Exploitation of Task Level Parallelism

MAYANK MANGAL[1], ANKIT SANGHAVI[2], SANDEEP PARODKAR[3]

[1]Assistant Professor, Dept of IT, ARMIET, Thane, Mumbai, India,

[2]Assistant Professor, Dept of COMPUTER, ARMIET, Thane, Mumbai, India,

[3]Assistant Professor, Dept of EXTC, ARMIET, Thane, Mumbai, India

Abstract: Existing many systems were supporting task level parallelisms usually involving the process of task creation and synchronization. The synchronization of task requires the clear definition about existing dependencies in a program or data-flow restraints among functions(tasks), or data usable information of the tasks. This paper describes a method called Symbol-Table method which will used to exploits and detects the task level parallelism at inner level of sequential C-programs.

This method is made up of two levels: a normal symbol table and an extended symbol table. A sequential program of C language is the input to the normal gcc compiler in which the procedures are defines as functions(tasks).Than wegenerate a normal symbol table with specific command by gcc compiler in Linuxas an output.Then we use the information of that symbol table for generatingthe extended symbol table with additional information about variable's extended

scopes and inner level function dependency. This extended symbol table is generated by the use of previously generated normal symbol table on the basis ofvariable's scope and L-value/R-value attributes. By that table we can identify thefunctions and variables those who are sharing the common variables and those who are accessing the different functions with extended scopes respectively. Thenwe can generate the program dependency graph by the using of that extended

symbol table's information with a specific java program. A simple program forusing this method has been implemented on a 64-bits Linux based multiprocessor.Finally we can generate the Function graph for every variable in the programwith the help of table's info and dependency graph's states.From that graph wecan get the info about extended scoping of variables to identify and exploit thetask level parallelism in the program. Then we can apply the parallelism withMPI or other parallel platforms to get optimized and error free parallelism.

Index Terms: Task synchronization; Function dependencies; Symbol table; Program Dependency graph;

INTRODUCTION

Parallel programming is much more difficult and erroneous than sequential programming. So, more intension is required to confirm that the parallel source programs give the correct outcomes. Aerror-free parallel source program does not always compulsorily resultsin a good performance.So, to achieve the good performance and good efficiency,a parallel program must have load balance,low overhead and the good data locality. In general, parallel programs are much complex and difficult to maintainbecause they implement complex type of parallel behaviour algorithms, and hold platform-specific optimized source code.

The uninvited difficulties of parallel programming have energized research inthe range of parallelizing and rebuilding compilers. These parallelizing compilers consequently discover parallelism in successive projects and rebuild them intoparallel projects.

The lion's share of parallelizing compilers [1] [2] [3] have concentrated on parallelism inside loops, where this parallelism gives outcome fromexecuting free cycles of a loop in a parallel way. That is usually called as loop levelparallelism. In spite of the fact that these type of parallelizing compilers take outthe requirement for parallel programming and are for the most part compelling,later studies [4] [5] have indicated that they have some constraints, and that this parallelism is not sufficient to use the all resources of the parallel computers. So we finally came to task-level parallelism which provides a task(function)as a procedure invocation,a program block or an arithmetic operation. A fewprovisions are all the more characteristically communicated as an accumulation ofrelated tasks [6] [5]. Besides, for extensive provisions, it is important to exploitthe loop level parallelism and task level parallelisms [5]. On the other hand,not like as parallel compilers, existing frameworks that using task level parallelisminterest programming exertion, which extends from needing to physically make andsynchronize of tasks to needing to program in diverse dialects and ideal models.So because of this there is a requirement for frameworks that can exploit task level

parallelism with the sequential type of programs.

I. METHODDESCRIPTION

MethodDescription

This method is mainly composed of two levels: a normal symbol table and an extended symbol table. The input given to the normal gcc compiler is a sequentialC program in which the procedures taken as functions(tasks) with declarations.For generate a normal symbol table we provide a specific command to gcc compiler inlinux. Then we use the information of that symbol table for generatingthe extended symbol table with additional information about variable's extendedscopes, L/R-value attributes and inner level function dependency. This extendedsymbol table is generated by the use of previously generated normal symbol table on the basis of variable's scope, Declared line and referenced line of variables and functions with-in the program. By that table we can identify the functionsand variables those who are sharing the common variables and those who are accessing the different functions with extended scopes respectively. Then we cangenerate dependency graph based on that table's information to get the clear understandings about the dependencies which are exists in the program. Finallywe can generate the Function graph for every variable in the program with thehelp of table's info and

dependency graph's states.From that graph we can getthe info about extended scoping of variables to identify and exploit the task levelparallelism in the program. Then we can apply the parallelism with MPI or otherparallel platforms to get optimized and error free parallelism.

Types of Data dependence

This part describes about data dependence and its types. That dataabout dependence is covered in literature [7] [8] [9] and presented
here.

Data dependencies are occurs when two or more iterations, statements or operations of a loop cycle can be executed in parallel.Basically four types of data-dependencies are there:

1. True dependency: (also known as Flow Dependency) it occurs betweenthe two statements of a program, if the first statement write the data andthe second other statement read it later.

For example, in this program

St1: c = a + b
St2: d = c * 2

So from these statements it is clearly shown that here is a true dependencyexists between these statements St1 and St2, denoted as St1 St2, becauseSt1 writes var c and St2 reads var c.

2. Anti Dependency
3. Output Dependency
4. Input Dependency

So these types of dependencies can be overcomes by variable renaming technique[9]. The only actual dependency is true dependency.

Dependencies can be occur between parts or instances of statements in a loopcycle, when the same element(variable) of an array can accessed by two instancesof statements. If these havingdependencies related to the same loop cycle thenthe dependency is known as a loop independent dependency , and without synchronization they can be executed parallel in a concurrent way .

For example, in the program

```
for i = 0 to N-1
St1: p[i] = q[i] + r[i]
St2: s[i] = p[i] + 1
endfor
```

Here it is clearly shown that the dependency that is exist here among theinstances of St1 and St2 are loop independent dependency; so, without synchronization they can be executed parallel in a concurrent way. In such type of cases,that loop is called as a parallel loop.however, if the instances relate to different-different loop cycles,then that type of dependency is called as a loop carrieddependency, and without synchronization they cannot be executed parallel in a
concurrent way.

For example,

```
for i = 0 to N-1
St1: p[i] = q[i] + r[i]
St2: s[i] = p[i-1] + 1
endfor
```

So it is clearly shown that a loop carried true dependency is here between theSt1 in iteration i and the St2 in iteration i + 1 where i = 0 to N - 1.

Task level Parallelism

It is an another type of parallelism which is a group of co-operating tasks. A unitof computation as task or procedure which can be as coarse-grain as a procedureinvocation or as fine-grain as an arithmetic operation that executes more and morenumber of instructions. When the independent tasks are executing concurrently then This type of Parallelism occurs. Tasks which are executing concurrently arenot in limited range to operating same type of set of operations in comparison todata-parallelism.

The thought of data dependence could be reached out to tasks. In frameworkswhere tasks are indicated by the data stream stipulations around the tasks, whena task P generates some data worth which is needed by another task Q then itis said that task Q is dependent on

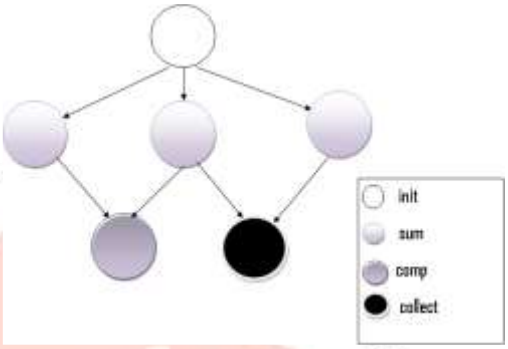

Figure1: An example type of task graph

task P. Subsequently, task Q can't begin itsexecution until task P has finished its execution about the data. In frameworkswhere the tasks are well synchronized to concurring a particular order, for example, the successive order of data-access of the tasks, at that point when task P goes inexecution before task Q in execution then it is also said that task Q is dependent ontask P, furthermore the same data is written by either P or Q. For that situation,task Q can't begin its execution until the task P has completed its execution of thedata.In both of the situations, we are referring the task P as the pre-requisite taskand task Q referred as the dependent type task on P. A graph i.e. which is used torepresent the dependencies among the tasks is called task graph. The task graphis a directed type acyclic graph in which the nodes are representing the tasks andthe edges are showing the dependencies between the tasks. The source and sinkof a dependency edge is a pre-requisite task and a dependent task respectively. Ingeneral, when a pre-requisite task completes its execution then only a dependenttask can execute.

For example, A task graph is represented in figure of a program. In this thetask is taken as a procedure invocation. The program execution would be followslike as. First of all Task 'init' will execute because of no pre-requisite tasks it has.After the completion of the task 'init' three 'comp' type tasks will be execute inparallel. Another task 'sum' will begin its execution when the first two 'comp' tasks will finish their executions, and the another task 'collect' will executes whenthe last two 'comp' tasks will finish their execution. So it is clearly shown inexample that the results of parallelism occurs from executing the 3 'comp' tasksin parallel, and the tasks 'sum' and 'collect' are executing concurrently.

Nonetheless, the adaptable behaviour of this task-level parallelism prompts afew drawbacks. As opposed to building parallelizing tools

concentrating on a particular programming developed loop, different languages or frameworks have beenmade to help diverse kinds of task-level parallelism. According to their meaning of the task the frameworks changes; according to the time on which tasksare made;according to the systems which is used to help in task synchronization
and communication between tasks; and additionally according to the programming languages and standards used to exploit the parallelism. Hence, to exploitthe task-level parallelism, a developer should first pick a framework, and either broadly adjust the sequential type of C-programs or modify them in the standardor programming languages needed by the framework.

II. Proposed method andimplementation

The proposed method of this research work motivated by the task level parallelism [16] to exploit the parallelism at inner level of functional dependencies in asequential c program. The idea of the proposed method developed here only toidentify the inner level dependencies of tasks(functions) statically at compile time.For exploiting the concurrency with Task level parallelism we have implementeda simple sequential c program with four functions on the Linux OS with gcc compiler. Here we are targeting the general workings of the compiler about symboltable creation.From that symbol table we are generating the Extended symbol table with additional information about variables and functions of program. Fromthat table we can identify the inner dependencies of functions in a program andby the functional graph we can show the relation between functions about thedependencies. We are using "NetBeans" tool with java platform for generatingthe dependency graph of program by using that Extended symbol table data injava program.

Exploitation of Task level parallelism with Symbol-Table method:

Our objective is "if 2 functions have common variable then they can not execute parallel(because they are accessing the same memory location of common variable). How these functions are accessing the same memory location?" For provingthis we are using symbol table analysis. According to our proposal there are three levels to solve that problem.

1. Generalized Symbol-table.
2. Extended Symbol-table.
3. Dependency and Function graph representation.

First we are taking a simple sequential c program as input to the gcc compiler withthe Linux OS.Then with the specific commands
(i.e. gccfilename.c -o filename
gcc -c filename.c
readelf -a filename.o)
We can generate internal process of execution and compilation of program(ExecutableLocate File(ELF)). It has all information about program like ELF Headers, section headers, section groups, program headers, key to flags, relocation sections,unwind section and most important "Symbol-table".
From that we can generatethe generalized Symbol-table with information like:
1. Name of Variables and Functions
2. Characteristics Class
3. Token id
4. Scope of Variable and Function
5. Declared line
6. Referenced line
7. Other information like parameters used by functions.

Then for getting the Scope information we generate the Symbol-table for everyfunction which is present in the program.After generating the generalized Symbol-table with all this information we can generate the second level of this method which is called "Extended Symbol-table"with the additional information about extended scopes of variables and the L-value/R-value attributes of the variables as output with the help of previouslygenerated generalized Symbol-table. In that Symbol-table clear information isprovided about extended scopes of the variables and L-value/R-value attributesof the variables. For "Extended Symbol-table" the attributes are:
1. Name of Variables and Functions.
2. Scope of Variables and Functions.
3. Declared Line of Variables and Functions in program.
4. Referenced Line of Variables and Functions.
5. Extended Scope of Variables and Functions.
6. L/R value attributes.

On the basis of Declared line and Referenced line we can draw the dependencygraph with the help of java based program of NetBeans tool. And on the basisof Extended scope and L-value attribute we can represent the function graphsmanually regarding the information about variables those are changing their scopeswith L-value attribute. From these graphs we can easily identify the dependenciesamong functions and which functions can execute parallel to each other.

Design and implementation of the proposed method:

Our proposed method is based on Symbol-table design and implementation. HereSymbol-table designing process is divided in to mainly two parts.

- Generalized Symbol-table.
- Extended Symbol-table.

So for designing and implementing using this Symbol-table analysis method wehave taken a simple and small sequential C-program. The source code of thatprogram is given here:

Program source code

```
#include <stdio.h>
#include<conio.h>
int g = 10;
int I = 15;
Int *h;
int add(int a, int b)
{
        a = a + b;
        g = a + b;
}
int sub(int a, int b)
{
        a = g - b;
        g = *h - b;
}
intmul(int *a, int *b)
{
        *a = *a * *b;
        *b = *h * *a;
}
int div(int *a, int *b)
```

```
{
        *a = *a / *b;
        *h = *h / *b;
}
int main ()
{
printf(in main1 I m = %d %u, g,&g);
Int x = 10;
int y = 5;
int sum = 0, minus = 0;
```

It is a 40 line simple source program which has mainly four functions i.e. add(),Sub(), mul(), div(). These functions are sharing some common variables.Thevariables are used in this program are g,h,i,a,b,x and y in which g,h and i areglobal variables and rest are local variables of different functions. Because of thedata inconsistency these functions can't execute with parallel platform. So, For finding the inner dependencies among the functions we are using this Symbol-tablemethod.

So first we generate the Symbol-table by gcc compiler with specific commandson Linux platform. Than from that table we can generate the "GeneralizedSymbol-table" using unordered-listed data structure on the basis of some important attributes like Name, Char class, Token id, Scope, Declared line, ReferencedLine and other info about variables and functions in the program.

We can generate a separate Symbol-table for each scope(function) which isused in the source program.The "Generalized Symbol-table" for that aboveprogram is shown in table 1, 2, 3, 4, 5:

Table 1: Generalized Symbol-table with Main Function

Name	Char class	Token id	Scope	Dec. line	Ref line	others
g	var int	*id* < 1 >	0(global)	3	9,13,14,28,38	-
i	var int	*id* < 2 >	0(global)	4	5,14,19,24,38	-
h	ptr var int	*id* < 3 >	0(global)	5	14,19,24	-
add	func. int	-	1	6-10	34	two parameters a,b
sub	func. int	-	2	11-15	35	two parameters a,b
mul	func. int	-	3	16-20	36	two parameters a,b
div	func. int	-	4	21-25	37	two parameters a,b
main	func. int	-	5	26-40	-	-
x	var int	*id* < 12 >	5	29	34,35,36,37,38	-
y	var int	*id* < 13 >	5	30	34,35,36,37	-
sum	var int	*id* < 14 >	5	31	34	-
minus	var int	*id* < 15 >	5	31	35	-
multi	var int	*id* < 16 >	5	32	36	-
div	var int	*id* < 17 >	5	32	37	-

Table 2: Symbol-table for add() Function

Name	Char class	Token id	Scope	Dec. line	Ref line	others
a	var int	*id* < 4 >	1(local)	6	8,9	-
b	var int	*id* < 5 >	1(local)	6	8,9	-

Table 3: Symbol-table for sub() Function

Name	Char class	Token id	Scope	Dec. line	Ref line	others
a	var int	*id* < 6 >	2(local)	11	13	-
b	var int	*id* < 7 >	2(local)	11	13,14	-

Table 4: Symbol-table for mul() Function

Name	Char class	Token id	Scope	Dec. line	Ref line	others
a	var int	*id* < 8 >	3(local)	16	18,19	-
b	var int	*id* < 9 >	3(local)	16	18,19	-

Table 5: Symbol-table for div() Function

```
int multi = 0, divide = 0;
h = &I;
sum = add(x,y);
minus = sub(x,y);
multi = mul(&x , &y);
divide = div(&x , &y);
printf(in main2 I m = %d %u %d %d %u, g,&g,x,I,&i);
return 0;
}
```

Name	Char class	Token id	Scope	Dec. line	Ref line	others
a	var int	*id* < 10 >	4(local)	21	23	-
b	var int	*id* < 11 >	4(local)	21	23,24	-

This is a Generalized Symbol-table for above program which contains the information about variables and functions like name,type,token-id,scope,declared linein program, referenced line in program etc. Now on the basis of scope,declared lineand referenced line we can generate the "Extended Symbol-table" with additional information about variables like extended scopes and L/R-values attributes.

The Extended Symbol-table over the Generalized Symbol-table is shown in table6. From this table we can get the additional info about variables like Extendedscopes and L/R-value on the basis of reference lines. We can get the commonvariables from this table those are shared by functions.The variables those haveExtended scopes with L-value attributes are comes in focus only.

So with the help of this additional information we can go to generate for innerdependency graph and Function graphs as result.

Table 6: Extended Symbol-table for whole program

Name	Scope	Dec. line	Ref. line	Extended scope	Value(L-R)
g	0(global)	3	9	1	L-value
g	0	3	13	2	R-value
g	0	3	14	2	L-value
g	0	3	28	5	-
g	0	3	38	5	-
i	0(global)	4	5	local	-
i	0	4	14	2	R-value
i	0	4	19	3	R-value
i	0	4	24	4	L-value
i	0	4	38	5	-
h	0(global)	5	14	2	R-value
h	0	5	19	3	R-value
h	0	5	24	4	L-value
add	1	6-10	34	5	-
a	1(local)	6	8,9	local	L-val,R-val
b	1(local)	6	8,9	local	R-val,R-val
sub	2	11-15	35	5	-
a	2(local)	11	13	local	L-value
b	2(local)	11	13,14	local	R-val,R-val
mul	3	16-20	36	5	-
a	3(local)	16	18,19	local	L-val,R-val
b	3(local)	16	18,19	local	R-val,L-val
div	4	21-25	37	5	-
a	4(local)	21	23	local	L-value
b	4(local)	21	23,24	local	R-val,R-val
main	5	26-40	-	local	-
x	5	29	34	local	-
x	5	29	35	local	-
x	5	29	36	3	L-val,R-val
x	5	29	37	4	L-value
x	5	29	38	local	-
y	5	30	34	local	-
y	5	30	35	local	-
y	5	30	36	3	R-val,L-val
y	5	30	37	4	R-val,R-val
sum	5	31	34	local	-
minus	5	31	35	local	-
multi	5	32	36	local	-
divide	5	32	37	local	-

Results:

As we have completed both parts of this Symbol-table analysis method.Now we cangenerate the inner-Dependency graph with the help of both tables with Declared line and Reference line. The figure 2 shows the dependency graph on the basis ofdeclared and referenced line. This dependency graph is generated by the NetBeanstool which is a java based platform tool. With the help of a java program we arecreating a .txt file. Then we are putting the information about declared line,referenced line and number of lines from the generated tables into the .txt file.Than we can execute that java program to get the inner-dependency graph. In thisgraph the nodes or circles represent the line number at which the variable declaredor referenced in the program. And the edges represent the data dependencies fromdeclared line node to referenced line node.

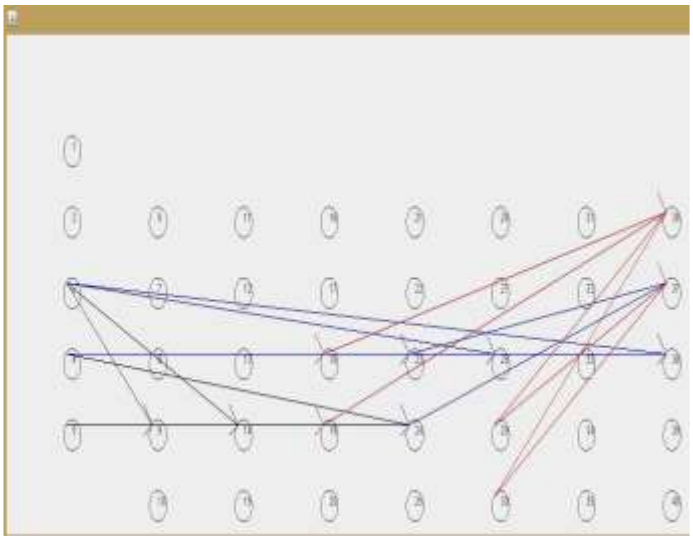

Figure 2: Dependency graph based on Table's info

In this Dependency graph, black color edges representing Data dependenciesbetween nodes(lines) and blue color edges representing temporary dependenciesand red color edges representing the referential dependencies. With the help ofthis dependency graph and Extended Symbol-table's additional information wecan generate the "Function graph" finally.

Function graph:

On the basis of Dependency graph and Extended Symbol-table's information likeL-value attribute(only) and extended scope of variables through referenced linewe can get the data to draw the Function graph.

The data is like that:

- g{0,3}[{1,9}{2,14}{5,(28,38)}]
- i{0,4}[{4,24}{5,38}]
- h{0,s} [{4,24}]
- x{5,29} →[{3,36,18}{4,37,23}]
- x{5,30} →[{3,36,19}{4,37,(23,24)}]
-

The expressions is written as like: the description of first expression data is:

Here, the left hand side data of an arrow represents the info about variable, forwhich we have to generate the Function graph.The info is, g is a variable which is defined in scope 0 and declared at line 3 in the program.And the right hand sidedata of an arrow represents the info about variable g that what are the extendedscopes of g with-in it is used.i.e. in scope 1 on line 9 (which is referenced line ofg) variable g is used.like that it is all the data about var g.Like that for variables h,i,x and y that data is given by expressions. on thebasis of that data we can generate the graph. From these graphs we can concludesome assumptions about exploit the task level parallelism that:

- 1 to 1 scoping is allowed between scopes.
- self-loop is allowed(local scoping).
- 1 to many scoping is not allowed.

Figure 3 shows the Function graph for var g. The Function graph clearly shows

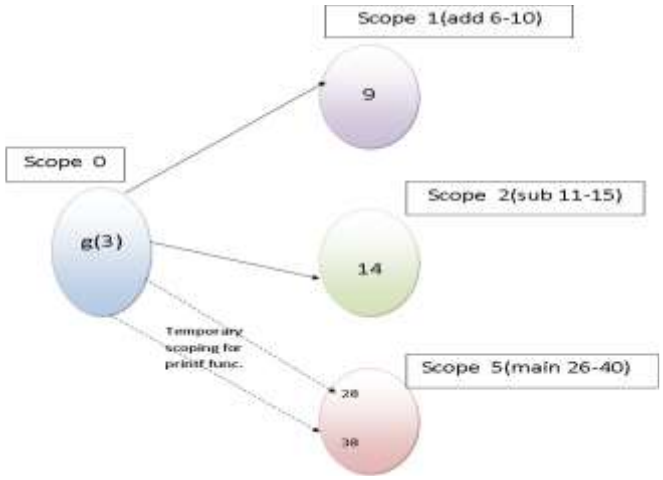

Figure 3: Function graph of g var.

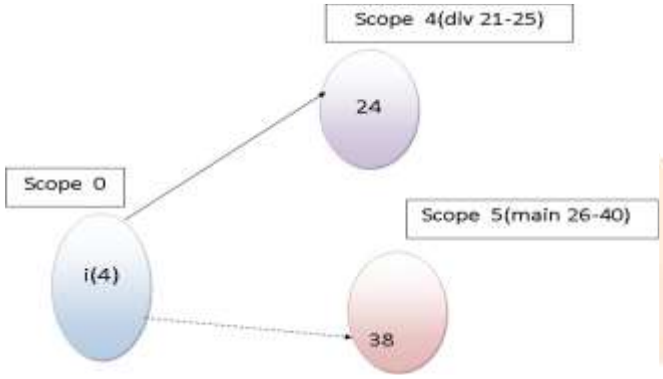

Figure 4: Function graph of var i.

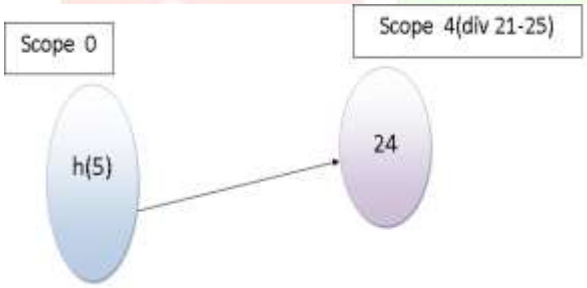

Figure 5: Function graph of var h.

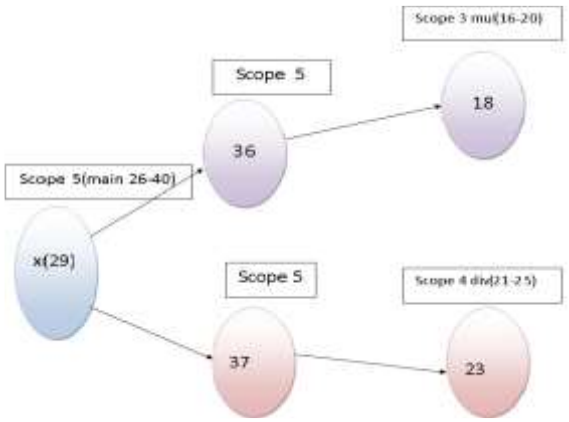

Figure 6: Function graph of var x.

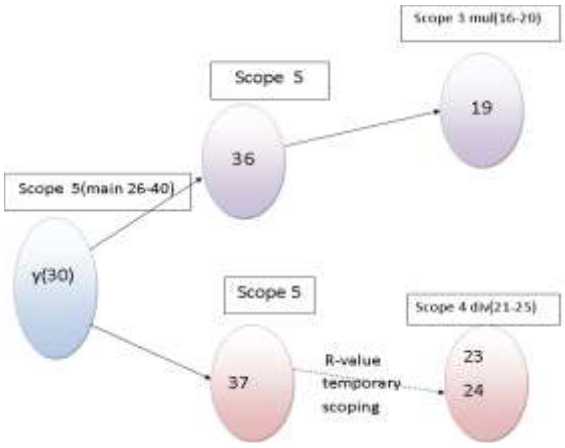

Figure 7: Function graph of var y.

From figure 4 which is showing Function graph for var i, we can evaluatethat var i is used by only scope 4 (div() function) with actual scoping(Extendedscope + L-value attribute only).
By scope 5 it is used as temporary scoping. So there are no problem with var i. Likewise figure 5 showing function graph forvar h, in which it is clearly shown that var h has only one actual scoping in scope4. According to assumption 1 to 1 scoping is allowed between scopes so there areno problem with var h.
Figure 6 showing the Function graph for var x, in which because of referencing dependency which is identified by dependency graph var x is referencedthrough scope 5 to scope 3.by graph it is clear that there 1 to many actualscoping is present between scopes for var x. So these specific scoping-functions aresharing common var x, thats by they can't execute parallel to each other. Likewise figure 7 showing function graph for var y, in which it is clearly shown that var y has 1 to 1 actual scoping in scope 3. So there are no problem with variable y.
After analysis of all these graphs finally we can say that:

- add() function and sub() function can't be execute parallel because of varg's 1 to many actual scoping.
- mul() function and div() function also can't be execute parallel because ofvar x's 1 to many referencing actual scoping.
- add() and mul() or add() and div() may be execute parallel because thereare no dependency or scoping conict between these functions.
- Also sub() and mul() or sub() and div() may be execute parallel becausethere are no dependency or scoping conict between these functions.

III. CONCLUSION:

The Loop level parallelism had some limitations which are solved by the task levelparallelism. After all, with the currently existing frameworks also those supportthis type of TLP, a system programmer must make the executable updates to thesequential C source program to achieve the required level of task parallelism.Inthis thesis this work has been done using a method is called Symbol-table method
at the time of compilation. This method has basically two different parts i.e.Generalized Symbol-table generation and Extended Symbol-table generation.Bythese parts of the method we can get the additional information like referenceline, declared line, scope, extended scope and L/R-value attributes about thevariables and functions which is used in the source program. With the help of this

information first we can draw the program dependency graph and after thatwith whole information about variables we can generate the Function graph foreach variable.From that graph we can clearly evaluate the inner level dependenciesamong the functions and extended scoping information about variables. On thebasis of that information we can detect and exploit the task level parallelism.Then we can apply the parallelism with MPI or other parallel platforms to get optimizedand error free parallelism.

LIMITATIONS AND FUTURE SCOPE:

This method has some limitations because of the quality of available compiler'sanalysis, data dependencies that occurs in the sequential C-programs and thechoice of only functions as tasks as units of parallelism.The Symbol table methodwe are using to get information about variables and functions, is generated manually. So the correct exploitation of parallelism depends on the correctness of thesymbol table, which totally depends on the programmer. So we suggest somefuture work for Symbol-table method to exploit the Task Level parallelism in general. To generate the generalized and extended symbol tables with correct and alladditional information for program we have to improve the quality of compiler'sanalysis.
The next one is that if we want to implement an analysis module that
provides side-effects of the statements in a program. Then, in place of providing awide set of features like as side-effect analysis,code generation and manipulationand dependency analysis, this will concentrate on the information that would helpsystems to support task level parallelism. This module should be able to accurately describe the accesses of Symbol-table to dynamic data structures such as
linked-lists and trees.To identify the recursive procedure's data accesses is also achallenge.

REFERENCES

[1] K. D. Cooper, M. W. Hall, R. T. Hood, K. Kennedy, K. S. McKinley, J. M. Mellor-Crummey, L. Torczon, and S. K. Warren, “The parascope parallelprogramming environment," Proceedings of the IEEE, vol. 81, no. 2, pp. 244-263, 1993.
[2] S. Hiranandani, K. Kennedy, and C.-W. Tseng, “Compiler optimizations forfortran d on mimd distributed-memory machines," in Proceedings of the 1991ACM/IEEE conference on Supercomputing, pp. 86-100, ACM, 1991.
[3] C. D. Polychronopoulos, M. B. Gikar, M. R. Haghighat, C. L. Lee, B. P. Leung, and D. A. Schouten, “The structure of parafrase-2: An advancedparallelizing compiler for c and fortran," in Selected papers of the secondworkshop on Languages and compilers for parallel computing, pp. 423-453,Pitman Publishing, 1990.
[4] R. Eigenmann and W. Blume, “An effectiveness study of parallelizing compiler," in Proceedings 20th International Conference Parallel Processing 1991,vol. 2, p. 17, CRC Press, 1991.
[5] J. Subhlok, J. M. Stichnoth, D. R. O'hallaron, and T. Gross, “Exploitingtask and data parallelism on a multicomputer," in ACM SIGPLAN Notices,vol. 28, pp. 13{22, ACM, 1993.
[6] H. Printz, H. Kung, T. Mummert, and P. Scherer, “Automatic mapping oflarge signal processing systems to a parallel machine," in 33rd Annual Techincal Symposium, pp. 2-16, International Society for Optics and Photonics,1989.
[7] U. K. Banerjee, Dependence analysis for supercomputing. Kluwer AcademicPublishers, 1988.
[8] G. Goff, K. Kennedy, and C.-W. Tseng, Practical dependence testing, vol. 26.ACM, 1991.
[9] D. A. Padua and M. J. Wolfe, “Advanced compiler optimizations for super-computers," Communications of the ACM, vol. 29, no. 12, pp. 1184-1201,1986.
[10] S. F. Hummel, E. Schonberg, and L. E. Flynn, “Factoring: A method forscheduling parallel loops," Communications of the ACM, vol. 35, no. 8, pp. 90-101, 1992.
[11] M. Chandy, K. Kennedy, C. Koelbel, C.-W. Tseng, et al., “Integrated supportfor task and data parallelism," International Journal of High PerformanceComputing Applications, vol. 8, no. 2, pp. 80-98, 1994.
[12] J. J. Dongarra and D. C. Sorensen, “A portable environment for developingparallel fortran programs," Parallel Computing, vol. 5, no. 1, pp. 175-186,1987.
[13] P. A. Suhler, J. Biswas, K. M. Korner, and J. C. Browne, “Tdfl: A task-leveldata flow language," Journal of Parallel and Distributed Computing, vol. 9,no. 2, pp. 103{115, 1990.
[14] R. G. Babb II and D. C. DiNucci, “Scientific parallel processing with lgdf2," inProceedings of the third SIAM Conference on Parallel Processing for ScientificComputing, pp. 307-311, Society for Industrial and Applied Mathematics, 1987.
[15] R. Chandra, A. Gupta, and J. L. Hennessy, COOL: A language for parallelprogramming. Computer Systems Laboratory, Stanford University, 1989.
[16] S. Huynh, Exploiting task-level parallelism automatically using pTask. University of Toronto, 1996.
[17] D. Scales, M. Rinard, M. Lam, and J. Anderson, “Hierarchical concurrencyin jade," in Languages and Compilers for Parallel Computing, pp. 50-64,Springer, 1992.
[18] M. S. Lam and M. C. Rinard, “Coarse-grain parallel programming in jade,"in ACM SIGPLAN Notices, vol. 26, pp. 94-105, ACM, 1991.
[19] M.-Y. Wu and D. D. Gajski, “A programming aid for hypercube architectures," The journal of Supercomputing, vol. 2, no. 3, pp. 349-372, 1988.
[20] T. Yang and A. Gerasoulis, “Pyrros: static task scheduling and code generation for message passing multiprocessors," in Proceedings of the 6th international conference on Supercomputing, pp. 428{437, ACM, 1992.
[21] T. Gross, D. R. O'Hallaron, and J. Subhlok, “Task parallelism in a highperformance fortran framework," IEEE Concurrency, vol. 2, no. 3, pp. 16-26, 1994.

Author's Profile:

Mr. MayankMangal, received the Master OfTechnology degree in ComputerScience &Engineeringwith Software Engineering Specialization from the NIT Rourkela, he received the Bachelor Of Technology degreefrom RCERT Jaipur. He is currently working as Assistant Professor

in Information Technology Department at ARMIET, A. S. Rao Nagar, Shahapur, Thane, Mumbai.

Mr. AnkitSanghavi, received the Master OfTechnology degree in Computer Engineeringf rom the Mumbai University, he received the Bachelor Of Engineering degreefrom SantGadge Baba Amravati University. He is currently working as Assistant Professor in Computer Department at ARMIET, A. S. Rao Nagar, Shahapur, Thane, Mumbai.

Mr. SandeepParodkar, received the Master OfTechnology degree in EXTC Engineering f rom the Mumbai University, he received the Bachelor Of Engineering degreefrom North Maharashtra University. He is currently working as Assistant Professor in EXTC Department at ARMIET, A. S. Rao Nagar, Shahapur, Thane, Mumbai.