by following the forward engineering principle. The main lacuna observed here is that the legacy system should have proper well maintained, understandable documentation.

The aim of [2] was to migrate a legacy software written in obsolete programming language to modern newer programming language environment. There by utilizing the power of modern programming language paradigm. But for this, the skilled software migration team has to have in-depth knowledge of the older programming language and also modern programming environment. They need to understand the older programming constructs, what tasks it does, how it affects the other statements. It is very cumbersome process.

So, it was carried on a Legacy test program only. Many limitations, impediments were also noted down.
In paper [4], the author proposes a set of transformative rules for migrating older languages to newer programming language. The migration team has to understand line by line of the legacy software by scanning each line, reading-understanding the comment lines. For each sub-system, the author analyzed the program through constructing Control flow graph, data flow, its interfaces with rest of the system. For each data type, structures, unions a mapping table was developed and mapped all these to target languages program constructs. Here a case study software was considered and migrated into C/C++ code. The migration was carried out in two phases. This process is time consuming and getting skilled man power to work on this type of project poses a challenging task. As many software engineers bend always towards learning new technologies and working on them to improve their skill set. Motivating them to learn obsolete programming language and work on migration of legacy system itself is a challenging task.
In paper [6], the migration study was carried out for large scale software project. First data was migrated to new database technology and then the software was migrated. During the migration process, constant customer involvement was necessitated so that the customer can test, verify the migrated task. More time and resources were spent on testing and satisfying the customer itself.
A number of research authors worked on source code migration. In [17], a tool is designed to migrate Pascal program to C. In [18], Fortran to C and C++ converted is designed. In [20] a framework for translating smalltalk program into C is proposed. A number of translators from Ada-83 to Ada-91, CMS/Jovial/Fortran to Ada have been developed by Xinotech[19].

III. Methodology

A. Selecting a Template (Heading 2)

The input to our methodology is a legacy program running as a single threaded program. This program is pre-processed, abstracting the modules, examining their dependences and reengineer them into multi-threaded parallel program using p-threads. Afterwards, scheduling them on a multi-core machine. The entire process flow is depicted in the following figure 2.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5837
***Article Received**: 27 September 2022 **Revised**: 09 November 2022 **Accepted**: 28 November 2022*

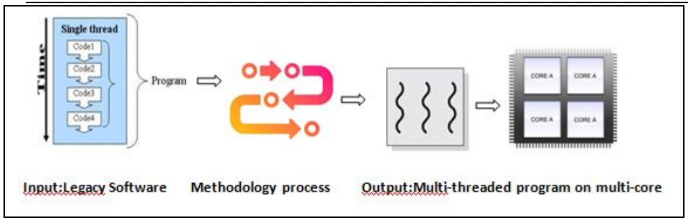

Figure 2: Process flow of proposed methodology

The methodology is expressed by means of an algorithm.
Algorithm: Conversion of Single to Multi-threaded parallel program.
Input: Single threaded legacy program
Output: Multi-threaded parallel program
Step 1: Pre-processing the legacy program for analysis;

The particular language semantics has to be understood first and then can be readjusted conveniently for further analysis. Different software programmers use different style of writing programs. Such as writing multiple statements in one line or writing single statement in more than one line. Here we fix one statement in one-line order. The others are removing blank lines and comment lines. Finally inserting the line numbers.

Lex regular expressions for removing comment lines:
/*Identifying single line comment in the given program*/
\/\/(.*) ;
/* Identifying multi line comment in the given program and removing them*/
{start}.*{end} ;

Step 2: Extract all variables set of the program. Naming the set as Vu = { }
Step 3: Classify them into three categories as set of input variables **Vi** = { }**,** set of output variables **Vo** = { } and set of intermediate variables as **Vim** = { }.
Step 4: Extracting the signatures of a function. The signature means the return variable of the function and function arguments. It is extracted by constructing the abstract syntax tree of the function call.
Step 5: Constructing table [14,16] of signatures of functions. In each row, a function name, its return variable and the arguments of the function is inserted.

TABLE 1: TABLE OF SIGNATURE-OF-FUNCTIONS.

Sl.No.	Function Name	Return Variable	Arguments list	Remarks
1.	Function-1			
	…			
N	Function-N			

Step 6: Analyzing the Signature-of-functions table**.**
If any one of the arguments of one function matches with the return variable of another function. Concluding that they are dependent on each other and need to execute them in program order by applying mutex on that variable.

Else, if non-matches, functions are independent.

Step 7: Identifying and counting the number of independent functions.
Step 8: Creating p-threads for all the independent functions and scheduling them on to execute on a multi-core machine.
END.

IV. CASE STUDY PROGRAM: SORTING AND SEARCHING PROGRAM

The following case-study program below will be treated using the aforementioned methodology.

```
#include <stdio.h>
int * Sort(int a[ ])
int Search(int a[], int key, int *index)
int main()
{
   int a[size], int key1, key2,key3, index1= -1,index2= -1,
index3=-1;
   // assigning values for array a, key1,key2 and key3
   a = Sort(a);
Search(a,key1, &index1);
Search(a,key2, &index2);
Search(a,key3, &index3);
   // printing the values of index values from three Search
functions.
   return 0;
}
```

Figure 3: Sample case study program.

By observation, the above program runs as a single threaded program executing Sort and Search function (calls Search function three times) sequentially. By applying the above methodology, the program can be sliced [2,3] and executed in-parallel by creating parallel threads. Thus, improvement in execution speed of the program can be noted.
Applying the methodology described in Section-3 onto the program given in Figure 3, the following intermediate outcomes are as follows:

// set of all variables
Vu = { a, key1, key2, key3, index1,index2, index3}
Vi = { a, key1,key2,key3}   // set of input variables
Vo = { index1,index2, index3 } // set of output variables

TABLE-2: SIGNATURE-OF-FUNCTIONS FOR THE GIVEN PROGRAM

Sl. No.	Function Name	Return Variable	Arguments list	Remarks
1.	Sort	a	a	Independent function
2.	Search	index1	a, key1	Dependent on function

ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5837
***Article Received**: 27 September 2022 **Revised**: 09 November 2022 **Accepted**: 28 November 2022*

				in row-1 and Independent of functions in row 3 and 4
3.	Search	index2	a, key2	Dependent on function in row-1 and Independent of functions in row 1 and 4
4.	Search	index3	a, key3	Dependent on function in row-1 and Independent of functions in row 2 and 3

From the above Table, the inference is that the sort function is independent and Search function is dependent on Sort function. The Search function which is called three times can be executed in-parallel by creating three parallel threads after the Sort function is executed.
The reengineered case study program is given below:

```
#include <stdio.h>
int * Sort(int a[ ])
int *Search(int a[], int key, index)
int main()
{
   int a[size], int key1, key2,key3, index1,index2, index3;
// assigning(reading) values for array a, key1,key2 and key3
pthread_t thread1, thread2, thread3;
int  iret1, iret2;

  a = Sort(a);
iret1 = pthread_create( &thread1, NULL,Search(a,key1,&index1));
iret2 = pthread_create( &thread2, NULL, Search(a,key2,&index2));
iret3 = pthread_create( &thread2, NULL, Search(a,key3,&index3));

pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
pthread_join( thread3, NULL);
 // printing the values of index values from three Search
functions.
   return 0;
}
```

Figure 4: The reengineering case study program of figure 3.

In the above program, first the Sort function is executed then, the program is sliced and three threads starts executing in-parallel, each one executes Search function independently and in-parallel. After their completion of execution, they are spliced again. The time taken to execute both the programs for different array size is recorded.

V. EXECUTION TIME ANALYSIS

Both the original program and the reengineered program is executed on Processor   Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, 2904 Mhz, 8 Core(s), 16 Logical Processor(s).
The size of the array is varied and the execution time in each case is noted as given in the below Table 3 and graph is plotted for the same and is seen below in figure-5.

TABLE 3: TIME TAKEN TO EXECUTE THE ORIGINAL PROGRAM AND THE REENGINEERED PARALLEL PROGRAM

Array size (n) is varied in each iteration	**Execution Time of Original Program in seconds**	**Execution time of Reengineered parallel threaded program in seconds**
1000	0.000323	0.27308
10000	0.005367	0.27581
20000	0.011803	0.27986
40000	0.023472	0.28263
60000	0.046380	0.28585
80000	0.102473	0.28934
100000	0.118250	0.31681
200000	0.478015	0.38903
300000	1.035308	0.80378
400000	1.792921	1.18351
500000	2.815044	1.90635

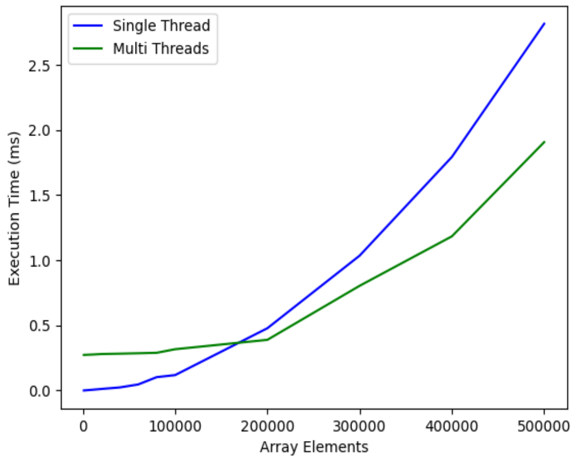

Figure 5: Execution time of the program by running it as a single thread and multi-thread program for varying array size.

Observation:

Initially the multi-threaded parallel program is taking more time to execute and as the size of the array increases more than one lakh in this case, this reengineered program is taking less time to execute compared to single threaded program. Ideally when a greater number of parallel threads is created on a multi-core machine, it should take less time proportionally to the number of threads created. But according to the Amdahl's law observation, it depends on the program itself, as how much part of the program can be made parallel. Thus, the performance of reengineered parallel-program depends on the considered

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5837
***Article Received**: 27 September 2022 **Revised**: 09 November 2022 **Accepted**: 28 November 2022*

program itself, the machine configuration such as number of cores, the synchronization required among the parallel threads, memory bounds, correct reimplementation of code (creating p-threads and assigning them to functions), time taken to create and terminate the p-threads, seek time to access the array elements (this comes into effect if array's size becomes too large) and the operating system itself.

VI. CONCLUSION

Legacy Software's running as a single threaded sequential program on a single-core machine is reengineered to run on multi-core machine as multi-threaded parallel program. The semi-automatic methodology employed above **is** carried out at the corresponding abstraction-level itself(Implementation / code), without modifying the design or following the forward engineering principle. Ideally the speed-up achieved by converting a single thread program into two-thread parallel program should be 50% less than the original program, or if it is converted into four threaded program, it should have taken 75% less time than the original program. But according to the experimental result, the speedup achieved by running the same program in-parallel is restricted by Amdahl's law, because the time taken for creation and termination of extra threads and multi-core machine itself will take extra time. Nevertheless, the process of migration [13] of legacy software's is better approach rather than discarding the legacy software and building newer software in its place.

REFERENCES

[1] Hausi A Miller , "Reverse engineering Strategies for Software Migration" ICSE , ACM Transaction.

[2] Paul R. Salopek, "Migration of Legacy Test Program to modern programming environment" IEEE Transaction, 2000.

[3] Ying Zou, " Incorporating Quality Requirements in Software Migration Process" , 11th Annual STEP-04

[4] Kostas Kontogiannis et. al.,"Code Migration through Transformations: An Experience Report", IBM CASCON 1998.

[5] Louis Forite, Charlotte Hug[5], "FASSM: Fast and Accessible Software Migration Method" Universite Paris, IEEE Transaction 2014.

[6] Werner Teppe[6], "ARNO Project: Challenges and Experiences in a Large-scale Industrial Software Migration Project", IEEE Transaction 2009.

[7] Andreas Menychtas[7] et al. "ARTIST Methodology and Framework: A approach for the migration of legacy software", IEEE Transaction 2014.

[8] Dr.ShivanandM.Handigund. Reverse Engineering of Legacy COBOL Systems. Doctoral dissertation, IIT, Bombay.

[9] Handigund S.M., Arunakumari B.N., Chikkamannur A. (2018) Automated Methodology to Streamline Business Information Flow Embedded in SRS. In: Sa P., Bakshi S., Hatzilygeroudis I., Sahoo M. (eds) Recent Findings in Intelligent Computing Techniques. Advances in Intelligent Systems and Computing, vol 709. Springer, Singapore. https://doi.org/10.1007/978-981-10-8633-5_33

[10] Pankaj Jalote. An Integrated Approach to Software Engineering, Third Edition, Narosa Publishing House.

[11] A. Chikkamannur and S. M. Handigund, "An ameliorated methodology to design normalized relations," 2009 IEEE/ACS International Conference on Computer Systems and Applications, Rabat, 2009, pp. 861-864, doi: 10.1109/AICCSA.2009.5069431.

[12] AAChikkamannur,SMHandigund,An ameliorated methodology for ranking the tuple, International Journal of Computers and Technology (IJCT) 14 (4), 5616-5620.

[13] Vinay T R and A. A. Chikkamannur, "A methodology for migration of software from single-core to multi-core machine," 2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, 2016, pp. 367-369, doi: 10.1109/CSITSS.2016.7779388.

[14] VinayT R and A.A.Chikkamannur, A semi automatic transformational technique for transforming single threaded program into multi threaded program , https://ijarcce.com/wpcontent/uploads/2022/01/IJARCCE.2021.101251.pdf

[15] https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html#:~:text=ThePOSIXthreadlibrariesare,throughparallelordistccributedprocessing.

[16] Vinay, T.R., Chikkamannur, A.A. (2022). A Novel Methodology to Restructure Legacy Application onto Micro-Service-Based Architecture System. In: Shetty, N.R., Patnaik, L.M., Nagaraj, H.C., Hamsavath, P.N., Nalini, N. (eds) Emerging Research in Computing, Information, Communication and Applications. Lecture Notes in Electrical Engineering, vol 790. Springer, Singapore. https://doi.org/10.1007/978-981-16-1342-5_39.

[17] Gillespie, D.," A Pascal To C Converter", The HP-UX Porting and Archive Center, http://hpux.u-aizu.ac.jp/hppd/hpux/ Languages/p2c-1.20/readme.html

[18] Feldman, S., Gay, D., Maimone, M., Schryer, N., \A Fortran to C Converter", AT&T Technical Report No. 149, 1993.

[19] Xinotech Inc. http://www.xinotech.com.

[20] Yasumatsu, K., Doi, N., \Spice: A System for Translating SmallTalk Programs Into a C Environment" IEEE Transactions on Software Engineering, vol. 21.