San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

1-1-2009

Analysis of an OpenMP Program for Race Detection

Dhaval Shah
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Shah, Dhaval, "Analysis of an OpenMP Program for Race Detection" (2009). *Master's Projects*. Paper 44.

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Analysis of an OpenMP Program for Race Detection

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Dhaval Shah

Fall 2009

© 2009

Dhaval Shah

ALL RIGHTS RESERVED

ABSTRACT

The race condition in a shared memory parallel program is subtle and harder to find than in a sequential program. The race conditions cause non-deterministic and unexpected results from the program. It should be avoided in the parallel region of OpenMP programs. The proposed OpenMP Race Avoidance Tool statically analyzes the parallel region. It gives alerts regarding possible data races in that parallel region. The proposed tool has the capability to analyze the basic frequently occurring non-nested 'for loop(s)'. We are comparing the results of the proposed tool with the commercially available static analysis tool named Intel Parallel Lint and the dynamic analysis tool named Intel Thread Checker for race detection in OpenMP program. The proposed tool detects race conditions in the 'critical' region that have not been detected by existing analysis tools. The proposed tool also detects the race conditions for the 'atomic', 'parallel', 'master', 'single' and 'barrier' constructs. The OpenMP beginner programmers can use this tool to understand how to create a shared-memory parallel program.

ACKNOWLEDGEMENTS

First and foremost I am thankful to my advisor, Dr. Robert Chun, for his constant guidance and support. When I started working on this project, I was beginner for this subject. Dr. Robert Chun provided me enough time to understand the subject. He provided me invaluable guidance and help. He always encouraged me to keep researching on this subject. Also, while research was going on with this project, he gave me very helpful suggestions. He helped me to finish this project on time. Dr. Chun is very polite and courteous. He sits until late night to help students. That shows his dedication to the students. I would like to thank him for his constant confidence in me.

I am thankful to Dr Mark Stamp and Dr. Chris Pollett for their participation in my thesis committee. The three of you are resourceful, polite, and helpful professors of the computer science department. You are assets to the computer science society.

I also appreciate help from my friends Jignesh Borisa, Sri Harsha Allamraju and Parin Shah. They provided me enough advice whenever I required.

Finally, I am grateful to my family for their constant support and confidence in me.

Thank you.

Table of Contents

1. Introduction.....1
1.1 Problem Addressed……..……..……3
2. Introduction to OpenMP…….……..…..4
2.1 OpenMP Programming Model…….……..…...4
2.2 OpenMP Directives…….……,…….5
2.2.1 'parallel' Construct…….…,……..5
2.2.2 Loop Construct…….…,……6
2.2.3 Work Sharing Construct……..…..,…….6
2.2.3.1 'single' Construct…….……,……6
2.2.4 Synchronization Construct……..……,…….7
2.2.4.1 'master' Construct…….…….…...7
2.2.4.1 'barrier' Construct…….…7
2.2.4.1 'critical' Construct…….……8
2.2.4.1 'atomic' Construct…….…….……...8
3. Related Work.....9
3.1 Race Detection Techniques 9
3.2 Static Analysis….....9
3.2.1 Static Analysis Techniques……..… 9
3.3 Dynamic Analysis.....12
4. Analysis Tools…….……..14
4.1 Intel Parellel Lint…….14
4.2 Intel Thread Checker…….……..16
4.2.1 Intel Thread Checker Architecture Overview …….……16
4.2.2 Intel Thread Checker Implementation Phases…….……17
4.2.2.1 Instrumentation…….……17
4.2.2.2 Analysis……..……...18
4.2.2.2.1 Segment and Partial Order Graph……..……18
4.2.2.2.2 Vector Clocks……..……...19
4.2.2.2.3 Shadow Memory……..……..20
4.2.2.2.4 Data Race Detection……..…….20
5. The Proposed Methodology….……..……21
6. Design……..……...23
6.1 Code Parsing…….……23
6.2 Diagnosis Process…….……26
6.3 Example Problem…….……30
7. Results……..……..33
7.1 Test Case 1 Assignment of Constant to Shared Variable by Parallel Threads...33
7.2 Test Case 2 Output Statement Synchronization Test ……...……..34
7.3 Test Case 3 Critical Region Test ……..……..35
7.4 Test Case 4 Atomic Region Test ..……..36
7.5 Test Case 5 Dependency Test in Critical Region…….…..……37

7.6 Test Case 6 Dependency Test in Critical Region ……..38
7.7 Overall Statistics……..39
8. Conclusion…….40
9. Future Work……..….41

Appendices
Appendix A. Source Code……42

References……...….106

List of Tables

Table 1. Type of Statements Classified……..24
Table 2. Results Statistics……....39

List of Figures

Figure 1. Fork-Join model……...4
Figure 2. Syntax for Parallel Construct……..….5
Figure 3. Syntax for Loop Construct……...…6
Figure 4. Syntax for Single Construct……...….7
Figure 5. Syntax for Master Construct…….7
Figure 6. Syntax for Barrier Construct…….7
Figure 7. Syntax for Critical Construct…….....8
Figure 8. Syntax for Atomic Construct……..…..…8
Figure 9. Drawbacks of Parallel lint……..….15
Figure 10. Intel Thread Checker Architecture…….....16
Figure 11. Directed acyclic graph representing whole program execution……..……...21
Figure 12. Example to Describe Proposed Race Detection Methodology……..……30
Figure 13. Test Case-1 Assignment of Constant to Shared Variable by Parallel Threads..33
Figure 14. Test Case-2 Output Statement Synchronization Test ……34
Figure 15. Test Case-3 Critical Region Test …….……..……35
Figure 16. Test Case-4 Atomic Region Test ……..…36
Figure 17. Test Case-5 Dependency Test in Critical Region ……..……37
Figure 18. Test Case-6 Dependency Test in Critical Region ……..……38

1. INTRODUCTION

Recently, multicore processors are being used extensively. Programmers can utilize these multicore processors by developing parallel programs. There are many parallel programming technologies available to support parallelism. OpenMP is an API which explicitely provides multi-threaded, shared memory parallelism [12].

Two or more threads run simultaneously in a multithreaded program. These threads communicate with each other using synchronization calls. If two or more threads try to access the same memory location without any interfacing synchronization calls, then a race condition can occur. Due to the non-deterministic behavior of the multithreaded programs, data races are considered as programming errors which are very difficult to find and debug. Even if we run the program with the same inputs, data races are difficult to reproduce. Data races do not crash the program immediately but they can corrupt the existing data structures. Data races may even cause system failures in unrelated code. Automatic race detection is a high priority research problem for the shared memory multithreaded programs. [21]

OpenMP is an Application Program Interface (API) for multi-threaded shared memory parallelism. OpenMP directives parallelize a part of the program. This part of the program is distributed across multiple threads. The variables in the parallel region are declared as private to each thread or shared between the parallel running threads. It is very important to identify which variables should be private and which variables should be shared. If all the variables in the parallelized code are defined as private then each thread has its own copy of the variables. This approach avoids the race conditions but it wastes memory. If a parallelized program uses a huge set of variables, array or any other complicated data structure, then it will occupy a very large amount of memory space. This kind of approach results in insufficient memory errors. In other

cases, if each variable in a parallel region is defined as shared, each thread shares the same copy of the variable. This approach utilizes less memory but it can result in data race conditions. [4]

In OpenMP, the parallelized program cannot be executed in the same order as sequential program. If we parallelize each iteration of a 'for loop', each thread would distribute iterations of the loop. All the iterations would be executed once. But the certainty of any iteration which is executed by a thread is not fixed. Also, iterations would not be executed in sequential order. If one shared variable is changed in current iteration and the next iteration uses the same changed variable then a race condition can occur because of uncertain order of execution of iterations in the OpenMP program. The main causes for data race conditon in an OpenMP program are missing 'private' clauses and missing synchronization constructs like 'barrier', 'single', 'master' and 'critical'. [4]

There are two approaches for analysis of a parallel program for race detection. One is dynamic analysis and another is static analysis. Some of the tools use a combination of static and dynamic analysis to detect the race conditions.

Intel thread checker and Sun Thread Analyzer are dynamic analysis tools to detect race conditions in OpenMP programs. These tools dynamically determine data race conditons in an execution of an OpenMP program using their projection technology. [3] Intel Parallel Lint and VivaMP are other OpenMP tools which use static analysis to avoid race conditions in an OpenMP program. [20]

1.1 Problem Addressed

The existing static analysis tool named Intel Parallel lint and the dynamic analysis tool named Intel Thread Checker generate many false positives. Also, both the tools do not give the right diagnosis for the code block inside the critical region.

This paper describes a technique to avoid race condition in OpenMP programs. The idea is to create a tool named Race Avoidance Tool for an OpenMP program. This tool analyzes a parallel region of an OpenMP program and generates the possible race condition alerts in that parallel region. The proposed tool uses a static analysis technique to find out data race conditions in the OpenMP program. Perl is used to create this tool.

The proposed tool is taking care of OpenMP parallel constructs like 'parallel' and 'parallel for', synchronization constructs like 'single', 'master', 'barrier', 'critical' and 'atomic'. This tool will be useful to the OpenMP programmer who has just started using OpenMP to utilize a shared memory parallelism.

The goal is to avoid data race conditions in the parallel region of an OpenMP program and to utilize the multicore processors by using a multithreaded shared memory parallel program.

2. INTRODUCTION TO OpenMP

OpenMP stands for **Open Multi-Processing.** OpenMP is an application program interface by which a programmer can explicitly achieve multi-threaded, shared memory parallelism. This API is specified for C, C++ and FORTRAN under Unix/Linux and Windows NT platforms. [12]

2.1 OpenMP Programming Model

OpenMP supports shared memory, thread based parallelism. Multiple threads consist of a shared memory process. The programmer can explicitly achieve parallelism using OpenMP. Programmers specify compiler directives in the code where they want to achieve parallelism[12].

The fork-join model has been used for parallel execution in OpenMP. An OpenMP

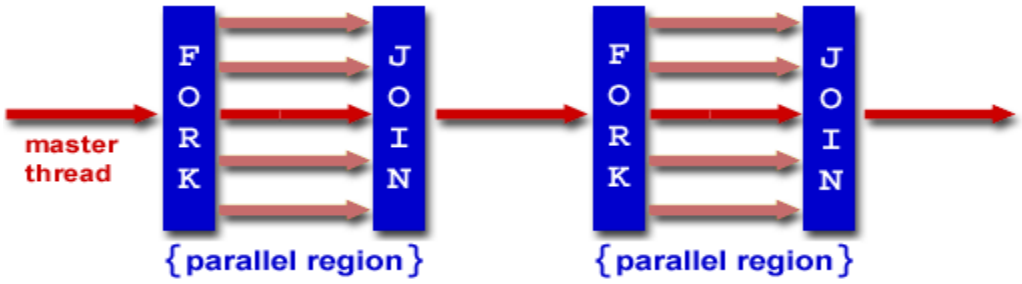

Figure 1 Fork-Join model[12]

program begins execution as a single master thread. When the parallel construct is encountered in the program, the master thread creates a team of parallel threads. Now the statements in the parallel region are executed in parallel among the team of threads. When the team of threads reaches the end of the parallel region, they are synchronized and terminated. There after only the master thread continues execution. A 'relaxed-consistency' memory model is used by OpenMP. Each thread caches its data and does not require maintaining exact consistency with real memory all the time. [12]

2.2 OpenMP Directives

List of directives have been explained which are taken into consideration in the proposed tool.

2.2.1 'parallel' Construct

The 'parallel' construct is a fundamental construct that initiates a parallel execution. Syntax for the construct is given below:

#pragma omp parallel *[clause[[,]clause] ...] new-line*
structured-block

where clause is one of the following:

if(*scalar-expression***)**
num_threads(*integer-expression***)**
default(shared | none)
private(*list***)**
firstprivate(*list***)**
shared(*list***)**
copyin(*list***)**
reduction(*operator***:** *list***)**

Figure 2 Syntax for Parallel Construct [13]

Here, the proposed tool only considers 'private' and 'shared' clauses for analysis purpose. 'private' and 'shared' clauses describe the list of variables defined as private and shared respectively in 'parallel' regions. When a thread encounters a parallel construct, a team of threads is created to be executed in the 'parallel' region. There is an implied barrier at the end of the 'parallel' region. After the end of the 'parallel' region processing only the master thread resumes execution. [13]

2.2.2 Loop Construct

The iterations of the loop construct can be executed by team of threads concurrently.

#pragma omp for *[clause[[,] clause] ...] new-line*
for-loops

where clause is one of the following:

private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*operator*: *list*)
schedule(*kind*[, *chunk_size*])
collapse(*n*)
ordered
nowait

Figure 3 Syntax for Loop Construct [13]

Here, the proposed tool only considers 'private', 'shared' and 'nowait' clauses for analysis purpose. There is no implied barrier at the start of the 'loop' construct. There is an implied barrier at the end of the loop construct. If a 'nowait' clause is specified then there is no implied barrier at the end of the loop construct. It means that once the thread has finished with the execution of parallel region, it terminates and can start execution of another parallel region. This can create a data race in certain situations. [13]

2.2.3 Work Sharing Construct

2.2.3.1 'single' Construct

The 'single' construct specifies only one of the threads from the team of threads that can execute the structured block. There is an implied barrier at the end of the 'single' construct. So, all other threads in the team wait at the end of the 'single' block for the thread to finish the execution of the 'single' block.

#pragma omp single *[clause[[,] clause] ...] new-line*
structured-block

where clause is one of the following:

private(*list***)**
firstprivate(*list***)**
copyprivate(*list***)**
nowait

Figure 4 Syntax for Single Construct [13]

Here, the proposed tool only considers a 'nowait' clause for analysis. If 'nowait' construct is specified then there is no implied barrier at the end of the single block. So, all other threads can be used in another parallel region. [13]

2.2.4 Synchronization Construct

2.2.4.1 'master' Construct

The 'master' construct specifies that only the master thread from the team of threads can execute the structured block.

#pragma omp master
structured-block

Figure 5 Syntax for Master Construct [13]

There is no implied barrier at the start or end of the 'master' construct. [13]

2.2.4.2 'barrier' Construct

The 'barrier' construct explicitly specifies barrier at the end of the construct.

#pragma omp barrier *new-line*

Figure 6 Syntax for Barrier Construct [13]

All threads of the team executing parallel region must execute barrier region and have to wait at that point until all the threads in the team finish the execution up to that point. [13]

2.2.4.3 'critical' Construct

At a given instance, only one thread can execute 'critical' region out of all the threads who are executing the 'critical' region with the same name.

#pragma omp critical *[(name)] new-line*
structured-block

Figure 7 Syntax for Critical Construct [13]

The 'name' is an optional and it can be used to identify the 'critical' construct. A 'critical' construct without a name can be considered as a same. A thread waits at the beginning of the 'critical' region until no thread is executing 'critical' region with the same name. [13]

2.2.4.4 'atomic' Construct

Specific storage location can be updated atomically using the 'atomic' construct.

#pragma omp atomic *new-line*
expression-stmt

where expression-stmt is an expression statement with one of the following forms:

*x binop***=** *expr*
*x***++**
++*x*
*x***--**
--*x*

Figure 8 Syntax for Atomic Construct [13]

The 'atomic' construct is applicable to only one statement. [13]

3. RELATED WORK

3.1 Race Detection Techniques

Analysis of a multithreaded shared memory parallel application is a difficult task. Due to the non-deterministic behavior of a parallel application, it is difficult to find and debug errors. Even if the code is modified, it is difficult to make sure that the error is corrected and not concealed. [4]

There are two approaches for race detection in multi threaded programs.

- Static Analysis
- Dynamic Analysis

3.1.1 Static Analysis

Static analysis employs compile-time analysis on source program. It finds out all the execution paths in a program. The static analysis tool is known for finding out low-level programming errors like null pointer dereferences, buffer overflows, use of uninitialized variables, etc. For race detection, the static analysis tool tries to find out data races at compile time. To find out data races in parallel programs using static analysis is very difficult. [17]

3.1.1.1 Static Analysis Techniques

- **Model Checking:** The model of a system, like hardware or software systems is tested automatically whether this model meets the specifications. The specifications include safety requirements to avoid deadlocks or race conditions [17].
- **Data-flow Analysis:** This technique gathers information about a possible set of values calculated at various points in a computer program. A control flow graph (CFG) is used to

determine those parts of a program to which a particular value assigned to a variable might propagate. Information gathered is often used by a compiler when optimizing a program [17].

Advantages

To detect the errors, a static analyzer does not run a program. A static analyzer does not depend on the execution environment. [10]

Disadvantages

It is very difficult to apply static analysis to find race conditions in multithreaded programs. It is difficult to determine an interleaving between threads. So, it makes a conservative estimation. Scaling is also difficult to achieve using static analysis. It produces more false positive, because it considers all the possible execution paths in the program. [10]

Concurrency analysis is a widely studied field, and several approaches have been presented. Callahan and Subhlok [6] proposed a data flow method. This method calculates a set of blocks which must be executed before a block can be executed for a parallel program. Callahan, Kennedy and Subhlok [7] extended this method to analyze parallel loops. Masticola and Ryder [18] proposed an iterative non-concurrency analysis (a complement problem of concurrency analysis) framework. This method looks for a set of blocks which cannot be concurrent at any time.

Jeremiassen and Eggers [19] proposed a method which deals with explicit parallel program. This method divides a parallel program into phases. Each phase consists of one or more sequence of statements that end with barrier and can execute concurrently. Lin [22] utilized Jeremiassen and Eggers work for an OpenMP programming model.

Sreedhar, Zhang and Gao [16] proposed a new framework for analysis and optimization of shared memory parallel program. This new framework is based on the two concepts of concurrency relation and isolation semantics. Two statements are said to be concurrent if there is execution in which more than one thread can execute the two statements simultaneously. Isolation semantics treat the critical section as being isolated or atomic.

In OpenMP programs, finding parallel regions are comparatively straight forward. So, static analysis technique can be used to detect the race conditions in the OpenMP program. Intel Parallel Lint and VivaMP are static analysis tools for OpenMP programs. This tool helps to find errors in the OpenMP program and provides information for correcting the errors. It also helps to increase the performance of an OpenMP program.

3.1.2 Dynamic Analysis

Dynamic race detector analyzes the trace generated by the execution of a parallel program. This trace contains memory accesses and synchronization operations that are occurred during the executions of a program. Dynamic analysis is totally dependent on the input dataset. So, it can be run more than once with a variety of dataset to make sure the correctness of a program. [21]

Advantages

Dynamic analysis generates less false positive compared to the static analysis. Dynamic analysis provides the reasonably accurate race detection for multithreaded shared memory parallel programs. [21]

Disadvantages

Dynamic analysis analyzes traces of the executed program. So, it does not consider all the possible execution paths of a program. So, Dynamic analysis is not a sound predictor, and race conditions can occur even after the dynamic analysis diagnosed it correct. Dynamic analysis is also dependant on the execution of a program. So, it has overhead of the program execution. [21]

The post-mortem approach records events during program executions and analyzes them later. If an application is long and it has a high interaction with its environment, then the post-mortem approach will not useful. On-the-fly is another approach for recording and analyzing program execution events. In the On-the-fly approach, all data is analyzed in the memory during execution. So, there is no need to store the trace into a file. [21]

There are two dynamic analysis techniques that have been used for race detections named lockset analysis and happens-before analysis. The lockset analysis method verifies programs that follow the locking discipline. If a multithreaded program follows the locking discipline, then it

would confirm an absence of data races. Eraser is a lockset detector proposed by S. Savage et al. [15] Each shared variable is protected by lock. Each shared variable is associated with lockset which keeps track of all locks held during the accesses. When lockset becomes empty, it generates warning. The drawback of this algorithm is that it generates a lot of false positives. Later, lockset algorithm has changed to accommodate more states and transitions to reduce the number of false positives. [21]

Dynamic race detectors can be categorized according to the technique used to monitor a program. One approach is to modify a program binary. It instruments each global memory load and store instructions. This approach is language independent, so it does not depend on source code. Dynamic binary detection has high overhead in terms of time and space. For example, Eraser runs 10 to 30 times slower and shadow memory is required to store lockset information for each word in global memory. This technique is unable to generate readable error reports. [21]

Intel Thread Checker and Sun Thread Analyzer are the most widely used dynamic race detection tools for OpenMP programs. Both tools use the application instrumentation. During the application run, both tools trace references to memory, and thread management operations. Using this information, data races have been checked for events on different threads. The results are post-processed and displayed to the user. Intel Thread Checker is explained in detail in the next section.

4. ANALYSIS TOOLS

4.1 Intel Parallel Lint

Parallel lint, a source code analysis feature, is provided with Intel® Parallel Composer. Parallel lint helps programmers writing and debugging parallel programs. Parallel lint helps programmers in development of parallel applications as well as parallelizing existing serial applications. Parallel lint performs static global analysis to locate parallelization errors in OpenMP programs. It also uncovers errors that would go undetected by the compiler. Parallel lint supports OpenMP 3.0 standard. OpenMP provides a variety of ways for expressing parallelization. Parallel lint can diagnose problems with OpenMP directives and clauses, the nested parallel region, dynamic extent of parallel regions, private/shared/reduction variables in parallel regions, threadprivate variables and expressions used in OpenMP clauses. [14]

Parallel lint also diagnoses some types of deadlocks, data races or potential data dependency and side effects without proper synchronization in an OpenMP program. [14]

Parallel lint analysis performs checks on all the execution paths of a program for all possible values simultaneously. While a dynamic analysis tool is checking a program for a fixed set of input values, it does not check all possible conditions. Parallel lint does not perform full interpretation of the analyzed programs. It can generate so called false-positive messages. [14]

Drawbacks

Parallel lint also does not do the right treatment for the critical region in an OpenMP program.

```
1   int n = 9,l,k = n,i,j;
2   k += n + 1;
3   n++;
4   scanf("%d",&l);
5   printf("%d %d",n,k);
6   #pragma omp parallel default(none) shared(n,k) private(j)
7  {
8   #pragma omp for
9    for(i = 0;i < n;i++)
10   {
11      #pragma omp critical
12     {
13          k = k + 1;
14           j = k + 1;
15          printf("Hello World");
16      }
17   }
18 }
```

Figure 9 Drawbacks of Parallel lint

In the above figure, variable 'k' has been written and read in the same region. In this case, the value assigned to 'j' in a sequential program on line number 14 can be different than in a parallel program. But Parallel lint does no detect this deficiency. Also, output statement in the critical region is atomic. But Parallel lint generates an error message called 'unsynchronized use of I/O statements by multiple threads'.

4.2 Intel Thread Checker

Intel Thread Checker is a dynamic analysis tool to observe the execution of a program. It reports the user where, in a code, problem can exist. It detects incorrect use of threading and synchronization API. It detects data race conditions and deadlock in the OpenMP program. [3]

4.2.1 Intel Thread Checker Architecture Overview

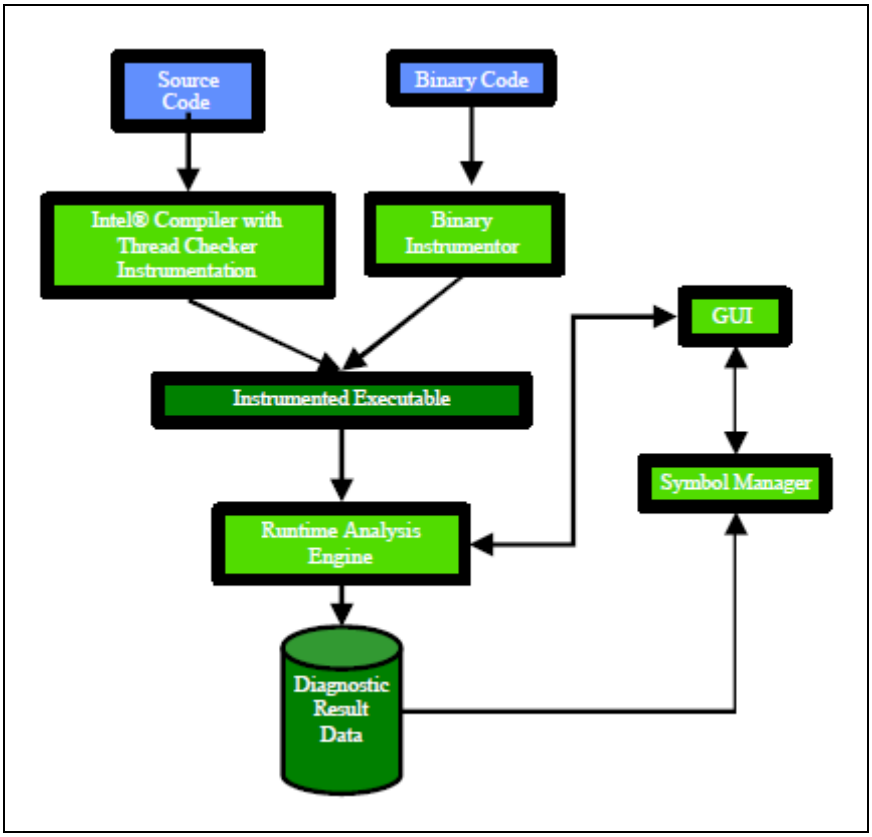

Figure 10 Intel Thread Checker Architecture [3]

Thread Checker starts with instrumentation of source code or binary code of a program. Instrumentation is applied to every memory reference instruction and thread synchronization primitive in a program. During an execution of an instrumented program, a runtime analysis engine inspects every memory reference event and every thread synchronization event. The

diagnostic result of runtime analysis is written in a file and displayed in the GUI in real time. For translation of any address in the diagnostic results in human readable symbolic form and for source line information, GUI interacts with symbol manager and presents precise diagnostics to the user. [3]

4.2.2 Intel Thread Checker Implementation Phases

4.2.2.1 Instrumentation

To determine a run time behavior of each thread, Intel Thread Checker inserts extra code in the instrumentation phase. During a runtime, this instrumented code passes its observation to a runtime analysis engine for analysis. Intel Thread Checker inserts memory instructions and synchronization instructions. Compiler instrumentation and binary instrumentation follow the same principle. [3]

Memory Instructions

Extra code is inserted to each memory instruction that catch the memory access record. Memory access records contain memory access type like read, write, or update, relative virtual instruction address, virtual memory address of the instruction and size of the memory instruction. This instrumentation technique is very expensive. [3]

Synchronization Instructions

Observation and analysis order of synchronization instructions by a runtime analysis engine has to match execution orders of the same instructions. For example, thread T releases a lock on one object and then another thread T acquires a lock on the same object. The analysis engine observes the order of these lock release and acquisition events. Inter Thread Checker replaces

synchronization instruction with a different function which make sure that observation and analysis order is the same as the execution order. [3]

4.2.2.2 Analysis

4.2.2.2.1 Segment and Partial Order Graph

Threads are partitioned into segments which are a sequence of instructions that end with a synchronization instruction. Synchronization instructions can be classified into posting synchronization instructions and receiving synchronization instructions. Posting synchronization posts an occurrence of an event to other threads or itself. ReleaseMutex() is a posting synchronization instruction which posts that mutex is now released and available for other threads for acquisition. Receiving synchronization instruction reads an occurrence of an event posted. WaitForSingleObject() is a receiving synchronization instruction which waits for an occurrence of a mutex releasing event by the releasing a thread. The segment that ends with a posting synchronization instruction is called a posting segment and the segment that ends with a receiving synchronization instruction is called a receiving segment. [3]

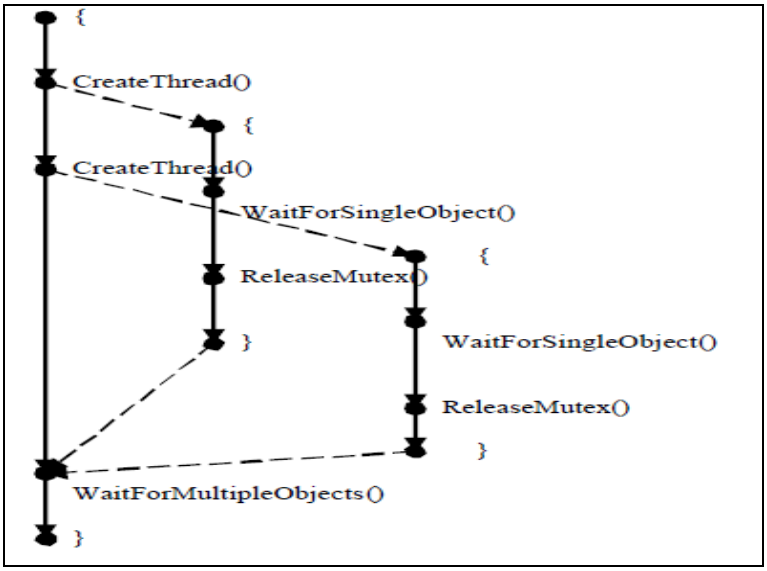

Figure 11 Directed acyclic graph representing program execution[3]

Figure 11 is a dynamic asynchronous graph representing whole program execution. Dot represents synchronous instruction. A posting synchronization instruction must execute before corresponding receiving synchronization instruction.

Now, S1 and S2 are two different segments on the partial order graph.

We say that S1 < S2, if S1 executes before S2. Also, we say that S1 ‖ S2 if neither S1 < S2 nor S2 < S1. [3]

4.2.2.2.2 Vector Clocks

By finding a path from one segment to another on the partial order graph of a program, we can determine the order of any two segments. It is inefficient to find a path from one segment to another on the partial order graph. Vector clock proposes a numeric method for finding the order of two segments. [3]

A vector clock Vs of Segment S is an integer vector associated with segment S. Given set of threads T in the partial order graph, for each $T \in T$,

Vs(T) = [number of posting segments S' on T such that S' < S]

Consider S and S' are the segments on the different threads T and T' respectively, S < S' if Vs(T) < Vs'(T). [3]

When a thread T executes a posting synchronization instruction on an object O of segment S_i, S_i segment ends and a new segment $S_{(i+1)}$ starts. The analysis engine calculates the vector clock $V_{S(i+1)}$ from the vector clock V_{Si}, then it passes the value of $V_{S(i+1)}$ to K_O. When T' executes a receiving synchronization instructions on synchronization object O of segment S'_j, segment S'_j ends and a new segment $S'_{(j+1)}$ starts, the analysis engine calculates the vector clock $V_{S'(j+1)}$ from the vector clock $V_{S'j}$ and the vector clock K_O. [3]

4.2.2.2.3 Shadow Memory

All the shared memory regions are shadowed in Intel Thread Checker. A shared memory region is a collection of shadowed memory cells. A shadow memory cell is an 8-element tuple <m, z, t_r, c_r, l_r, t_w, c_w, l_w>. m represents the starting address of a sub-region. z is the size of the sub-region. t_r is the identifier or thread most recently reading M(m, z). c_r is the vector clock $V_S(t_r)$, l_r is the location of instruction executed by thread t_r most recently reading M(m, z). t_w is the identifier of the thread most recently writing M(m, z). c_w is the vector clock $V_{S'}(t_w)$. l_w is the location of the instruction executed by thread t_w most recently writing M(m, z). [3]

4.2.2.2.4 Data Race Detection

Now consider x as a shared memory location. Segment S on a thread T accessed the x and then a segment S' on a different thread T' accessed the x. Data race occurs on x if $V_S(T) >= V_{S'}(T)$ and either S or S' or both writes to x. [3]

In each thread, the analysis engine monitors six kinds of events named memory read, memory write, synchronization object creation, memory update, posting synchronization instruction and receiving synchronization instructions. Data race detection algorithm by the Intel Thread Checker finds and reports the read-write, write-read, write-write races on the program.[3]

5. The Proposed Methodology

To achieve parallelization in an OpenMP program, we need to specify parallelization construct explicitly. The proposed tool is only considering `for loops` of an OpenMP program for race detection. In OpenMP, the loop iterations of the parallelized loop can be executed out of order. The tool is supporting parallel constructs like 'parallel', 'parallel for' and synchronization constructs like 'single', 'master', 'barrier', 'atomic' and 'critical' in this analysis. Also, the programmer has to declare all the variables used in a parallel region as shared or private.

A static analysis technique is used for analysis of an OpenMP program to detect race conditions. First, the tool gathers the necessary information from a program and stores it in the complex hash structure. The tool is trying to analyze read and write access of the shared variables in a parallel region. The tool does not analyze accesses to a private variable and a read only shared variable in a parallel region.

By analyzing a program, the tool finds out all read and write accesses to the shared variables in a parallel region. The tool also finds out possible concurrent read-write or write-write access statements to the same shared variable. The tool uses barrier based analysis for finding out possible concurrent statements. It means that the tool looks for the concurrent read or write access to the same shared variable until it finds an explicit or implicit barriers in the parallel region.

Once the tool has all the possible concurrent read-write, write-write, and write-read accesses to the same shared variable then it finds out the region in which this variable is written. If the shared variable is written in 'atomic' or 'critical' region then this write access must be atomic. In that case, only one thread can execute that region at a time. So, read-write and write-

read accesses related with write access in a ‘critical’ region is avoided if that shared variable is not written and read on the same line in that ‘critical’ region. In case of ‘atomic’ region, a variable is written as well as read in the same statement. It implies occurrence of read-write dependency race. If a variable is written from ‘critical’ regions with same name for write-write access of a shared variable then both writes are atomic. If both write accesses are done from the atomic region then read-write and write-write dependency race occurs. A read-write dependency race occurs when both write accesses are done from ‘critical’ regions with same name and at least one of the statements contains read and write access to that shared variable. If both write accesses are done from the ‘critical’ regions with different names then write-write race can occur.

The advantage of this methodology is that it can avoid many false positives. Also, Parallel lint and Intel Thread Checker cannot do the right diagnosis for the ‘critical’ regions. But, the proposed tool can provide a right diagnosis for the ‘critical’ region by using this approach. The tool also can detect race conditions in the parallel region of an OpenMP program with mentioned six directives.

6. Design

6.1 Code Parsing

First, input for the analyzer is a C program with a parallel region only. The parser does not parse include statements or void main method. Here, input is a C program and the proposed tool has to parse this program. The tool uses perl regular expression to extract necessary information from a C program. The tool parses a program and stores necessary information line by line.

The tool classifies each statement into various types like ‘DECLARATION’, ‘ASSIGNMENT’, ‘INPUT’, ‘OUTPUT’, ‘PARALLEL’, ‘PARALLEL_FOR’, ‘OMP_FOR’, ‘CRITICAL’ and ‘ATOMIC’. The tool creates a complex data structure to store necessary information regarding each type of statement, which uses hash for that. This hash has a key as a line number. According to the type of the statement, the tool stores related information regarding that statement. Information stored for each type of statement is give below:

A ‘DECLARATION’ type statement contains the declaration of variables. The tool stores type of the statement, statement itself, name and value of the variables declared in the statement and data type of the variables.

All the assignment, increment and decrement statements are considered to be ‘ASSIGNMENT’ type statements. In this statement, the tool stores information like the type of the statement, name and value of the variable written. Also, it stores the name and value of the variables that are read. It stores the operator used for assignment and operators used on the right hand side of the assignment statement and the assignment statement itself.

All the input statements such as ‘scanf’ are stored as a type ‘INPUT’. The tool stores the type of the statement, the statement itself, the name of the input variables. The tool stores the value of the variables as a nondeterministic because it depends on the user input.

All the output statements such as ‘printf’ are stored as a type ‘OUTPUT’. The tool stores the type of the statement, the statement itself, the name and value of the variables that are read for output statement.

A ‘parallel’ statement is stored as a ‘PARALLEL’ type. The tool stores type of the statement, the statement itself, the name of the variable that is declared private in a nested ‘private’ hash, the name of the variable that is declared shared in a nested ‘shared’ hash.

Table 1 Type of the statements classified

Statement	Type
int n = 9,l,k = n,i,j;	DECLARATION
k += n + 1; n++;	ASSIGNMENT
#pragma omp parallel default(none) shared(n,k) private(j)	PARALLEL
#pragma omp for	OMP_FOR
for(i = 0;i < n; i++)	OMP_FOR
#pragma omp critical	CRITICAL
#pragma omp atomic	ATOMIC
#pragma omp single	SINGLE
#prgama omp master	MASTER
printf(“Hello world %d”, n);	OUTPUT
scanf(“%d”, &k);	INPUT

The ‘omp for’ is stored as a ‘OMP_FOR’ type. The tool stores the type of the statement, the statement itself, the name of the variables that are declared private in a nested ‘private’ hash, the name of the variables that are declared shared in a nested ‘shared’ hash and ‘nowait’ value.

A ‘for’ statement after ‘OMP_FOR’ statement is considered a parallel for loop. The tool stores the type of statement, the statement itself, the name and value of the variables that are read and written in the initialization part in a nested ‘initialization’ hash. It stores the name and value of the variables that are read in the condition part in a nested ‘condition’ hash. Also, it stores the name and value of the variables that are written and read in the iterator part in a ‘mutator’ nested hash. It stores the name and value of the variables that are read in the whole ‘for’ statement in

nested ‘read’ hash. Also, it stores the name and value of the variables that are written in the whole ‘for’ statements in a nested ‘write’ hash.

The ‘critical’, ‘atomic’, ‘single’, ‘master’ block declarations are stored as a type ‘CRITICAL’, ‘ATOMIC’, ‘SINGLE’, ‘MASTER’ respectively. The tool stores the type and the statement itself for these statements.

Also stack is used to keep track of the start and end of the structured blocks like parallel, ‘for’, ‘critical’, ‘master’, ‘single’ and ‘atomic’. Stack stores information such as starting line number and ending line number of the structured block and name of the block. For the ‘critical’ region, it stores the region name as well.

The tool parses the code line by line. Once structure block ends, then the tool copies all the parameters in the stack for that structure to another queue. So, this queue stores information regarding structure block in the chronological order of their end line number. The purpose of creating queue is to find out nested structured blocks in a parallel program.

At the end of parsing process, the tool spreads private and shared variable that are declared in the parent parallel region to the nested structural blocks like ‘OMP_FOR’, ‘CRITICAL’, ‘ATOMIC’, ‘SINGLE’, ‘MASTER’. If the structural block has its own additional private variable then it would be added into a list of private variable of that region. For finding out the nested blocks, the tool uses queue created during the parsing process. So, now each and every structured block does have information regarding private and shared variables in the parallel region.

The tool also finds out read only variables in the parallel region. It means that a variable read only in the whole parallel region. The tool also finds out mutable shared variables in the parallel region.

The tool also creates another complex structure called ‘variableStructure’ which stores information regarding read and write access for all shared variable. The tool takes shared variable name as a key in this structure. All the read and write access for that variable is stored under another nested hash called ‘read’ and ‘write’. Also, it stores the line number and the region number for each access. This structure is useful during diagnosis process.

6.2 Diagnosis Process

Now, the proposed tool has a ‘variableStructure’ which stores all the access information regarding read and write access to the shared variables. The tool goes through all the read accesses of every shared variable. Also, the tool goes through all the write accesses to the same shared variable inside each read access. For each combination of read and write access, the tool finds out read and write access to the same shared variable on the same line (READ_WRITE_SAME_LINE), read and write access to the same shared variable in same structured block (READ_WRITE_SAME_REGION) and read and write access to the same shared variable in same parallel region (READ_WRITE_SAME_PARALLEL_REGION). The tool stores this information in ‘variableStructure’ inside a shared variable key as another nested hash with key ‘pairs’.

Also, the tool finds out concurrent write access for each read access. For each write access, the tool finds out concurrent read access or write access. For that the tool scans the program line by line until it finds an explicit or implicit barrier in the parallel region. According to the region in which access occured, there are the following three cases for finding out concurrent read-write or write-write accesses:

CASE – 1: A read or write access is in the 'single' region with nowait value set , read or write access in the 'master' region or read or write access in the 'OMP_FOR' region.

Description: In this case, the tool finds the concurrent write or read access on both the sides of the region in which this read or write access occurred because any of these regions do not have barrier at the end as well at the beginning of the region in which read or write access occurred.

First, the tool finds the boundary of the region in which read or write access occurred. After that, the tool analyzes the program for read or write access beyond the boundary of that region. A concurrent read and write access to the same shared variable by different threads is called READ_WRITE_DEPENDENCY. A concurrent write and read access to the same shared variable by different threads is called WRITE_READ_DEPENDENCY. A concurrent write accesses to the same shared variable by different threads is called WRITE_WRITE_DEPENDENCY. If the tool finds any of the concurrent access to the shared variable, the tool will log it as READ_WRITE_DEPENDENCY, WRITE_READ_DEPENDENCY or WRITE_WRITE_DEPENDENCY. The tool repeats the same procedure for read access. But, the tool only looks for concurrent write access for that current read access. The tool logs concurrent access as WRITE_READ_DEPENDENCY or READ_WRITE_DEPENDANCY.

CASE – 2: A read or write access occurred in 'single' region without 'nowait' is set or in 'OMP_FOR' region without 'nowait' is set

Description: In this case, there is implied barrier at the end of the 'single' and 'OMP_FOR' regions. So, the tool only looks for the concurrent access to the same shared variable from bottom to top at the starting point of the region. This region is a region in which read or write access occurred and the tool is looking for the concurrent read or write access outside of that

region. The tool analyzes the code until an explicit or implicit barrier is found. The tool logs all the concurrent accesses for the shared variable as 'READ_WRITE_DEPENDENCY', 'WRITE_READ_DEPENDENCY', or 'WRITE_WRITE_DEPENDENCY'.

CASE – 3: A read or write occurred in a 'PARALLEL' region only.

Description: The tool looks for concurrent access in both the sides of that line on which read or write access occurred until it finds a barrier in each direction.

Also, the tool stores the write status according to the region in which write access is occurred for each write access to the shared variable. For example, if write access occurred on ATOMIC region then it will store the write status as WRITE_ATOMIC.

Once all the read-write, write-read and write-write possible concurrent accesses are found, the tool further analyzes the code to find the conflicts. It checks the following rules for finding race condition:

1) If WRITE_WRITE_DEPENDENCY is found and both write accesses are WRITE_ATOMIC then write-read or read-write dependency race will occur.
2) If WRITE_WRITE_DEPENDENCY is found and one of the write accesses is WRITE_ATOMIC then write-read dependency race will occur.
3) If WRITE_WRITE_DEPENDENCY is found and both write accesses are WRITE_CRITICAL and have different region name then write-write race will occur.
4) If WRITE_WRITE_DEPENDENCY is found and both write accesses are WRITE_CRITICAL and have same region name and one of the write accesses have READ_WRITE_SAME_LINE is set then write-write dependency race will occur.
5) If WRITE_WRITE_DEPENDENCY is found and one of the write accesses is WRITE_CRITICAL then write-read or read-write dependency race will occur.

6) If WRITE_PARALLEL or WRITE_OMP_FOR is found then write-write race will occur.
7) If READ_WRITE_DEPENDENCY or WRITE_READ_DEPENDENCY is found, one write is WRITE_CRITICAL and READ_WRITE_SAME_LINE is set and READ_WRITE_SAME_REGION is set then read-write dependency race will occur.
8) IF READ_WRITE_DEPENDENCY or WRITE_READ_DEPENDENCY is found and one write is WRITE_ATOMIC or WRITE_CRITICAL with READ_WRITE_SAME_LINE is set and READ_WRITE_SAME_PARALLEL_REGION is set then read-write dependency race will occur.
9) IF READ_WRITE_DEPENDENCY or WRITE_READ_DEPENDENCY is found and one write is WRITE_CRITICAL without READ_WRITE_SAME_LINE set then read-write dependency race will not occur.
10) For all other READ_WRITE_DEPENDENCY or WRITE_READ_DEPENDENCY, read-write dependency race will occur.

These rules are defined to detect race conditions in the OpenMP parallel program. If these rules fail to detect race conditions in the OpenMP program in the specific circumstance, then we can add rules to detect the race condition for that circumstance.

6.3 Example Problem

We take an example problem to understand the implemented methodology.

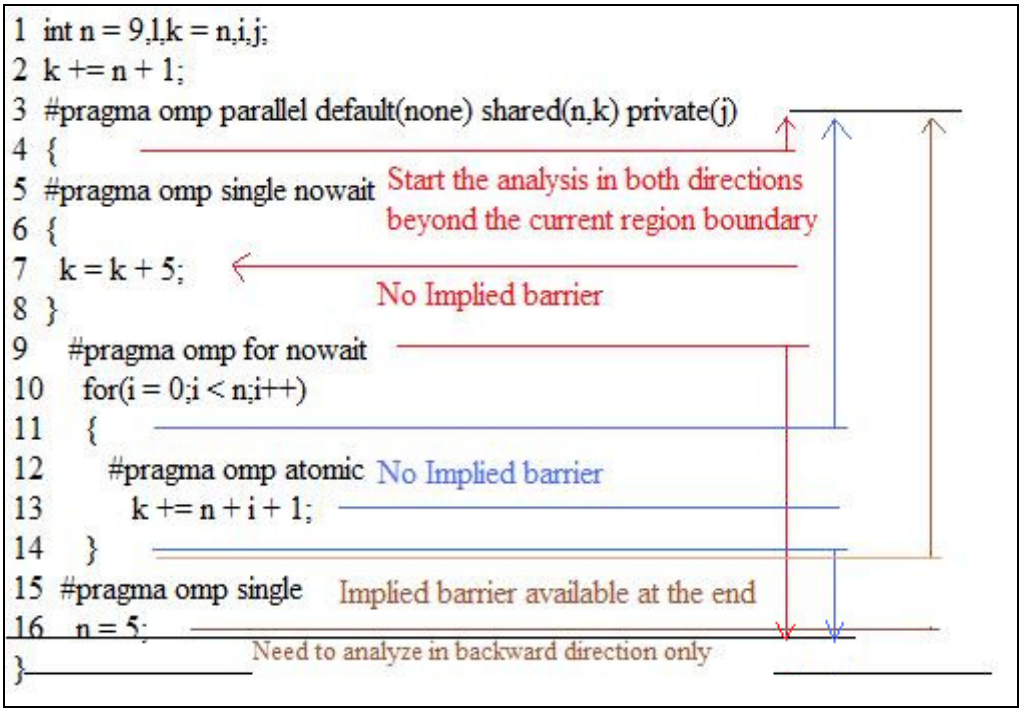

Figure 12 Example to Describe Proposed Race Detection Methodology

The example described in Figure 12 contains two shared variables ‘k’ and ‘n’. The proposed tool finds all the read and write accesses to each of the shared variables. Shared variable ‘k’ is written and read on line number 7 and under region name ‘single’ with ‘nowait’. ‘k’ is also read and written on line number 12 in the ‘atomic’ region. To find out concurrent accesses to the same ‘k’ variable from the line number 7, it analyzes the read and write access to the variable ‘k’ from the boundary of the ‘single’ region in both directions until an implied or explicit barrier reached. In this case, it analyzes the code from line number 4 to line number 3 in an upper direction and the tool analyzes the code from line number 9 to line number 16. To find out the concurrent access to the ‘k’ at line number 13, it analyzes the code in both direction of the atomic region until it

finds an implied or explicit barrier. In this way, it finds all the concurrent read-write, write-write and write-read accesses to the shared variable for each read or write operation.

All possible pairs of concurrent accesses to variable ‘k’ found by the proposed method is given below,

```
1) 'lineno1' => 7,                          // Read access line number
   'region1' => '0',                        // Read access region number
   'name' => 'READ_WRITE_DEPENDENCY',
   'region2' => '1',                        // Write access region number
   'lineno2' => 13                          // Write access line number

2) 'lineno1' => 7,                          // Write access line number
   'region1' => '0',                        // Write access region number
   'name' => 'WRITE_READ_DEPENDENCY',
   'region2' => '1',                        // Read access region number
   'lineno2' => 13                          // Read access line number

3) 'lineno1' => 7,                          // Write access line number
   'region1' => '0',                        // Write access region number
   'name' => 'WRITE_WRITE_DEPENDENCY',
   'region2' => '1',                        // Write access region number
   'lineno2' => 13                          // Write access line number

4) 'lineno1' => 7,                          // Read access line number
   'region1' => '0',                        // Read access region number
   'name' => 'READ_WRITE_DEPENDENCY',
   'region2' => '1',                        // Write access region number
   'lineno2' => 13                          // Write access line number

5) 'lineno1' => 7,                          // Write access line number
   'region1' => '0',                        // Write access region number
   'name' => 'WRITE_WRITE_DEPENDENCY',
   'region2' => '1',                        // Write access region number
   'lineno2' => 13                          // Write access line number
```

Following are the write statuses for shared variable ‘k’

```
1) 'region' => '0',  // Write is occurred in single region
   'name' => 'WRITE_SINGLE',
   'lineno' => 7

2) 'region' => '0', // Read and write for variable ‘k’ occurred on same line
```

'name' => 'READ_WRITE_SAME_LINE',
'lineno' => 7 // line number 7

3) 'region' => '1', // Write is occurred in atomic region
'name' => 'WRITE_ATOMIC',
'lineno' => 13

4) 'region' => '2', // Read and write for variable ‘k’ is occurred on same line
'name' => 'READ_WRITE_SAME_LINE',
'lineno' => 13 // line number 13

Same way for the shared variable ‘n’, tool found following list of the possible concurrent

accesses.

Pairs of possible concurrent accesses for the variable ‘n’ is give below:

1) 'lineno1' => 13,
'region1' => '1',
'name' => 'READ_WRITE_DEPENDENCY',
'region2' => '3',
'lineno2' => 16

2) 'lineno1' => 10,
'region1' => '2',
'name' => 'READ_WRITE_DEPENDENCY',
'region2' => '3',
'lineno2' => 16

Following are the write statuses for shared variable ‘n’

1) 'region' => '3',
'name' => 'WRITE_SINGLE',
'lineno' => 16

Diagnosis for the program given in the figure 12 is as below:

1) Read - Write dependancy race can occure for shared variable name 'n' at line no : 13 and 16
2) Read - Write dependancy race can occure for shared variable name 'n' at line no : 10 and 16
3) Read - Write dependancy race can occure for shared variable name 'k' at line no : 7 and 13
4) Write - Write dependancy race can occure for shared variable name 'k' at line no : 7 and 13

7. RESULTS

We have compared the diagnosis of the proposed methodology, Intel Parallel lint and Intel Thread Checker. We have certain test cases where our proposed methodology gives a better diagnosis compared to the Intel Parallel lint and Intel Thread Checker. All the test cases are explained below:

7.1 Test Case 1: Assignment of Constant to Shared Variable by Parallel Threads

In this test case, 'k' is a shared variable. 'k' is assigned value '5' in parallel region. So, each and every thread in the parallel region would assign value `5` to `k`. This can result into write-write conflict. More than one thread can assign the same value. It is preferable to put the statement into a single region, so that only one thread can assign value to 'k'. Once that thread finishes the execution of a single region, all the threads resume execution from the next statement. This program should be modified and an analysis tool should generate some error messages for the precaution.

```
1  int n = 9,l,k = n,i,j;
2  k += n + 1;
3  n++;
4  printf("Hello World %d %d",n,k);
5  #pragma omp parallel default(none) shared(n,k) private(j)
6  {
7    k =  5;
8     #pragma omp for
9     for(i = 0;i < n;i++)
10     {
11        j = k + 1;
12      }
13  }
```

Figure 13 Test Case-1 Assignment of Constant to Shared Variable by Parallel Threads

Parallel lint Diagnosis: Parallel lint does not generate any error messages.

Intel Thread Checker Diagnosis: Intel Thread checker generates following error message:

Write-Write data races at line number 7

The Proposed Tool Diagnosis:

1) Write of constant to shared variable 'k' should be in single region at line no : 7
2) Read - Write dependency race can occur for shared variable name 'k' at line number 7 and 11

7.2 Test Case 2: Output Statement Synchronization Test

In this test case, the output statement is in critical region, so the output statement is atomic. Parallel lint still generates the error messages like an unsynchronized use of I/O statement which is false positive. This program is race free and analysis tool should not generate any error messages.

```
1  int n = 9,l,k = n,i,j;
2  k += n + 1;
3  n++;
4  printf("Hello World %d %d",n,k);
5  #pragma omp parallel default(none) shared(n,k) private(j)
6  {
7   #pragma omp for
8    for(i = 0;i < n;i++)
9    {
10    #pragma omp critical
11     {
12        k = k + 1;
13        printf(" Hello World !!!");
14      }
15   }
16  }
```

Figure 14 Test Case-2 Output Statement Synchronization Test

Parallel lint Diagnosis: In this case, Parallel lint generates the false positive.

error #12158: unsynchronized use of I/O statements by multiple threads.

Intel Thread Checker Diagnosis: Intel Thread Checker could not determine the critical region so they generates the following error messages

1) Read-Write data race
2) Write-Read data race
3) Write-Write data race

So, Intel Thread Checker generates false positives.

The Proposed Tool Diagnosis: It does not generate any warning or error messages.

7.3 Test Case - 3: Critical Region Test

In this test case, read and write accesses to the shared variable ‘k’ occurs in the same statement, then in the next line, ‘k’ is read. In OpenMP, loop iterations are executed out of order by parallel threads. So, value assigned to ‘j’ at particular iteration is not the same as in the sequential program for the same iteration. In this case, read-write dependency race occurs. An analysis tool should generate warning or error messages for this defect.

```
1  int n = 9,l,k = n,i,j;
2  k += n + 1;
3  n++;
4  printf("Hello World %d %d",n,k);
5  #pragma omp parallel default(none) shared(n,k) private(j)
6  {
7    #pragma omp for
8      for(i = 0;i < n;i++)
9       {
10        #pragma omp critical
11          {
12              k = k + 1;
13              j = k + 1;
14          }
15     }
16 }
```

Figure 15 Test Case-3 Critical Region Test

Parallel lint Diagnosis: In this case, Parallel lint has a false negative. It does not generate any error messages.

Intel Thread Checker Diagnosis: Intel Thread Checker could not determine the critical region so, they generates the following error messages

1) Read-Write data race
2) Write-Read data race
3) Write-Write data race

So, Intel Thread Checker generates false positives as well as false negative.

The Proposed Tool Diagnosis: It generates following warning message.

Read - Write dependency race can occur for shared variable name 'k' at line number 13 and 12.

7.4 Test Case – 4: Atomic Region Test

This test case is similar to Test Case - 3. Only we use atomic region instead of critical region. In this case, read-write dependency race occurs. An analysis tool should generate warning or error messages for this defect.

```
1   int n = 9,l,k = n,i,j;
2   k += n + 1;
3   n++;
4   printf("Hello World %d %d",n,k);
5  #pragma omp parallel default(none) shared(n,k) private(j)
6   {
7     #pragma omp for
8     for(i = 0;i < n;i++)
9    {
10   #pragma omp atomic
11       k += 1;
12    j = k + 1;
13    }
14  }
```

Figure 16 Test Case-4 Atomic Region Test

Parallel lint Diagnosis: In this case, Parallel lint generates the proper warning message.

Warning: flow data dependence from (file:openmptest1.cpp line:11) to (file:openmptest1.cpp line:12), due to "k" may lead to incorrect program execution in parallel mode

Intel Thread Checker Diagnosis: Intel Thread Checker can not determine data dependency issue. So it does not generate any error messages. So, Intel Thread Checker generates a false negative.

The Proposed Tool Diagnosis: It generates the following warning message:

Read - Write dependency race can occur for shared variable name 'k' at line number 12 and 11.

7.5 Test Case – 5: Dependency Test in Critical Region

In this test case, read and write access to the shared variable ‘k’ is done at the same line. ‘k’ variable has not been read by any other statements. This program is fine and an analysis tool should not generate any error messages.

```
1   int n = 9,l,k = n,i,j;
2   k += n + 1;
3   n++;
4   printf("Hello World  %d %d",n,k);
5   #pragma omp parallel default(none) shared(n,k) private(j)
6    {
7      #pragma omp for
8        for(i = 0;i < n;i++)
9        {
10         #pragma omp critical
11          {
12             k += 1;
13           }
14      }
15   }
```

Figure 17 Test Case-5 Dependency Test in Critical Region

Parallel lint Diagnosis: In this case, Parallel lint does not generate any error message.

Intel Thread Checker Diagnosis: Intel Thread Checker can not determine the critical region, so it generates the following error messages

1) Read-Write data race
2) Write-Read data race
3) Write-Write data race

So, Intel Thread Checker generates false positives.

The Proposed Tool Diagnosis: It does not generate any warning messages.

7.6 Test Case – 6: Dependency Test in Critical Region

In this test case, even if the shared variable 'k' is written and read in the next line, the program generates proper outcome because read and write operation to the shared variable 'k' has not occurred on the same statement. The value of 'k' is dependent on private variables. In this case, no error message should be generated.

```
1   int n = 9,l,k = n,i,j;
2   k += n + 1;
3   n++;
4  printf("Hello World %d %d",n,k);
5  #pragma omp parallel default(none) shared(n,k) private(j)
6   {
7     #pragma omp for
8       for(i = 0;i < n;i++)
9       {
10        #pragma omp critical
11        {
12             k = i + 1;
13             j = k + 1;
14         }
15     }
16  }
```

Figure 18 Test Case-6 Dependency Test in Critical Region

Parallel lint Diagnosis: Parallel lint does not generate any messages.

Intel Thread Checker Diagnosis: Intel Thread Checker cannot determine the critical region. So, they generate the following error message

1) Write-Read data race

So, Intel Thread Checker generates false positives.

The Proposed Tool Diagnosis: It does not generate any warning messages.

7.7 Overall Statistics

Table 2 Results statistics

Test Cases	The Proposed Tool	Intel Parallel lint	Intel Thread Checker
Assignment of constant to shared variable	✓	✗	✓
Output Statement Synchronization Test	✓	✗ False Positive	✗ False Positive
Critical Region Test	✓	✗ Fase Negative	✗ False Positive
Atomic Region Test	✓	✓	✗ False Negative
Dependancy Test in Critical Region	✓	✓	✗ False Positive
Dependancy Test in Critical Region	✓	✓	✗ False Positive

So, overall statistics show that, the proposed method detects the race conditions in the ‘critical’ region that have not been detected by commercial tools named Intel Parellel lint and Intel Thread Checker.

8. CONCLUSION

The results show that the proposed method detects the race conditions in the ‘critical’ region that is not detected by commercial tools named Intel Parellel lint and Intel Thread Checker. Also apart from ‘critical’ construct, the proposed tool can detect the race conditions for the ‘parallel’, ‘omp for’, ‘single’, ‘master’, ‘atomic’ and ‘barrier’ constructs.

The proposed method of analysis utilizes rules for OpenMP directives. The proposed method of analysis finds out combinations of possible concurrent accesses for the shared variables. It also detects ‘atomic’ and ‘critical’ regions and provides correct diagnosis. It applies the rules on all the possible concurrent accesses to find out the race conditions in the OpenMP parallel program.

9. FUTURE WORK

The proposed static analysis technique does not generate false negatives for the ‘critical’ region that are observed in Intel Parallel lint or Intel Thread Checker.

Perl regualer expression is used to parse the code. The regulare expression is not the sound technique to parse the code. We can develop proper OpenMP grammer to parse the code.

Right now, the proposed tool works only for ‘for loop’. It only supports ‘parallel’ and ‘parallel for’ constructs for parallelization as well as ‘single’,’ master’, ‘barrier’, ‘critical’ and ‘atomic’ synchronization constructs. We can extend this tool to analyze an OpenMP program for all OpenMP constructs. We can extend this tool to analyze pointer and array for race condition detection.

Also, the proposed tool uses limited dependency analysis for finding out all execution paths of a parallel program. We can apply more dependency analysis to give comprehensive diagnosis for occurrences of race conditions.

APPENDIX A

```
#!/usr/bin/perl
use warnings;
use CGI qw(:standard);
use Switch;


my $inputCode = param("inputCode");
print header(),start_html("CS298 : Analysis of OpenMP code to avoid Race Condition");
print h1("CS298 : Analysis of OpenMP code to avoid Race Condition.");
print h3("Please Enter you code in Text Area <br>");
print '<div id = "main" style="align:center; margin:5px; border:2px; width:90%; "><div id = "form_element" style =
"float:left; width: 40%; margin: 10px ">';
print start_form(), textarea(-name => "inputCode", -rows => "35" , -cols => '80');
print submit(" Analyse Code"), end_form();
print '</div><div id = "results" style ="float:right; width: 40%; display:inline margin: 10px">';

if ($inputCode)
{
        my @lines = ();
        @lines = split(/\n/,$inputCode);

        @analysisResults = doAnalyze(\@lines);
        my $counter = 1;
        print "Analysis Results : <br />";
        foreach my $analysisResult (@analysisResults)
        {
                print  " $counter ) $analysisResult <br />";
                $counter++;
        }
  }
print '</div></div>';
print end_html();

sub doAnalyze
{
my $codeLines = $_[0];
my %data = ();
my %structureStack = ();
my %structureQueue = ();
my @words;
my $lineno = 0;
my $top = -1;
my $nextElement = '';
my $checkNext = 0;
my @variableReadOnly;
my @varibaleShared;
my @variablePrivate;
my @variableFirstPrivate;
my @variableLastPrivate;
my @combineVariable;
my %variableStruct;
my @finalMessages;
```

```
foreach $line (@$codeLines)
{
$lineno++;
$line = trim($line);

        if($nextElement ne '' && $line !~ /$nextElement/)
        {
        }
        elsif($nextElement ne '' && $line =~ /for/)
        {
          $nextCheck = 1;
        }
        if($line =~ /^(int|omp_lock_t)\s+(.*)?;$/)
        {
          declarationStatements(\%data,$line,$lineno);
             }
        elsif($line =~ /^#pragma\s+omp\s+parallel\s+(for)?/)
             {
         ($top,$nextElement) = parallelStatement(\%data,$line,$lineno,\%structureStack,$top);
             }
             elsif($line =~ /^#pragma\s+omp\s+barrier/)
             {
         ($top,$nextElement) = barrierStatement(\%data,$line,$lineno,\%structureStack,$top,\%structureQueue);
             }
        elsif($line =~ /^#pragma\s+omp\s+critical/)
        {
         ($top,$nextElement) = criticalStatement(\%data,$line,$lineno,\%structureStack,$top);
        }
        elsif($line =~ /^#pragma\s+omp\s+master/)
        {
          ($top,$nextElement) = masterStatement(\%data,$line,$lineno,\%structureStack,$top);
        }
        elsif($line =~ /^#pragma\s+omp\s+single/)
        {
                ($top,$nextElement) = singleStatement(\%data,$line,$lineno,\%structureStack,$top);
        }
        elsif($line =~ /^#pragma\s+omp\s+atomic/)
        {
          ($top,$nextElement) = atomicStatement(\%data,$line,$lineno,\%structureStack,$top);
        }
        elsif($line =~ /^omp_set_lock/)
        {
          ($top,$nextElement) = lockStatement(\%data,$line,$lineno,\%structureStack,$top);
        }
        elsif($line =~ /^omp_unset_lock\(&(.*?)\);$/)
        {
           if($top>=0 && $structureStack{$top}->{'name'} eq "SIMPLE_LOCK" &&  $structureStack{$top}-
>{'lockname'} eq trim($1))
           {
                        $data{$lineno}{'type'} = 'SIMPLE_LOCK';
                        $data{$lineno}{'lineno'} = $lineno;
                    $data{$lineno}{'statement'} = $line;
               $structureStack{$top}->{'end'} = $lineno;
               $top--;
           }
```

```
        elsif($top>=0 && $structureStack{$top}->{'name'} eq "SIMPLE_LOCK" &&  $structureStack{$top}-
>{'lockname'} ne trim($1))
         {
         }
         $nextElement = '';
      }
      elsif($line =~ /^{/)
      {
        if($top>=0 && $structureStack{$top}->{'name'} eq 'ATOMIC' )
        {

        }
        elsif($top>=0 && (($structureStack{$top}->{'name'} eq 'PARALLEL_FOR' || $structureStack{$top}-
>{'name'} eq 'OMP_FOR')  &&  ($lineno - $structureStack{$top}->{'start'})  > 2) || (($structureStack{$top}-
>{'name'} ne 'PARALLEL_FOR' && $structureStack{$top}->{'name'} ne 'OMP_FOR')  &&  ($lineno -
$structureStack{$top}->{'start'})  > 1))  )
        {
         $structureStack{++$top}->{'name'} = 'BRACES';
         $structureStack{$top}->{'start'} = $lineno;
                  $data{$lineno}{'type'} = $structureStack{$top}->{'name'};
                  $data{$lineno}{'lineno'} = $lineno;
                  $data{$lineno}{'statement'} = $line;
        }
        else
        {
                  $data{$lineno}{'type'} = $structureStack{$top}->{'name'};
                  $data{$lineno}{'lineno'} = $lineno;
                  $data{$lineno}{'statement'} = $line;
            }
         $nextElement = '';
      }
      elsif($line =~ /^}/)
      {
        if($top >= 0)
        {
          $data{$lineno}{'type'} = $structureStack{$top}->{'name'};
               $data{$lineno}{'lineno'} = $lineno;
               $data{$lineno}{'statement'} = $line;
          $structureStack{$top}{'end'} = $lineno;
          $nextQueueIndex = scalar(keys(%structureQueue));
          foreach $key (keys(%{$structureStack{$top}}))
          {
           $structureQueue{$nextQueueIndex}->{$key} = $structureStack{$top}->{$key};
          }
          $top--;
        }
        $nextElement = '';
      }
      elsif($line =~ /^#pragma\s+omp\s+for/)
      {
          ($top,$nextElement) = forStatement(\%data,$line,$lineno,\%structureStack,$top);
      }
      elsif($line =~ /^(for|if)\((.*?)\)$/)
      {
          ($top,$nextElement) = controlStructure(\%data,$line,$lineno,\%structureStack,$top);
```

```
            if($nextCheck == 1)
        {
                $data{$lineno}{'type'} = 'OMP_FOR';
          $nextCheck = 0;
            }
    }
    else
    {
        ($top, $nextElement) = normalStatement(\%data,$line,$lineno,\%structureStack,$top,\%structureQueue);
    }
}

recalculate(\%data,\%structureQueue);
copyVariable(\%data,\%structureQueue);
findReadOnlyShared(\%data,\%structureQueue,\@variableReadOnly);
findShared(\%data,\%structureQueue,\@variableShared);
findPrivate(\%data,\%structureQueue,\@variablePrivate);
findFirstPrivate(\%data,\%structureQueue,\@variableFirstPrivate);
findLastPrivate(\%data,\%structureQueue,\@variableLastPrivate);
combineVariables(\@combineVariable,\@variableShared,\@variablePrivate,\@variableFirstPrivate,\@variableLast
Private);

getVariableDetails(\%data,\%structureQueue,\%variableStruct);
diagnose(\%data,\%structureQueue,\%variableStruct,\@combineVariable,\@variableShared,\@variablePrivate,\@va
riableFirstPrivate,\@variableLastPrivate,\@variableReadOnly);
displayResults(\%data,\%structureQueue,\%variableStruct,\@combineVariable,\@variableShared,\@variablePrivate
,\@variableFirstPrivate,\@variableLastPrivate,\@variableReadOnly,\@finalMessages);
return @finalMessages;
}

sub displayResults
{
  my $data = $_[0];
  my $stQueue = $_[1];
  my $varInfo = $_[2];
  my $combinedVars = $_[3];
  my $sharedVars = $_[4];
  my $privateVars = $_[5];
  my $firstprivateVars = $_[6];
  my $lastprivateVars = $_[7];
  my $readonlyVars = $_[8];
  my $msgs = $_[9];
  my $check = 0;
  my $i = 0;
  my $parallelStart = 0;
  my $parallelEnd = 0;
  my $regionName = '';
  my $regionIndex = -1;
  my $regionIndex1 = -1;
  my $writeLineNo = -1;
  my $readWriteSameLine = 0;
  my $tmpWrites1 = '';
  my $tmpRegion1 = -1;
  my $tmpWrites2 = '';
  my $tmpRegion2 = -1;
```

```
        foreach  $key ( sort { $a <=> $b } keys(%$stQueue))
    {
        if($stQueue->{$key}{'name'} eq 'PARALLEL' || $stQueue->{$key}{'name'} eq 'PARALLEL_FOR')
        {
                $parallelStart = $stQueue->{$key}{'start'};
          $parallelEnd = $stQueue->{$key}{'end'};
          last;
        }
    }

        for($i = $parallelStart;$i <= $parallelEnd;$i++)
        {
                if($data->{$i}{'type'} eq 'OUTPUT')
                {
                        my $regionIndex = findRegionIndex($stQueue,$i);
                        if($stQueue->{$regionIndex}{'name'} ne 'CRITICAL' &&
                                $stQueue->{$regionIndex}{'name'} ne 'SINGLE' &&
                                $stQueue->{$regionIndex}{'name'} ne 'MASTER')
                        {
                        print $stQueue->{$regionIndex}{'name'};
                                push(@$msgs, "printf statement, at line no $i, should be in Critical, Single or
Master region .\n");
                        }
                }
        }



        foreach my $sharedVar (@$sharedVars)
    {
                $check = 0;
                foreach $readonlyVar (@$readonlyVars)
                {
                        if($readonlyVar eq $sharedVar)
                        {
                                $check = 1;
                        }
                }
                if($check == 1)
                {
                   next;
                }

                foreach my $status (@{$varInfo->{$sharedVar}{'status'}})
                {
                        if($status->{'name'} eq 'WRITE_PARALLEL' || $status->{'name'} eq
'WRITE_PARALLEL_FOR' || $status->{'name'} eq 'WRITE_OMP_FOR')
                        {
                                push(@$msgs, "Write - Write Race can occure for shared variable name
'$sharedVar' at line no : $status->{'lineno'}");
                        }
                }
                foreach my $pairs (@{$varInfo->{$sharedVar}{'pairs'}})
```

```
		{
			if(defined($pairs->{'done'}))
			{
				next;
			}
		  if($pairs->{'name'} eq 'WRITE_WRITE_DEPENDANCY')
		  {
			$tmpWrites1 = '';
			$tmpRegion1 = -1;
			$tmpWrites2 = '';
			$tmpRegion2 = -1;
			$readWriteSameLine = 0;
			foreach my $tmpPairs (@{$varInfo->{$sharedVar}{'pairs'}})
			{

				if($pairs->{'name'} eq 'WRITE_WRITE_DEPENDANCY' &&
					($pairs->{'lineno1'} == $tmpPairs->{'lineno2'} || $pairs->{'lineno1'}
== $tmpPairs->{'lineno1'}) &&
					($pairs->{'lineno2'} == $tmpPairs->{'lineno2'} || $pairs->{'lineno2'}
== $tmpPairs->{'lineno1'}) &&
					$pairs->{'name'} eq $tmpPairs->{'name'})
				{
					$tmpPairs->{'done'} = 1;
				}
			}

			foreach my $writes (@{$varInfo->{$sharedVar}{'status'}})
			{
				if(defined($writes->{'lineno'}) && $writes->{'lineno'} == $pairs->{'lineno1'}
&&
					$writes->{'name'} ne 'READ_WRITE_SAME_LINE')
				{
					$tmpWrites1 = $writes->{'name'};
					$tmpRegion1 = $writes->{'region'};
				}
				elsif(defined($writes->{'lineno'}) && $writes->{'lineno'} == $pairs->{'lineno2'}
&&
				$writes->{'name'} ne 'READ_WRITE_SAME_LINE')
				{
					$tmpWrites2 = $writes->{'name'};
					$tmpRegion2 = $writes->{'region'};
				}
				if(defined($writes->{'lineno'}) &&
					($writes->{'lineno'} == $pairs->{'lineno2'} || $writes->{'lineno'} ==
$pairs->{'lineno1'}) &&
					$writes->{'name'} eq 'READ_WRITE_SAME_LINE' )
				{
					$readWriteSameLine = 1;
				}

			}

			if(($tmpWrites1 eq 'WRITE_ATOMIC' || $tmpWrites2 eq 'WRITE_ATOMIC') &&
$tmpWrites2 ne $tmpWrites1 )
			{
```

```
                                        push(@$msgs, "Write - Write dependancy race can occure for shared variable
name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                                elsif($tmpWrites1 eq 'WRITE_ATOMIC' && $tmpWrites2 eq 'WRITE_ATOMIC')
                                {
                                        push(@$msgs, "Write - Read or Read - Write dependancy race can occure for
shared variable name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                                elsif($tmpWrites1 eq $tmpWrites2 && $tmpWrites1 eq 'WRITE_CRITICAL')
                                {
                                                #print "\n$tmpWrites1 : $tmpWrites2";
                                        if($tmpRegion1 ne $tmpRegion2 )
                                        {
                                                if($stQueue->{$tmpRegion1}->{'regionname'} ne $stQueue-
>{$tmpRegion2}->{'regionname'} &&
                                                        $readWriteSameLine == 1)
                                                {
                                                        push(@$msgs, "Write - Write dependancy race can occure for
shared variable name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                                }
                                                elsif($stQueue->{$tmpRegion1}->{'regionname'} ne $stQueue-
>{$tmpRegion2}->{'regionname'} &&
                                                        $readWriteSameLine == 0)
                                                {
                                                        push(@$msgs, "Write - Write race can occure for shared
variable name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                                }
                                        }
                                }
                                elsif($tmpWrites1 ne $tmpWrites2 && ($tmpWrites1 eq 'WRITE_CRITICAL' ||
$tmpWrites2 eq 'WRITE_CRITICAL')  &&
                                        $readWriteSameLine == 1)
                                {
                                        push(@$msgs, "Write - Write dependancy race can occure for shared variable
name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                                elsif($tmpWrites1 ne $tmpWrites2 && ($tmpWrites1 eq 'WRITE_CRITICAL' ||
$tmpWrites2 eq 'WRITE_CRITICAL')  &&
                                        $readWriteSameLine == 0)
                                {
                                        push(@$msgs, "Write - Write race can occure for shared variable name
'$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                                elsif($tmpWrites1 ne 'WRITE_CRITICAL' && $tmpWrites1 ne 'WRITE_ATOMIC' &&
$tmpWrites2 ne 'WRITE_CRITICAL'  && $tmpWrites2 ne 'WRITE_ATOMIC')
                                {
                                        push(@$msgs, "Write - Write race can occure for shared variable name
'$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }

                        }
                        else
                        {
                                $regionName = '';
                                $regionIndex = -1;
```

```
$regionIndex1 = -1;
$writeLineNo = -1;
$readWriteSameLine = 0;
$tmpWrites1 = '';
foreach my $tmpPairs (@{$varInfo->{$sharedVar}{'pairs'}})
{


	if($tmpPairs->{'name'} ne 'WRITE_WRITE_DEPENDANCY' &&
		($pairs->{'lineno1'} == $tmpPairs->{'lineno2'} || $pairs->{'lineno1'}
== $tmpPairs->{'lineno1'}) &&
		($pairs->{'lineno2'} == $tmpPairs->{'lineno2'} || $pairs->{'lineno2'}
== $tmpPairs->{'lineno1'}))
	{

		$tmpPairs->{'done'} = 1;
		if($tmpPairs->{'name'} eq 'READ_WRITE_SAME_REGION' ||
$tmpPairs->{'name'} eq "READ_WRITE_SAME_PARELLEL_REGION")
		{
			$regionName = $tmpPairs->{'name'};
			$regionIndex = $tmpPairs->{'region2'};
		}
	}
}
if($pairs->{'name'} eq 'READ_WRITE_SAME_REGION' || $pairs->{'name'} eq
'READ_WRITE_DEPENDANCY' )
{
	$writeLineNo = $pairs->{'lineno2'};
}
elsif($pairs->{'name'} eq 'WRITE_READ_DEPENDANCY')
{
	$writeLineNo = $pairs->{'lineno1'};
}
foreach $writes (@{$varInfo->{$sharedVar}{'status'}})
{
	if(defined($writes->{'lineno'}) &&  $writes->{'lineno'} == $writeLineNo)
	{
		$tmpWrites1 = $writes->{'name'};
		$tmpRegion1 = $writes->{'region'};
	}
	if(defined($writes->{'lineno'}) &&  $writes->{'lineno'} == $writeLineNo &&
$writes->{'name'} eq 'READ_WRITE_SAME_LINE' )
	{
		$readWriteSameLine = 1;
	}
}
if($tmpWrites1 eq 'WRITE_CRITICAL' && $readWriteSameLine == 1 &&
	$regionName eq 'READ_WRITE_SAME_REGION')
{
	push(@$msgs, "Read - Write dependancy race can occure for shared variable
name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
}
elsif($tmpWrites1 eq 'WRITE_CRITICAL' &&
	$readWriteSameLine == 0 && $regionName eq
'READ_WRITE_SAME_REGION')
```

```
                                {
                                #leave blank
                                }
                                elsif(($tmpWrites1 eq 'WRITE_CRITICAL' || $tmpWrites1 eq 'WRITE_ATOMIC') &&
                                        $regionName eq 'READ_WRITE_SAME_PARALLEL_REGION')
                                {
                                        push(@$msgs, "Read - Write dependancy race can occure for shared variable
name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                                else
                                {
                                        push(@$msgs, "Read - Write dependancy race can occure for shared variable
name '$sharedVar' at line no : $pairs->{'lineno1'} and $pairs->{'lineno2'}");
                                }
                           }
                        }

        }

}



sub combineVariables
{
  my $combinedVar = $_[0];
  my $sharedvar = $_[1];
  my $privatevar = $_[2];
  my $firstprivatevar = $_[3];
  my $lastprivatevar = $_[4];

        foreach $var1 (@$sharedvar)
        {
                push(@{$combinedVar},$var1);
        }
        foreach $var1 (@$privatevar)
        {
                push(@{$combinedVar},$var1);
        }
        foreach $var1 (@$firstprivatevar)
    {
                my $status = 0;
                foreach $var2 (@$combinedVar)
                {
                        if($var2 eq $var1)
                        {
                                $status = 1;
                        }
                }
                if($status != 1)
                {
                        push(@{$combinedVar},$var1);
                }
        }
        foreach $var1 (@$lastprivatevar)
```

```
	{
		my $status = 0;
		foreach $var2 (@$combinedVar)
		{
			if($var2 eq $var1)
			{
				$status = 1;
			}
		}
		if($status != 1)
		{
			push(@{$combinedVar},$var1);
		}
	}
}

sub diagnose
{
	my $data = $_[0];
	my $stQueue = $_[1];
	my $varInfo = $_[2];
	my $combinedVars = $_[3];
	my $sharedVars = $_[4];
	my $privateVars = $_[5];
	my $firstprivateVars = $_[6];
	my $lastprivateVars = $_[7];
      my $readonlyVars = $_[8];
	my $check = 0;

	foreach my $sharedVar (@$sharedVars)
	{
		$check = 0;
		foreach $readonlyVar (@$readonlyVars)
		{
			if($readonlyVar eq $sharedVar)
			{
				$check = 1;
			}
		}
		if($check == 1)
		{
		   next;
		}
	   if(defined($varInfo->{$sharedVar}{'read'}))
	   {
		foreach $readAccess (@{$varInfo->{$sharedVar}{'read'}})
	     {
				$readRegionIndex = $readAccess->{'regionindex'};
				if($stQueue->{$readRegionIndex}{'name'} eq 'MASTER' ||
					$stQueue->{$readRegionIndex}{'name'} eq 'SINGLE' ||
					$stQueue->{$readRegionIndex}{'name'} eq 'PARALLEL' ||
					$stQueue->{$readRegionIndex}{'name'} eq 'PARALLEL_FOR' ||
					$stQueue->{$readRegionIndex}{'name'} eq 'OMP_FOR' )
		  {
```

```
findOutBarriers($data,$stQueue,$varInfo,$sharedVar,$readRegionIndex,$readAccess->{'lineno'},"READ");
                                        $readAccess->{'done'} = 1;

                                }

                                while($stQueue->{$readRegionIndex}{'name'} eq 'CRITICAL' ||
                                        $stQueue->{$readRegionIndex}{'name'} eq 'IF'  ||
                                        $stQueue->{$readRegionIndex}{'name'} eq 'FOR' ||
                                        $stQueue->{$readRegionIndex}{'name'} eq 'ATOMIC')
                                {
                                        $readRegionIndex = findParentRegionIndex($stQueue,
$readRegionIndex);
                                }

                        foreach $writeAccess (@{$varInfo->{$sharedVar}{'write'}})
                        {
                                $writeRegionIndex = $writeAccess->{'regionindex'};

                                if(!defined($writeAccess->{'done'}))
                                {
                                        if(defined($varInfo->{$sharedVar}{'status'}))
                                        {
                                                $cnt = scalar(@{$varInfo->{$sharedVar}{'status'}});
                                        }
                                        else
                                        {
                                                $cnt = 0;
                                        }
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'name'} =
'WRITE_'.$stQueue->{$writeAccess->{'regionindex'}}{'name'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'region'} = $writeAccess-
>{'regionindex'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'lineno'} = $writeAccess-
>{'lineno'};
                                }
                                if(!defined($writeAccess->{'done'}) &&
                                        $stQueue->{$writeRegionIndex}{'name'} ne 'FOR' &&
                                        $stQueue->{$writeRegionIndex}{'name'} ne 'IF')
                        {



        findOutBarriers($data,$stQueue,$varInfo,$sharedVar,$writeRegionIndex,$writeAccess-
>{'lineno'},"WRITE");
                                        $writeAccess->{'done'} = 1;

                                }

                                while($stQueue->{$writeRegionIndex}{'name'} eq 'CRITICAL' ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'IF'  ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'FOR' ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'ATOMIC')
                                {
```

```
                $writeRegionIndex = findParentRegionIndex($stQueue,
$writeRegionIndex);
        }
        if($readAccess->{'lineno'} == $writeAccess->{'lineno'})
        {
                if(defined($varInfo->{$sharedVar}{'status'}))
                {
                        $cnt = scalar(@{$varInfo->{$sharedVar}{'status'}});
                }
                else
                {
                        $cnt = 0;
                }
                $varInfo->{$sharedVar}{'status'}->[$cnt]{'name'} =
'READ_WRITE_SAME_LINE';
                $varInfo->{$sharedVar}{'status'}->[$cnt]{'region'} =
$writeRegionIndex;
                $varInfo->{$sharedVar}{'status'}->[$cnt]{'lineno'} = $writeAccess-
>{'lineno'};
        }
        elsif($readAccess->{'regionindex'} == $writeAccess->{'regionindex'})
        {
                if(defined($varInfo->{$sharedVar}{'pairs'}))
                {
                        $cnt = scalar(@{$varInfo->{$sharedVar}{'pairs'}});
                }
                else
                {
                        $cnt = 0;
                }

                $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_SAME_REGION';
                $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'region1'} = $readAccess-
>{'regionindex'};
                $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'lineno1'} = $readAccess-
>{'lineno'};
                $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'region2'} = $writeAccess-
>{'regionindex'};
                $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'lineno2'} = $writeAccess-
>{'lineno'};


        }
        elsif($readRegionIndex == $writeRegionIndex)
        {
                if(defined($varInfo->{$sharedVar}{'pairs'}))
                {
                        $cnt = scalar(@{$varInfo->{$sharedVar}{'pairs'}});
                }
                else
                {
                        $cnt = 0;
                }
```

```
                                        $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_SAME_PARELLEL_REGION';
                                        $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'region1'} = $readAccess-
>{'regionindex'};
                                        $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'lineno1'} = $readAccess-
>{'lineno'};
                                        $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'region2'} = $writeAccess-
>{'regionindex'};
                                        $varInfo->{$sharedVar}{'pairs'}->[$cnt]{'lineno2'} = $writeAccess-
>{'lineno'};


                                }
                        }
                }
        }
        else
            {
                        foreach $writeAccess (@{$varInfo->{$sharedVar}{'write'}})
                        {
                                $writeRegionIndex = $writeAccess->{'regionindex'};


                while($stQueue->{$writeRegionIndex}{'name'} eq 'CRITICAL' ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'IF'  ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'FOR' ||
                                        $stQueue->{$writeRegionIndex}{'name'} eq 'ATOMIC')
                                {
                                        $writeRegionIndex = findParentRegionIndex($stQueue,
$writeRegionIndex);
                                }

                                if($stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'OMP_FOR' ||
                                        $stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'PARALLEL'
||
                                        $stQueue->{$writeAccess->{'regionindex'}}{'name'} eq
'PARALLEL_FOR')
                                {
                                        if(defined($varInfo->{$sharedVar}{'status'}))
                                        {
                                                $cnt = scalar(@{$varInfo->{$sharedVar}{'status'}});
                                        }
                                        else
                                        {
                                                $cnt = 0;
                                        }
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'name'} = 'WRITE_' .
$stQueue->{$writeAccess->{'regionindex'}}{'name'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'region'} = $writeAccess-
>{'regionindex'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'lineno'} = $writeAccess-
>{'lineno'};
                                }
                                elsif($stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'CRITICAL'  ||
                                        $stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'ATOMIC' ||
```

```
                                        $stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'SINGLE' ||
                                        $stQueue->{$writeAccess->{'regionindex'}}{'name'} eq 'MASTER')
                                {
                                        if(defined($varInfo->{$sharedVar}{'status'}))
                                        {
                                                $cnt = scalar(@{$varInfo->{$sharedVar}{'status'}});
                                        }
                                        else
                                        {
                                                $cnt = 0;
                                        }
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'name'} = 'WRITE_' .
$stQueue->{$writeAccess->{'regionindex'}}{'name'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt]{'region'} = $writeAccess-
>{'regionindex'};
                                        $varInfo->{$sharedVar}{'status'}->[$cnt++]{'lineno'} = $writeAccess-
>{'lineno'};

                                }
                        }
                }
        }

}

sub findOutBarriers
{
        my $data = $_[0];
        my $stQueue = $_[1];
        my $varInfo = $_[2];
        my $sharedV = $_[3];
        my $indx = $_[4];
        my $ln = $_[5];
        my $accessType = $_[6];
        my $parallelStart = -1;
        my $parallelEnd = -1;

        foreach  $key ( sort { $a <=> $b } keys(%$stQueue))
    {
        if($stQueue->{$key}{'name'} eq 'PARALLEL' || $stQueue->{$key}{'name'} eq 'PARALLEL_FOR')
        {
                $parallelStart = $stQueue->{$key}{'start'};
         $parallelEnd = $stQueue->{$key}{'end'};
         last;
        }
    }

        my $stindx = $stQueue->{$indx}{'start'};
        my $endindx = $stQueue->{$indx}{'end'};

        if((($stQueue->{$indx}{'name'} eq 'SINGLE') &&
                (int($data->{$stQueue->{$indx}{'start'}}{'nowait'}) == 1)) ||
                (($stQueue->{$indx}{'name'} eq 'OMP_FOR') &&
                (int($data->{$stQueue->{$indx}{'start'}}{'nowait'}) == 1)) ||
                ($stQueue->{$indx}{'name'} eq 'MASTER') ||
```

```
		($stQueue->{$indx}{'name'} eq 'CRITICAL') ||
		($stQueue->{$indx}{'name'} eq 'ATOMIC'))
	{

		if($accessType eq 'WRITE')
		{
			for(my $i = $stindx-1;$i >= $parallelStart ;$i--)
			{#print ":21:$i";
				$newindx = findRegionIndex($stQueue,$i);
				if(($stQueue->{$newindx}{'name'}  eq   'OMP_FOR') &&
					int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
				{#print ":22:$i";
					last;
				}
				elsif(($stQueue->{$newindx}->{'name'} eq 'SINGLE') &&
					int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
				{#print ":23:$i";
					last;
				}
				elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
				{
					last;
				}
				elsif(($data->{$i}{'type'} eq 'FOR') ||
				($data->{$i}{'type'} eq 'OMP_FOR' && defined($data->{$i}{'datatype'}) &&
$data->{$i}{'datatype'} eq 'loop') ||
				 ($data->{$i}{'type'} eq 'ASSIGNMENT') ||
				 ($data->{$i}{'type'} eq 'OUTPUT'))
				{#print ":24:$i";
					foreach $readVar (@{$data->{$i}{'read'}})
					{#print ":25:$i";
						if($readVar->{'name'} eq $sharedV)
						{#print ":26:$i";
							$cnt = 0;
							if(defined($varInfo->{$sharedV}{'pairs'}))
							{
								$cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
							}

							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_DEPENDANCY';
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;
					#		push(@{$varInfo->{$sharedV}{'Warning'}}, "Read -
Write Dependancy for varaible $sharedV at lines $ln and $i");
						}
					}
					foreach $writeVar (@{$data->{$i}{'write'}})
```

```
                                        {#print ":27:$i";
                                                if($writeVar->{'name'} eq $sharedV)
                                                {#print ":28:$i";
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_WRITE_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                }
                                        }
                                }


                        }
                        for($i = $endindx+1;$i <= $parallelEnd ;$i++)
                        {
                                $newindx = findRegionIndex($stQueue,$i);
                                if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                        int($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                        $stQueue->{$newindx}{'end'}) == $i)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                        int($stQueue->{$newindx}{'end'}) == $i)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                {
                                        last;
                                }
                                elsif($data->{$i}{'type'} eq 'FOR' ||
                                        ($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                        $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                        $data->{$i}{'type'} eq 'OUTPUT')
                                {
                                        foreach $readVar (@{$data->{$i}{'read'}})
```

```
                                        {
                                                if($readVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_READ_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;

                                                }
                                        }
                                        foreach $writeVar (@{$data->{$i}{'write'}})
                                        {
                                                if($writeVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_WRITE_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;

                                                }
                                        }
                                }

                        }
                }
                else
                {
                        for($i = $stindx-1;$i >= $parallelStart ;$i--)
                        {
```

```
                                        $newindx = findRegionIndex($stQueue,$i);
#print " :$newindx:$i: " . $stQueue->{$newindx}->{'name'} . " : ". $data->{$stQueue-
>{$newindx}{'start'}}{'nowait'} ."\n";

                                        if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                                $data->{$stQueue->{$newindx}{'start'}}{'nowait'} eq '0')
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                        {
                                                last;
                                        }
                                        elsif($data->{$i}{'type'} eq 'FOR' ||
                                                ($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                                $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                                $data->{$i}{'type'} eq 'OUTPUT')
                                        {

                                                foreach $writeVar (@{$data->{$i}{'write'}})
                                                {
                                                        if($writeVar->{'name'} eq $sharedV)
                                                        {
                                                                $cnt = 0;
                                                                if(defined($varInfo->{$sharedV}{'pairs'}))
                                                                {
                                                                        $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                                }

                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_READ_DEPENDANCY';
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                        }
                                                }
                                        }

                                }
                                for($i = $endindx+1;$i <= $parallelEnd ;$i++)
                                {
                                        $newindx = findRegionIndex($stQueue,$i);
```

```
                                        if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                                 $stQueue->{$newindx}{'end'} == $i)
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                                $stQueue->{$newindx}{'end'} == $i)
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                        {
                                                last;
                                        }
                                        elsif($data->{$i}{'type'} eq 'FOR' ||
                                                ($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                                $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                                $data->{$i}{'type'} eq 'OUTPUT')
                                        {

                                                foreach $writeVar (@{$data->{$i}{'write'}})
                                                {
                                                        if($writeVar->{'name'} eq $sharedV)
                                                        {
                                                                $cnt = 0;
                                                                if(defined($varInfo->{$sharedV}{'pairs'}))
                                                                {
                                                                        $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                                }

                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_DEPENDANCY';
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;

                                                        }
                                                }
                                        }

                                }
                        }

                }
                elsif(($stQueue->{$indx}{'name'} eq 'SINGLE' &&
                        int($data->{$stQueue->{$indx}{'start'}}{'nowait'}) == 0) ||
```

```
		$stQueue->{$indx}{'name'} eq 'OMP_FOR' &&
		int($data->{$stQueue->{$indx}{'start'}}{'nowait'}) == 0)
	{
		if($accessType eq 'WRITE')
		{
			for(my $i = $stindx-1;$i >= $parallelStart ;$i--)
			{
				$newindx = findRegionIndex($stQueue,$i);
#print Dumper($stQueue->{$newindx});
#print Dumper($data->{$stQueue->{$newindx}{'start'}});
#print Dumper($data->{$i});
				if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
					int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
				{
					last;
				}
				elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
					int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
				{
					last;
				}
				elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
				{
					last;
				}
				elsif(($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data->{$i}{'datatype'})
&& $data->{$i}{'datatype'} eq 'loop') ||
					$data->{$i}{'type'} eq 'FOR' ||
					$data->{$i}{'type'} eq 'ASSIGNMENT' ||
					$data->{$i}{'type'} eq 'OUTPUT')
				{
					foreach $readVar (@{$data->{$i}{'read'}})
					{
						if($readVar->{'name'} eq $sharedV)
						{
							$cnt = 0;
							if(defined($varInfo->{$sharedV}{'pairs'}))
							{
								$cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
							}

							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_DEPENDANCY';
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
							$varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

						}
					}
```

```
                                        foreach $writeVar (@{$data->{$i}{'write'}})
                                        {
                                                if($writeVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_WRITE_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                }
                                        }
                                }

                        }

                }
                else
                {
                        for(my $i = $stindx-1;$i >= $parallelStart ;$i--)
                        {
                                $newindx = findRegionIndex($stQueue,$i);
                                if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                {
                                        last;
                                }
                                elsif($data->{$i}{'type'} eq 'FOR' ||
                                        ($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                        $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                        $data->{$i}{'type'} eq 'OUTPUT')
                                {
```

```
                                        foreach $writeVar (@{$data->{$i}{'write'}})
                                        {
                                                if($writeVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_READ_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                }
                                        }
                                }

                        }

                }

        }
        elsif($stQueue->{$indx}{'name'} eq 'PARALLEL' || $stQueue->{$indx}{'name'} eq 'PARALLEL_FOR')
    {
                if($accessType eq 'WRITE')
                {
                        for( $i = $ln-1;$i >= $parallelStart ;$i--)
                        {
                                $newindx = findRegionIndex($stQueue,$i);
                                if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' && int($data->{$stQueue-
>{$newindx}{'start'}}{'nowait'}) == 0)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' && int($data->{$stQueue-
>{$newindx}{'start'}}{'nowait'}) == 0)
                                {
                                        last;
                                }
                                elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                {
                                        last;
                                }
                                elsif($data->{$i}{'type'} eq 'FOR' ||
                                        ($data->{$i}{'type'} eq 'OMP_FOR' &&  defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
```

```
                                        $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                        $data->{$i}{'type'} eq 'OUTPUT')
                                {
                                        foreach $readVar (@{$data->{$i}{'read'}})
                                        {
                                                if($readVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                }
                                        }
                                        foreach $writeVar (@{$data->{$i}{'write'}})
                                        {
                                                if($writeVar->{'name'} eq $sharedV)
                                                {
                                                        $cnt = 0;
                                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                                        {
                                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                        }

                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_WRITE_DEPENDANCY';
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                                }
                                        }
                                }

                        }
                        for($i = $ln+1;$i <= $parallelEnd ;$i++)
```

```
                {
                        $newindx = findRegionIndex($stQueue,$i);
                        if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                int($stQueue->{$newindx}{'end'}) == $i)
                        {
                                last;
                        }
                        elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                                int($stQueue->{$newindx}{'end'}) == $i)
                        {
                                last;
                        }
                        elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                        {
                                last;
                        }
                        elsif($data->{$i}{'type'} eq 'FOR' ||
                                ($data->{$i}{'type'} eq 'OMP_FOR' && defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                $data->{$i}{'type'} eq 'OUTPUT')
                        {
                                foreach $readVar (@{$data->{$i}{'read'}})
                                {
                                        if($readVar->{'name'} eq $sharedV)
                                        {
                                                $cnt = 0;
                                                if(defined($varInfo->{$sharedV}{'pairs'}))
                                                {
                                                        $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                }

                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_READ_DEPENDANCY';
                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;
                                        #       push(@{$varInfo->{$sharedV}{'Warning'}}, "Read -
Write Dependancy for varaible $sharedV at lines $i and $ln");
                                        }
                                }
                                foreach $writeVar (@{$data->{$i}{'write'}})
                                {
                                        if($writeVar->{'name'} eq $sharedV)
                                        {
                                                $cnt = 0;
                                                if(defined($varInfo->{$sharedV}{'pairs'}))
```

```
                                                                {
                                                                        $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                                                }

                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_WRITE_DEPENDANCY';
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                                $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;

                                                        }
                                                }
                                        }

                                }
                        }
                        else
                        {
                                for( $i = $stindx-1;$i >= $parallelStart ;$i--)
                                {
                                        $newindx = findRegionIndex($stQueue,$i);
                                        if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                                                int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0)
                                        {
                                                last;
                                        }
                                        elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                                        {
                                                last;
                                        }
                                        elsif($data->{$i}{'type'} eq 'FOR' ||
                                                ($data->{$i}{'type'} eq 'OMP_FOR' && defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                                                $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                                                $data->{$i}{'type'} eq 'OUTPUT')
                                        {

                                                foreach $writeVar (@{$data->{$i}{'write'}})
                                                {
                                                        if($writeVar->{'name'} eq $sharedV)
                                                        {
                                                                $cnt = 0;
                                                                if(defined($varInfo->{$sharedV}{'pairs'}))
                                                                {
```

```
                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
                                        }

                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'WRITE_READ_DEPENDANCY';
                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$newindx;
                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$i;
                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$indx;
                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$ln;

                                }
                        }
                }

        }
        for($i = $endindx+1;$i <= $parallelEnd ;$i++)
        {
                $newindx = findRegionIndex($stQueue,$i);
                if($stQueue->{$newindx}{'name'} eq 'OMP_FOR' &&
                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                        $stQueue->{$newindx}{'end'} == $i)
                {
                        last;
                }
                elsif($stQueue->{$newindx}{'name'} eq 'SINGLE' &&
                        int($data->{$stQueue->{$newindx}{'start'}}{'nowait'}) == 0 &&
                        $stQueue->{$newindx}{'end'} == $i)
                {
                        last;
                }
                elsif($stQueue->{$newindx}->{'name'} eq 'BARRIER')
                {
                        last;
                }
                elsif($data->{$i}{'type'} eq 'FOR' ||
                        ($data->{$i}{'type'} eq 'OMP_FOR' && defined($data-
>{$i}{'datatype'}) && $data->{$i}{'datatype'} eq 'loop') ||
                        $data->{$i}{'type'} eq 'ASSIGNMENT' ||
                        $data->{$i}{'type'} eq 'OUTPUT')
                {

                        foreach $writeVar (@{$data->{$i}{'write'}})
                        {
                                if($writeVar->{'name'} eq $sharedV)
                                {
                                        $cnt = 0;
                                        if(defined($varInfo->{$sharedV}{'pairs'}))
                                        {
                                                $cnt = scalar(@{$varInfo-
>{$sharedV}{'pairs'}});
```

```
                                                                        }

                                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'name'} =
'READ_WRITE_DEPENDANCY';
                                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region1'} =
$indx;
                                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno1'} =
$ln;
                                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'region2'} =
$newindx;
                                                                        $varInfo->{$sharedV}{'pairs'}->[$cnt]{'lineno2'} =
$i;

                                                                }
                                                        }
                                                }

                                        }
                                }

                        }

}
sub getVariableDetails
{
        my $structQueue = $_[1];
        my $data = $_[0];
        my $parallelStart = -1;
    my $parallelEnd = -1;
        my $varInfo = $_[2];
    my $key;
    my $cnt = 0;
        foreach  $key ( sort { $a <=> $b } keys(%$structQueue))
     {
        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
        {
                $parallelStart = $structQueue->{$key}{'start'};
         $parallelEnd = $structQueue->{$key}{'end'};
         last;
        }
      }
                for(my $i = $parallelStart; $i <= $parallelEnd;$i++)
         {#print "$i\n";
                        if(defined($data->{$i}{'read'}))
                        {
                                foreach $readVar (@{$data->{$i}{'read'}})
                                {
                                        if(defined($varInfo->{$readVar->{'name'}}{'read'}))
                                        {
                                                $cnt = scalar(@{$varInfo->{$readVar->{'name'}}{'read'}})
                                    }
                                        else
                                        {
                                                $cnt = 0;
```

```
                                        }
                                        $varInfo->{$readVar->{'name'}}{'read'}->[$cnt]{'regionindex'}
=findRegionIndex($structQueue,$i);
                                        $varInfo->{$readVar->{'name'}}{'read'}->[$cnt]{'lineno'} = $i;

                                }
                        }
                        if(defined($data->{$i}{'write'}))
                        {
                                foreach $writeVar (@{$data->{$i}{'write'}})
                                {
                                        if(defined($varInfo->{$writeVar->{'name'}}{'write'}))
                                        {
                                                $cnt = scalar(@{$varInfo->{$writeVar->{'name'}}{'write'}})
                                    }
                                        else
                                        {
                                                $cnt = 0;
                                        }
                                        $varInfo->{$writeVar->{'name'}}{'write'}->[$cnt]{'regionindex'}
=findRegionIndex($structQueue,$i);
                                        $varInfo->{$writeVar->{'name'}}{'write'}->[$cnt]{'lineno'} = $i;
                                }
                        }
                }

}

sub findRegionIndex
{
        my $stQueue = $_[0];
        my $indx = $_[1];
        foreach  my $k ( sort { $a <=> $b } keys(%$stQueue))
    {
        if(int($stQueue->{$k}{'start'}) <= int($indx) && int($stQueue->{$k}{'end'}) >= int($indx) )
                {
                        return $k;
                }
    }
        return -1;
}

sub findParentRegionIndex
{
        my $stQueue = $_[0];
        my $indx = $_[1];
        foreach  my $k ( sort { $a <=> $b } keys(%$stQueue))
    {
        if(int($stQueue->{$k}{'start'}) < int($stQueue->{$indx}{'start'}) && int($stQueue->{$k}{'end'}) >
int($stQueue->{$indx}{'end'}) )
                {
                        return $k;
                }
    }
        return -1;
```

```
}

sub atomicStatement
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $top = $_[4];

    $top++;
    $line =~ /^#pragma\s+omp\s+atomic(.*)/;
    if(defined($1))
    {
        $data->{$lineno}{'type'} = 'ATOMIC';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}{'name'} = 'ATOMIC';
        $structStack->{$top}{'start'} = $lineno;
        $nextElement = '';
    }
    return ($top,$nextElement);
}

sub lockStatement
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $top = $_[4];

    $top++;
    $line =~ /^omp_set_lock\s*\(&(.*)\)/;
    if(defined($1))
    {
        $data->{$lineno}{'type'} = 'SIMPLE_LOCK';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $top++;
        $structStack->{$top}->{'name'} = 'SIMPLE_LOCK';
        $structStack->{$top}->{'start'} = $lineno;
        $structStack->{$top}->{'varname'} = trim($1);
        $nextElement = '';
    }
    return ($top,$nextElement);

}

sub masterStatement
{
```

```
    my $line = $_[1];
    my $data = $_[0];
    my $lineno = $_[2];
    my $cnt = 0;
    my $structStack = $_[3];
    my $nextElement = '';
    my $top = $_[4];

     $top++;
     $line =~ /^#pragma\s+omp\s+master(.*)/;
    #  if(defined($1))
    #  {
            $data->{$lineno}{'type'} = 'MASTER';
                $data->{$lineno}{'lineno'} = $lineno;
                $data->{$lineno}{'statement'} = $line;
            $structStack->{$top}->{'name'} = 'MASTER';
            $structStack->{$top}->{'start'} = $lineno;
            $nextElement = '{?';
     # }
     return ($top,$nextElement);
}

sub singleStatement
{
    my $line = $_[1];
    my $data = $_[0];
    my $lineno = $_[2];
    my $cnt = 0;
    my $structStack = $_[3];
    my $nextElement = '';
    my $top = $_[4];

     $top++;
     $line =~ /^#pragma\s+omp\s+single(.*)/;

     $data->{$lineno}{'type'} = 'SINGLE';
     $data->{$lineno}{'lineno'} = $lineno;
     $data->{$lineno}{'statement'} = $line;
     $structStack->{$top}->{'name'} = 'SINGLE';
     $structStack->{$top}->{'start'} = $lineno;
     $nextElement = '{?';

     if(defined($1))
     {
          $lineElements = $1;
               if($lineElements =~ /nowait/)
               {
            $data->{$lineno}{'nowait'} = 1;
               }
               else
               {
            $data->{$lineno}{'nowait'} = 0;
          }
     }
     else
```

```
    {
            $data->{$lineno}{'nowait'} = 0;
    }

    return ($top,$nextElement);
}

sub criticalStatement
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $top = $_[4];

    $top++;
    $line =~ /^#pragma\s+omp\s+critical\s+\(?(.*?)\)?/;
    if(defined($1))
    {
        $data->{$lineno}{'type'} = 'CRITICAL';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'CRITICAL';
        $structStack->{$top}->{'start'} = $lineno;
        $structStack->{$top}->{'regionname'} = trim($1);
        $nextElement = '{?';
    }
    else
    {
        $data->{$lineno}{'type'} = 'CRITICAL';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'CRITICAL';
        $structStack->{$top}->{'start'} = $lineno;
        $structStack->{$top}->{'regionname'} = 'ANONYMOUS';
        $nextElement = '{?';
    }
    return ($top,$nextElement);
}

sub forStatement
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $top = $_[4];

    $top++;
    $line =~ /^#pragma\s+omp\s+for(.*)/;
        $data->{$lineno}{'type'} = 'OMP_FOR';
```

```
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'OMP_FOR';
        $structStack->{$top}->{'start'} = $lineno;
        $nextElement = 'for\(';

 $lineElements = $1;
if($lineElements =~ /shared\((.*?)\)/)
{
        @sharedVariables = split(/\s*,\s*/, $1);
        $cnt = defined(@{$data->{$lineno}{'shared'}}) ? scalar(@{$data->{$lineno}{'shared'}}) : 0;
        foreach $sharedVariable (@sharedVariables)
        {
           $data->{$lineno}{'shared'}[$cnt++]{'name'} = $sharedVariable;
        }

}
if($lineElements =~ /private\((.*?)\)/)
{
        @privateVariables = split(/\s*,\s*/, $1);
            $cnt = defined(@{$data->{$lineno}{'private'}}) ? scalar(@{$data->{$lineno}{'private'}}) : 0;
        foreach $privateVariable (@privateVariables)
        {
           $data->{$lineno}{'private'}[$cnt++]{'name'} = $privateVariable;
        }

}
if($lineElements =~ /default\((.*?)\)/)
{
        $data->{$lineno}{'default'} = $1;
}

if($lineElements =~ /firstprivate\((.*?)\)/)
{
        @privateVariables = split(/\s*,\s*/, $1);
            $cnt = defined(@{$data->{$lineno}{'firstprivate'}}) ? scalar(@{$data->{$lineno}{'firstprivate'}}) : 0;
        foreach $privateVariable (@privateVariables)
        {
           $data->{$lineno}{'firstprivate'}[$cnt++]{'name'} = $privateVariable;
        }

}
if($lineElements =~ /lastprivate\((.*?)\)/)
{
        @privateVariables = split(/\s*,\s*/, $1);
        $cnt = defined(@{$data->{$lineno}{'lastprivate'}}) ? scalar(@{$data->{$lineno}{'lastprivate'}}) : 0;
        foreach $privateVariable (@privateVariables)
        {
           $data->{$lineno}{'lastprivate'}[$cnt++]{'name'} = $privateVariable;
        }

}
if($lineElements =~ /nowait/)
{
        $data->{$lineno}{'nowait'} = 1;
```

```
    }
    else
    {
        $data->{$lineno}{'nowait'} = 0;
    }
    if($lineElements =~ /reduction\((.*?)\)/)
    {
        @reductionElements = split(/\s*:\s*/, $1);
            $cnt = defined(@{$data->{$lineno}{'reduction'}}) ? scalar(@{$data->{$lineno}{'reduction'}}) : 0;
        if(scalar(@reductionElements))
        {
            $cnt1 = 0;
            $operator = $reductionElements[0];
            $data->{$lineno}{'reduction'}[$cnt]{'operator'} = $operator;
            @reductionVariables = split(/\s*,\s*/, trim($reductionElements[1]));
                    foreach $reductionVariable (@reductionVariables)
          {
                    $data->{$lineno}{'reduction'}[$cnt]{'variables'}[$cnt1++]{'name'} = trim($reductionVariable);
          }

        }
    }


    return ($top,$nextElement);
}

sub barrierStatement
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $structQueue = $_[5];
   my $top = $_[4];
    $top++;
    $line =~ /^#pragma\s+omp\s+barrier(.*)/;
        $data->{$lineno}{'type'} = 'BARRIER';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'BARRIER';
        $structStack->{$top}->{'start'} = $lineno;
            $structStack->{$top}->{'end'} = $lineno;
            $nextQueueIndex = scalar(keys(%$structQueue));

            my $nextQueueIndex = scalar(keys(%$structQueue));
        foreach my $key (keys(%{$structStack->{$top}}))
        {
         $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
        }
        $top--;
        $nextElement = '';
```

```
      $lineElements = $1;
     return ($top,$nextElement);
}


sub controlStructure
{
   my $line = $_[1];
   my $data = $_[0];
   my $lineno = $_[2];
   my $cnt = 0;
   my $structStack = $_[3];
   my $nextElement = '';
   my $top = $_[4];

   if($line =~ /^for\((.*?)\)$/)
   {
        $data->{$lineno}{'type'} = 'FOR';
             $data->{$lineno}{'datatype'} = 'loop';
             $data->{$lineno}{'lineno'} = $lineno;
             $data->{$lineno}{'statement'} = $line;
             $data->{$lineno}{'statementline'} = $1;
        @statements = split(/\s*;\s*/,$1);
             $data->{$lineno}{'initialization'}{'content'} = trim($statements[0]);
        $data->{$lineno}{'condition'}{'content'} = trim($statements[1]);
        $data->{$lineno}{'mutator'}{'content'} = trim($statements[2]);
        if($lineno != ($structStack->{$top}{'start'}+1) && ($structStack->{$top}{'name'} ne 'PARALLEL_FOR'
&& $structStack->{$top}{'name'} ne 'OMP_FOR'))
        {
             $top++;
                  $structStack->{$top}->{'name'} = 'FOR';
             $structStact->{$top}->{'start'} = $lineno;
             $nextElement = '{?';
        }
        else
             {
                  $status = 0;
                  @priv = split(/\s+/,$statements[0]);
             $readCnt = 0;
             for($i = 2; $i < scalar(@priv); $i+=2 )
             {
                   $status = 0;
                   $data->{$lineno}{'initialization'}{'read'}->[$readCnt]{'name'} = trim($priv[$i]);
                        if(defined($data->{$lineno}{'read'}))
                        {
                                  $readLength = scalar(@{$data->{$lineno}{'read'}});
                                  foreach $varName (@{$data->{$lineno}{'read'}})
                    {
                                       if($varName eq $priv[$i])
                                       {
                                            $status = 1;
                                       }
                                  }
                                  if($status != 1)
                                  {
```

```
                                        $data->{$lineno}{'read'}->[$readLength]{'name'} = $priv[$i];
                                }
                        }
                        else
                        {
                                $data->{$lineno}{'read'}->[0]{'name'} = $priv[$i];
                        }

                }
                $data->{$lineno}{'initialization'}{'write'}->[0]{'name'} = trim($priv[0]);
                $data->{$lineno}{'mutator'}{'write'}->[0]{'name'} = trim($priv[0]);
                if(defined($data->{$lineno}{'write'}))
                {
                        $status = 0;
                        $readLength = scalar(@{$data->{$lineno}{'write'}});
                        foreach $varName (@{$data->{$lineno}{'write'}})
        {

                                if($varName eq $priv[0])
                                {
                                        $status = 1;
                                }
                        }
                        if($status != 1)
                        {
                                $data->{$lineno}{'write'}->[$readLength]{'name'} = $priv[0];
                        }
                }
                else
                {
                        $data->{$lineno}{'write'}->[0]{'name'} = $priv[0];
                }
        $status = 0;
        foreach $privateVariable (@{$data->{$structStack->{$top}->{'start'}}{'private'}})
         {
                        if($priv[0] eq $privateVariable->{'name'})
             {
                                $status = 1;
                        }
         }
                if($status != 1)
        {
                        $data->{$structStack->{$top}->{'start'}}{'private'}->[scalar(@{$data->{$structStack-
>{$top}->{'start'}}{'private'}})]{'name'} = $priv[0];
                        $data->{$structStack->{$top}->{'start'}}{'mustprivate'}->[scalar(@{$data-
>{$structStack->{$top}->{'start'}}{'mustprivate'}})]{'name'} = $priv[0];
                }

                $status = 0;
                @priv = split(/\s+/,$statements[1]);
        $readCnt = 0;
        for($i = 0; $i < scalar(@priv); $i+=2 )
        {
                 $status = 0;
                 $data->{$lineno}{'condition'}{'read'}->[$readCnt]{'name'} = trim($priv[$i]);
```

```
                        if(defined($data->{$lineno}{'read'}))
                        {
                                $readLength = scalar(@{$data->{$lineno}{'read'}});
                                foreach $varName (@{$data->{$lineno}{'read'}})
                {
                                        if($varName eq $priv[$i])
                                        {
                                                $status = 1;
                                        }
                                }
                                if($status != 1)
                                {
                                        $data->{$lineno}{'read'}->[$readLength]{'name'} = $priv[$i];
                                }
                        }
                        else
                        {
                                $data->{$lineno}{'read'}->[0]{'name'} = $priv[$i];
                        }
                }
                $status = 0;
                @priv = split(/\s+/,$statements[2]);
        $readCnt = 0;
        for($i = 2; $i < scalar(@priv); $i+=2 )
        {
                 $status = 0;
                 $data->{$lineno}{'mutator'}{'read'}->[$readCnt]{'name'} = trim($priv[$i]);
                        if(defined($data->{$lineno}{'read'}))
                        {
                                $readLength = scalar(@{$data->{$lineno}{'read'}});
                                foreach $varName (@{$data->{$lineno}{'read'}})
                 {
                                        if($varName eq $priv[$i])
                                        {
                                                $status = 1;
                                        }
                                }
                                if($status != 1)
                                {
                                        $data->{$lineno}{'read'}->[$readLength]{'name'} = $priv[$i];
                                }
                        }
                        else
                        {
                                $data->{$lineno}{'read'}->[0]{'name'} = $priv[$i];
                        }
                }
        }

}
elsif($line =~ /^if\((.*?)\)$/)
{
    $data->{$lineno}{'type'} = 'IF';
        $data->{$lineno}{'datatype'} = 'condition';
        $data->{$lineno}{'lineno'} = $lineno;
```

```
            $data->{$lineno}{'statement'} = $line;
            $data->{$lineno}{'statementline'} = $1;
        @statements = split(/\s+/,$1);
            $data->{$lineno}{'variables'}[0]['name'] = trim($statement[0]);
        $data->{$lineno}{'operator'} = trim($statement[1]);
        $data->{$lineno}{'variables'}[1]['name'] = trim($statement[2]);
        if($lineno != ($structStack->{$top}{'start'}-1))
        {
            $top++;
                    $structStack->{$top}->{'name'} = 'IF';
            $structStact->{$top}->{'start'} = $lineno;
            $nextElement = '{?';
        }
    }
    return ($top,$nextElement);
}

sub normalStatement
{
    my $line = $_[1];
    my $data = $_[0];
    my $lineno = $_[2];
    my $cnt = 0;
    my $structStack = $_[3];
    my $nextElement = '';
    my $top = $_[4];
    my $structQueue = $_[5];
    if($line =~ /^(omp_destroy_lock|omp_init_lock)\s+\(&(.*)\)?;$/)
    {
            if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC')
        {
            }

        if($top>=0 && $structStack->{$top}->{'name'} eq 'CRITICAL'  && $lineno == ($structStack-
>{$top}{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
              }
         elsif($top>=0 && ($structStack->{$top}->{'name'} eq 'FOR' || $structStack->{$top}->{'name'} eq 'IF')  &&
$lineno == ($structStack->{$top}->{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
              }
```

```
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'OMP_FOR'  && $lineno == ($structStack-
>{$top}->{'start'} + 2))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'MASTER'  && $lineno == ($structStack-
>{$top}->{'start'} + 1) )
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'SINGLE'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }

       $data->{$lineno}{'type'} = uc($1);
            $data->{$lineno}{'datatype'} = 'omp_lock_t';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
            $data->{$lineno}{'variables'}[$cnt]{'name'} = $2;
       if($1 == 'omp_init_lock')
       {
         $data->{$lineno}{'variables'}[$cnt++]{'value'} = 'DEFINED';
       }
       else
       {
             $data->{$lineno}{'variables'}[$cnt++]{'value'} = 'UNDEFINED';
           }
  }
  elsif($line =~ /^printf\(".*?",?(.*?)\);$/)
  {
           if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC')
       {
        #  print "Error : Atomic statement should be here";
           }
```

```
            if($top>=0 && $structStack->{$top}->{'name'} eq 'CRITICAL'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
       elsif($top>=0 && ($structStack->{$top}->{'name'} eq 'FOR' || $structStack->{$top}->{'name'} eq 'IF')  &&
$lineno == ($structStack->{$top}->{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'OMP_FOR'  && $lineno == ($structStack-
>{$top}->{'start'} + 2))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'MASTER'  && $lineno == ($structStack-
>{$top}->{'start'} + 1) )
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'SINGLE'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
```

```
        $data->{$lineno}{'type'} = 'OUTPUT';
             $data->{$lineno}{'datatype'} = 'printf';
             $data->{$lineno}{'lineno'} = $lineno;
             $data->{$lineno}{'statement'} = $line;

        if(defined($1))
        {
                        $data->{$lineno}{'variablesline'} = $1;
            @outputVariables = split(/\s*,\s*/,$1);
            $cnt = 0;
            foreach $outputVariable (@outputVariables)
            {
            $data->{$lineno}{'read'}->[$cnt]{'name'} = $outputVariable;
                                $data->{$lineno}{'read'}->[$cnt++]{'value'} = '';
            }
        }
    }
    elsif($line =~ /^scanf\(".*?",(.*?)\);$/)
    {

             if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC')
        {
         #  print "Error : Atomic statement should be here";
              }

               if($top>=0 && $structStack->{$top}->{'name'} eq 'CRITICAL'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
           {
            $structStack->{$top}->{'end'} = $lineno;
                 $nextQueueIndex = scalar(keys(%$structQueue));
            foreach $key (keys(%{$structStack->{$top}}))
            {
             $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
            }
                 $top--;
                }
          elsif($top>=0 && ($structStack->{$top}->{'name'} eq 'FOR' || $structStack->{$top}->{'name'} eq 'IF')  &&
$lineno == ($structStack->{$top}->{'start'} + 1))
           {
            $structStack->{$top}->{'end'} = $lineno;
                 $nextQueueIndex = scalar(keys(%$structQueue));
            foreach $key (keys(%{$structStack->{$top}}))
            {
             $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
            }
                 $top--;
                }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'OMP_FOR'  && $lineno == ($structStack-
>{$top}->{'start'} + 2))
           {
            $structStack->{$top}->{'end'} = $lineno;
                 $nextQueueIndex = scalar(keys(%$structQueue));
            foreach $key (keys(%{$structStack->{$top}}))
            {
```

```
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'MASTER'  && $lineno == ($structStack-
>{$top}->{'start'} + 1) )
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'SINGLE'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }

        $data->{$lineno}{'type'} = 'INPUT';
             $data->{$lineno}{'datatype'} = 'scanf';
             $data->{$lineno}{'lineno'} = $lineno;
             $data->{$lineno}{'statement'} = $line;
             $data->{$lineno}{'variablesline'} = $1;
        @inputVariables = split(/\s*,\s*/,$1);
        $cnt = 0;
        foreach $inputVariable (@inputVariables)
        {
          substr($inputVariable,0,1,'');
          $data->{$lineno}{'write'}->[$cnt]{'name'} = $inputVariable;
          $data->{$lineno}{'write'}->[$cnt++]{'value'} = 'NONDETERMINISTIC';
        }
    }
    elsif($line =~ /^(\w+\[?[\w\d]*\]?)\s*(\=|\+\=|\-\=|\/\=|\*\=)\s*(.*?);$/)
    {
                if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC'  && trim($2) eq '=')
          {
                }
#print $structStack->{$top}{'name'} .' ' . $top . ' ' . $structStack->{$top}{'start'} . ' ' . $2 . "\n";

          if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC' )
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
```

```
            }
                $top--;
               }
          elsif($top>=0 && $structStack->{$top}->{'name'} eq 'CRITICAL'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
    elsif($top>=0 && ($structStack->{$top}->{'name'} eq 'FOR' || $structStack->{$top}->{'name'} eq 'IF')  &&
$lineno == ($structStack->{$top}->{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'OMP_FOR'  && $lineno == ($structStack-
>{$top}->{'start'} + 2))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'MASTER'  && $lineno == ($structStack-
>{$top}->{'start'} + 1) )
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
           }
                $top--;
               }
               elsif($top>=0 && $structStack->{$top}->{'name'} eq 'SINGLE'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
          {
           $structStack->{$top}->{'end'} = $lineno;
                $nextQueueIndex = scalar(keys(%$structQueue));
           foreach $key (keys(%{$structStack->{$top}}))
           {
            $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
```

```
        }
            $top--;
           }

           $variables = $3;

           $data->{$lineno}{'type'} = 'ASSIGNMENT';
           $data->{$lineno}{'lineno'} = $lineno;
           $data->{$lineno}{'statement'} = $line;
           $data->{$lineno}{'variablesline'} = $variables;
           $data->{$lineno}{'write'}[0]{'name'} = trim($1);
      $data->{$lineno}{'write'}[0]{'value'} = $variables;
           $data->{$lineno}{'operator'} = trim($2);

           $cnt = 0;

           if($2 eq '+=' || $2 eq '-=' || $2 eq '*=' || $2 eq '/=')
           {
                   $data->{$lineno}{'read'}[$cnt]{'name'} = trim($1);
                   $data->{$lineno}{'read'}[$cnt++]{'value'} = '';
           }

           @variable = split(/\s+/,trim($variables));
           @variableNames = grep(/[\d\.\w\[\]]+/,@variable);

           foreach $variableName (@variableNames)
           {
             $data->{$lineno}{'read'}[$cnt]{'name'} = $variableName;
         if($variableName =~ /^(\d+\.?\d*|\.\d+)$/)
                   {
                           $data->{$lineno}{'read'}[$cnt++]{'value'} = $variableName;
                   }
              else
          {
                           $data->{$lineno}{'read'}[$cnt++]{'value'} = '';
          }
             }
             @operatorNames = grep(/[^\s\d\.\w\[\]]+/,@variable);
             $cnt = 0;
             foreach $operatorName (@operatorNames)
             {
               $data->{$lineno}{'operators'}[$cnt++]{'name'} = $operatorName;
             }
}
elsif($line =~ /^(\w+\[?[\w\d]*\]?)(\+\+|\-\-)\s*(.*?);$/)
{
           if($top>=0 && $structStack->{$top}->{'name'} eq 'ATOMIC')
       {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
```

```
            }
        elsif($top>=0 && $structStack->{$top}->{'name'} eq 'CRITICAL'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
        elsif($top>=0 && ($structStack->{$top}->{'name'} eq 'FOR' || $structStack->{$top}->{'name'} eq 'IF')  &&
$lineno == ($structStack->{$top}->{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'OMP_FOR'  && $lineno == ($structStack-
>{$top}->{'start'} + 2))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'MASTER'  && $lineno == ($structStack-
>{$top}->{'start'} + 1) )
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
            }
            elsif($top>=0 && $structStack->{$top}->{'name'} eq 'SINGLE'  && $lineno == ($structStack->{$top}-
>{'start'} + 1))
        {
         $structStack->{$top}->{'end'} = $lineno;
              $nextQueueIndex = scalar(keys(%$structQueue));
         foreach $key (keys(%{$structStack->{$top}}))
         {
          $structQueue->{$nextQueueIndex}->{$key} = $structStack->{$top}->{$key};
         }
              $top--;
```

```
                }

                $variables = $1;
                $data->{$lineno}{'type'} = 'ASSIGNMENT';
                $data->{$lineno}{'lineno'} = $lineno;
                $data->{$lineno}{'statement'} = $line;
                $data->{$lineno}{'variablesline'} = $variables;
                $data->{$lineno}{'write'}[0]{'name'} = trim($1);
        $data->{$lineno}{'write'}[0]{'value'} = trim($1);
                $data->{$lineno}{'read'}[0]{'name'} = trim($1);
        $data->{$lineno}{'read'}[0]{'value'} = '';
                $data->{$lineno}{'operator'} = trim($2);
    }
    else
    {
      if($structStack->{$top}->{'name'} eq 'ATOMIC')
      {
      }
     }
     return ($top,'');
}


sub declarationStatements
{
    my $line = $_[1];
    my $data = $_[0];
    my $lineno = $_[2];
    my $cnt = 0;
    $line =~ /^(int|omp_lock_t)\s+(.*)?;$/;
    $variables = $2;
    @variable = split(/,/,$variables);
    $data->{$lineno}{'type'} = 'DECLARATION';
    $data->{$lineno}{'datatype'} = $1;
    $data->{$lineno}{'lineno'} = $lineno;
    $data->{$lineno}{'statement'} = $line;
    $data->{$lineno}{'variablesline'} = $variables;
    foreach $individual (@variable)
    {

      @assignment = split(/\s*=\s*/, $individual );
      if(scalar(@assignment) > 1)
      {
        $name = trim($assignment[0]);
        $value = trim($assignment[1]);
      }
      else
      {
            $name = trim($assignment[0]);
        $value = 'UNDEFINED';
      }

      $data->{$lineno}{'variables'}[$cnt]{'name'} = $name;
      $data->{$lineno}{'variables'}[$cnt]{'isArray'} = 1 if($assignment[0] =~ /\]$/);
      $data->{$lineno}{'variables'}[$cnt++]{'value'} = $value;
```

```
    }

}



sub parallelStatement
{
    my $line = $_[1];
    my $data = $_[0];
    my $lineno = $_[2];
    my $cnt = 0;
    my $structStack = $_[3];
    my $nextElement = '';
    my $top = $_[4];
    $top++;
    $line =~ /^#pragma\s+omp\s+parallel\s+(for)?(.*)/;
    if(defined($1))
    {
        $data->{$lineno}{'type'} = 'PARALLEL_FOR';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'PARALLEL_FOR';
        $structStack->{$top}->{'start'} = $lineno;
        $nextElement = 'for(';
    }
    else
    {
            $data->{$lineno}{'type'} = 'PARALLEL';
            $data->{$lineno}{'lineno'} = $lineno;
            $data->{$lineno}{'statement'} = $line;
        $structStack->{$top}->{'name'} = 'PARALLEL';
        $structStack->{$top}->{'start'} = $lineno;
        $nextElement = '{';
    }

   $lineElements = $2;
     if($lineElements =~ /shared\((.*?)\)/)
    {
        @sharedVariables = split(/\s*,\s*/, $1);
        $cnt = defined(@{$data->{$lineno}{'shared'}}) ? scalar(@{$data->{$lineno}{'shared'}}) : 0;
        foreach $sharedVariable (@sharedVariables)
        {
           $data->{$lineno}{'shared'}[$cnt++]{'name'} = $sharedVariable;
        }

    }
    if($lineElements =~ /private\((.*?)\)/)
    {
        @privateVariables = split(/\s*,\s*/, $1);
            $cnt = defined(@{$data->{$lineno}{'private'}}) ? scalar(@{$data->{$lineno}{'private'}}) : 0;
        foreach $privateVariable (@privateVariables)
        {
           $data->{$lineno}{'private'}[$cnt++]{'name'} = $privateVariable;
        }
```

```
}
if($lineElements =~ /default\((.*?)\)/)
{
    $data->{$lineno}{'default'} = $1;
}
else
{
    $data->{$lineno}{'default'} = 'shared';
}
if($lineElements =~ /firstprivate\((.*?)\)/)
{
    @privateVariables = split(/\s*,\s*/, $1);
        $cnt = defined(@{$data->{$lineno}{'firstprivate'}}) ? scalar(@{$data->{$lineno}{'firstprivate'}}) : 0;
    foreach $privateVariable (@privateVariables)
    {
       $data->{$lineno}{'firstprivate'}[$cnt++]{'name'} = $privateVariable;
    }

}
if($lineElements =~ /lastprivate\((.*?)\)/)
{
    @privateVariables = split(/\s*,\s*/, $1);
    $cnt = defined(@{$data->{$lineno}{'lastprivate'}}) ? scalar(@{$data->{$lineno}{'lastprivate'}}) : 0;
    foreach $privateVariable (@privateVariables)
    {
       $data->{$lineno}{'lastprivate'}[$cnt++]{'name'} = $privateVariable;
    }

}
if($lineElements =~ /nowait/)
{
    $data->{$lineno}{'nowait'} = 1;
}
else
{
    $data->{$lineno}{'nowait'} = 0;
}
if($lineElements =~ /reduction\((.*?)\)/)
{
    @reductionElements = split(/\s*:\s*/, $1);
        $cnt = defined(@{$data->{$lineno}{'reduction'}}) ? scalar(@{$data->{$lineno}{'reduction'}}) : 0;
    if(scalar(@reductionElements))
    {
        $cnt1 = 0;
        $operator = $reductionElements[0];
        $data->{$lineno}{'reduction'}[$cnt]{'operator'} = $operator;
        @reductionVariables = split(/\s*,\s*/, trim($reductionElements[1]));
                foreach $reductionVariable (@reductionVariables)
      {
                $data->{$lineno}{'reduction'}[$cnt]{'variables'}[$cnt1++]{'name'} = trim($reductionVariable);
      }

    }
}
```

```
    return ($top,$nextElement);
}

# Perl trim function to remove whitespace from the start and end of the string
sub trim
{
        my $string = shift;
        $string =~ s/^\s+//;
        $string =~ s/\s+$//;
        return $string;
}
# Left trim function to remove leading whitespace
sub ltrim
{
        my $string = shift;
        $string =~ s/^\s+//;
        return $string;
}
# Right trim function to remove trailing whitespace
sub rtrim
{
        my $string = shift;
        $string =~ s/\s+$//;
        return $string;
}


sub recalculate
{
    my $structQueue = $_[1];
        my $parallelStart = -1;
    my $parallelEnd = -1;
    my $data = $_[0];
    my %expression;
    my $key;
    my $cnt = 0;
        foreach  $key ( sort { $a <=> $b } keys(%$structQueue))
    {

        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
        {
                $parallelStart = $structQueue->{$key}{'start'};
          $parallelEnd = $structQueue->{$key}{'end'};
          last;
        }
    }
    foreach  $key ( sort { $a <=> $b } keys(%$data))
    {

         if(int($key) < int($parallelStart))
        {
                if($data->{$key}{'type'} eq 'DECLARATION')
         {
           foreach $index (@{$data->{$key}{'variables'}})
```

```
{
                if(length($index->{'value'}) > 0 && $index->{'value'} !~ /^\d+$/)
    {
                  $expression{'top'} = 0;
                  $expression{'rank'} = 0;
                  $expression{'stack'} = ['#'];
                  $expression{'postfix'} = [];
                  @infix = split(/\s+/,$index->{'value'}.' #');
                  push(@{$expression{'infix'}},@infix);
      findPostfix(\%expression);
      $cnt  = 0;
      $cnt1  = 0;
      $loc = -1;
      foreach $val (@{$expression{'postfix'}})
      {
                        if($val =~ /\w+/)
        {
          foreach $variableName (@{$data->{$key}{'variables'}})
          {
                              $cnt1++;
            if($variableName->{'name'} eq $val)
                              {
              if($variableName->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/ )
                                  {
                                      $expression{'postfix'}->[$cnt] = $variableName->{'value'};
               }
              elsif(length($variableName->{'value'}) == 0)
               {
                                      $loc = $cnt1;
                                  }
              else
                                  {
                                      $index->{'value'} = $variableName->{'value'};
                                  }
            }

          }
          if($loc > 0)
          {
                                $expression{'postfix'}->[$cnt] = getValue($data,$val,$key);
                                $data->{$key}{'variables'}->[$loc]{'value'} =  $expression{'postfix'}-
>[$cnt];
          }
                              $loc = -1;

                        }
            $cnt++;
        }
      if($index->{'value'} ne 'UNDEFINED' && $index->{'value'} ne 'NONDETERMINISTIC')
      {
                        $index->{'value'}  = evaluate(\%expression);

            }

              }
```

```
#print $index. "\n";
                        }
                }
                elsif($data->{$key}{'type'} eq 'ASSIGNMENT')
                {
                        foreach $index (@{$data->{$key}{'write'}})
                  {
                                if(length($index->{'value'}) > 0 && $index->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
                      {
                                  $expression{'top'} = 0;
                                  $expression{'rank'} = 0;
                                  $expression{'stack'} = ['#'];
                                  $expression{'postfix'} = [];
                                  @infix = split(/\s+/,$index->{'value'}.' #');
                                  push(@{$expression{'infix'}},@infix);
                          findPostfix(\%expression);
                          $cnt  = 0;
                          $cnt1  = 0;
                          $loc = -1;
                          foreach $val (@{$expression{'postfix'}})
                          {#print Dumper($data->{$key}{'write'});
                                        if($val =~ /^[a-zA-Z]+$/)
                              {
                                foreach $variableName (@{$data->{$key}{'read'}})
                                {

                                    if($variableName->{'name'} eq $val)
                                                {
                                      if($variableName->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/ )
                                                        {
                                                                $expression{'postfix'}->[$cnt] = $variableName->{'value'};
                                       }
                                       elsif($variableName->{'name'} eq $val && length($variableName->{'value'}) == 0)
                                       {
                                                                $loc = $cnt1;
                                                        }
                                       else
                                                        {

                                                                $index->{'value'} = $variableName->{'value'};
                                                        }
                                     }
                                     $cnt1++;
                                   }
                                 if($loc > 0)
                                 {
#print ":$val:\n";
                                                $expression{'postfix'}->[$cnt] = getValue($data,$val,$key);
                                                $data->{$key}{'read'}->[$loc]{'value'} =  $expression{'postfix'}->[$cnt];
                                 }
                                                $loc = -1;

                                                }
                                        $cnt++;
                                }
```

```
                if(length($index->{'value'}) == 0 || ($index->{'value'} ne 'UNDEFINED' && $index->{'value'} ne
'NONDETERMINISTIC'))
                {

                                $exprVal  = evaluate(\%expression);
                        $index->{'value'} = getValue($data,$index->{'name'},$key-1);
 #print $index->{'name'};
                        if($data->{$key}{'operator'} eq '+=')
                        {
                              $data->{$key}{'read'}->[0]{'value'} = $index->{'value'};
                                        $index->{'value'} += int($exprVal);

                        }
                                elsif($data->{$key}{'operator'} eq '-=')
                        {
                                        $data->{$key}{'read'}->[0]{'value'} = $index->{'value'};
                                        $index->{'value'}  -= int($exprVal);

                        }
                                elsif($data->{$key}{'operator'} eq '*=')
                        {
                                        $data->{$key}{'read'}->[0]{'value'} = $index->{'value'};
                                        $index->{'value'}  *= int($exprVal);

                        }
                                elsif($data->{$key}{'operator'} eq '/=')
                        {
                                        $data->{$key}{'read'}->[0]{'value'} = $index->{'value'};
                                        $index->{'value'}  /= int($exprVal);

                        }
                        elsif($data->{$key}{'operator'} eq '=')
                        {
                                        $index->{'value'} = int($exprVal);
                                }
                                elsif($data->{$key}{'operator'} eq '++')
                        {
                                        $data->{$key}{'read'}->[0]{'value'} = $index->{'value'};
                                        $index->{'value'} += 1;

                        }
                                elsif($data->{$key}{'operator'} eq '--')
                        {
                                        $index->{'value'}  -= 1;

                        }

                        }

                          }
 #print $index. "\n";
                        }

                      }
                      elsif($data->{$key}{'type'} eq 'OUTPUT')
```

```
            {
                        foreach $variableName (@{$data->{$key}{'read'}})
                        {
                                                if(length($variableName->{'value'}) == 0)
                                    {
                                                                $variableName->{'value'} = getValue($data,$variableName-
>{'name'},$key);
                                                        }


                        }

                    }
            }
            }

}

sub findShared()
{
            $structQueue = $_[1];
            $parallelStart = -1;
        $parallelEnd = -1;
        $data = $_[0];
            $sharedVariables = $_[2];
        my $key;
        my $cnt = 0;
            foreach  $key ( sort { $a <=> $b } keys(%$structQueue))
        {
            if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
            {
                        $parallelStart = $structQueue->{$key}{'start'};
             $parallelEnd = $structQueue->{$key}{'end'};
                        if(defined($data->{$parallelStart}{'shared'}))
                        {
                        foreach $sharedVariable (@{$data->{$parallelStart}{'shared'}})
                                    {
                                                push(@$sharedVariables,$sharedVariable->{'name'});
                                    }
                        }
             last;
            }
        }
}

sub findPrivate
{
            $structQueue = $_[1];
            $parallelStart = -1;
        $parallelEnd = -1;
        $data = $_[0];
            $privateVariables = $_[2];
        my $key;
        my $cnt = 0;
```

```
        my $status = 0;
        foreach  $key ( sort { $b <=> $a } keys(%$structQueue))
    {
        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
        {
                $status = 1;
                $parallelStart = $structQueue->{$key}{'start'};
         $parallelEnd = $structQueue->{$key}{'end'};
        }
            if($status == 1)
        {

                if($parallelEnd < $structQueue->{$key}{'start'})
                {
                        $status = 0;
                        last;
                }
                if(defined($data->{$structQueue->{$key}{'start'}}{'private'}))
                {
                        my $check = 0;
                foreach $priVariable (@{$data->{$structQueue->{$key}{'start'}}{'private'}})
                        {
                                foreach $pvar (@$privateVariables)
                                {
                                        if($pvar eq $priVariable->{'name'})
                                        {
                                          $check = 1;
                                        }
                                }
                                if($check != 1)
                                {
                                        push(@$privateVariables,$priVariable->{'name'});
                                        $check = 0;
                                }
                        }
                }
            }
    }
}

sub findFirstPrivate
{
        $structQueue = $_[1];
        $parallelStart = -1;
    $parallelEnd = -1;
    $data = $_[0];
        $firstprivateVariables = $_[2];
    my $key;
    my $cnt = 0;
        my $status = 0;
        foreach  $key ( sort { $b <=> $a } keys(%$structQueue))
    {
        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
```

```
        {
                $status = 1;
                $parallelStart = $structQueue->{$key}{'start'};
         $parallelEnd = $structQueue->{$key}{'end'};
        }
            if($status == 1)
        {

                if($parallelEnd < $structQueue->{$key}{'start'})
                {
                        $status = 0;
                        last;
                }
                if(defined($data->{$structQueue->{$key}{'start'}}{'firstprivate'}))
                {
                        my $check = 0;
                foreach $firstpriVariable (@{$data->{$structQueue->{$key}{'start'}}{'firstprivate'}})
                        {
                                foreach $pvar (@$firstprivateVariables)
                                {
                                        if($pvar eq $firstpriVariable->{'name'})
                                        {
                                          $check = 1;
                                        }
                                }
                                if($check != 1)
                                {
                                        push(@$firstprivateVariables,$firstpriVariable->{'name'});
                                        $check = 0;
                                }
                        }
                }
            }
      }}
sub findLastPrivate
{
        $structQueue = $_[1];
        $parallelStart = -1;
      $parallelEnd = -1;
      $data = $_[0];
        $lastprivateVariables = $_[2];
      my $key;
      my $cnt = 0;
        my $status = 0;
        foreach  $key ( sort { $b <=> $a } keys(%$structQueue))
      {
        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
        {
                $status = 1;
                $parallelStart = $structQueue->{$key}{'start'};
         $parallelEnd = $structQueue->{$key}{'end'};
        }
            if($status == 1)
        {
```

```
                if($parallelEnd < $structQueue->{$key}{'start'})
                {
                        $status = 0;
                        last;
                }
                if(defined($data->{$structQueue->{$key}{'start'}}{'lastprivate'}))
                {
                        my $check = 0;
                foreach $lastpriVariable (@{$data->{$structQueue->{$key}{'start'}}{'lastprivate'}})
                        {
                                foreach $pvar (@$lastprivateVariables)
                                {
                                        if($pvar eq $lastpriVariable->{'name'})
                                        {
                                          $check = 1;
                                        }
                                }
                                if($check != 1)
                                {
                                        push(@$lastprivateVariables,$lastpriVariable->{'name'});
                                        $check = 0;
                                }
                        }
                }
            }
    }
}



sub findReadOnlyShared
{
    my $structQueue = $_[1];
        my $parallelStart = -1;
    my $parallelEnd = -1;
    my $data = $_[0];
        my $readOnlys = $_[2];
    my $key;
    my $cnt = 0;
        foreach  $key ( sort { $a <=> $b } keys(%$structQueue))
    {
        if($structQueue->{$key}{'name'} eq 'PARALLEL' || $structQueue->{$key}{'name'} eq
'PARALLEL_FOR')
        {
                $parallelStart = $structQueue->{$key}{'start'};
          $parallelEnd = $structQueue->{$key}{'end'};
          last;
        }
    }

        for(my $start = $parallelStart ; $start <= $parallelEnd ; $start++)
    {
                if(defined($data->{$start}{'read'}))
          {
```

```
                        foreach my $readVar ( @{$data->{$start}{'read'}} )
                        {
                                $status = 0;
                                foreach my $readOnlyVar (@$readOnlys)
                  {
                                        if($readOnlyVar eq $readVar->{'name'})
                              {
                                                $status = 1;
                                        }
                                }

                                if($status != 1)
                                {
                                        if($readVar->{'name'} !~ /^(\d+\.?\d*|\.\d+)$/)
                                        {

                                                        push(@$readOnlys,$readVar->{'name'});
                                        }
                                }

                        }

                 }
                if(defined($data->{$start}{'write'}))
          {
                #       print "$start \n";
                        foreach my $writeVar ( @{$data->{$start}{'write'}} )
                        {
                                push(@writeVariable,$writeVar->{'name'});
                        }

                 }
        }

        foreach my $writeVar (@writeVariable)
        {
                my $cnt = 0;
                foreach $readVar (@$readOnlys)
                {
                        if($readVar eq $writeVar)
                        {
                                splice(@$readOnlys,$cnt,1);
                        }
                        $cnt++;
                }
        }}
sub copyVariable
{
        my $structQueue = $_[1];
        my $parallelStart = -1;
      my $parallelEnd = -1;
      my $data = $_[0];
        my $return = 0;
        my %privateVar;
```

```
        my %sharedVar;
        my %firstPrivateVar;
        my %lastPrivateVar;
        my $default = 'none';
    my $var1;
    my $queueLength = scalar(keys(%$structQueue));
        foreach my $key ( sort { $b <=> $a } keys(%$structQueue))
    {
#print "\n\n\n" .Dumper($structQueue->{$key}) . "\n";
                my $done = 0;

        for(my $ele = $key + 1;$done != 1 && $ele < $queueLength; $ele++)
          {
                  if($structQueue->{$ele}->{'start'} < $structQueue->{$key}->{'start'} && $structQueue-
>{$ele}->{'end'} > $structQueue->{$key}->{'end'})
           {
                        $done = 1;
                        $parallelStart = $structQueue->{$ele}->{'start'};
                        $parallelEnd = $structQueue->{$ele}->{'end'};
                        if(defined($data->{$parallelStart}{'private'}))
                {
                  my $cnt = 0;
                        foreach my $var1 (@{$data->{$parallelStart}{'private'}})
                                {
                                        $structQueue->{$key}->{'private'}->[$cnt]->{'name'} = $var1-
>{'name'};
                                        my $st = 0;
                                        my $cnt1 = 0;

                                        foreach my $var2 (@{$data->{$structQueue->{$key}-
>{'start'}}{'private'}})
                                        {
                                                if($var1->{'name'} eq $var2->{'name'})
                                                {
                                                  $st = 1;
                                                }
                                                $cnt1++;
                                        }
                        if($st != 1)
                        {
                                                $data->{$structQueue->{$key}->{'start'}}{'private'}-
>[$cnt1]{'name'} = $var1->{'name'};
                                        }
                        $cnt++;
                                }
                }
                  if(defined($data->{$parallelStart}{'firstprivate'}))
                  {
                  my $cnt = 0;
                  foreach my $var1 (@{$data->{$parallelStart}{'firstprivate'}})
                                {
                                        $structQueue->{$key}->{'firstprivate'}->[$cnt]->{'name'} = $var1-
>{'name'};
                                        my $st = 0;
```

```
                                        my $cnt1 = 0;

                                        foreach my $var2 (@{$data->{$structQueue->{$key}-
>{'start'}}{'firstprivate'}})
                                        {
                                                if($var1->{'name'} eq $var2->{'name'})
                                                {
                                                  $st = 1;
                                                }
                                                $cnt1++;
                                        }
                                if($st != 1)
                                {
                                                $data->{$structQueue->{$key}->{'start'}}{'firstprivate'}-
>[$cnt1]{'name'} = $var1->{'name'};
                                        }
                                $cnt++;
                                        }
                                  }
                                  if(defined($data->{$parallelStart}{'lastprivate'}))
                                  {
                                  my $cnt = 0;
                                  foreach my $var1 (@{$data->{$parallelStart}{'lastprivate'}})
                                        {
                                                $structQueue->{$key}->{'lastprivate'}->[$cnt]->{'name'} = $var1-
>{'name'};
                                        my $st = 0;
                                        my $cnt1 = 0;

                                        foreach my $var2 (@{$data->{$structQueue->{$key}-
>{'start'}}{'lastprivate'}})
                                        {
                                                if($var1->{'name'} eq $var2->{'name'})
                                                {
                                                  $st = 1;
                                                }
                                                $cnt1++;
                                        }
                                if($st != 1)
                                {
                                                $data->{$structQueue->{$key}->{'start'}}{'lastprivate'}-
>[$cnt1]{'name'} = $var1->{'name'};
                                        }
                                $cnt++;
                                        }
                                  }
                                  if(defined($data->{$parallelStart}{'shared'}))
                                  {
                                  my $cnt = 0;
                                  foreach my $var1 (@{$data->{$parallelStart}{'shared'}})
                                        {
                                                $structQueue->{$key}->{'shared'}->[$cnt]->{'name'} = $var1-
>{'name'};
                                        my $st = 0;
```

```
                                        my $cnt1 = 0;

                                        foreach my $var2 (@{$data->{$structQueue->{$key}-
>{'start'}}{'shared'}})
                                        {
                                                if($var1->{'name'} eq $var2->{'name'})
                                                {
                                                  $st = 1;
                                                }
                                                $cnt1++;
                                        }
                                if($st != 1)
                                {
                                                $data->{$structQueue->{$key}->{'start'}}{'shared'}-
>[$cnt1]{'name'} = $var1->{'name'};
                                        }
                                $cnt++;
                                        }
                                  }
                        }



          }
      }
}


sub getValue
{
my $data = $_[0];
my $var = $_[1];
my $index1 = $_[2];
 for(my $i = $index1;$i>0;$i--)
 {#print "\n\n\n" . $i . "\n\n\n";
    if($data->{$i}{'type'} eq 'DECLARATION')
    {
          foreach $variable (@{$data->{$i}{'variables'}})
      {
                if($variable->{'name'} eq $var && $variable->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/)
           {
                  return $variable->{'value'};
           }
                elsif($variable->{'name'} eq $var && length($variable->{'value'}) == 0)
           {

           }
           elsif($variable->{'name'} eq $var && $variable->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
           {
                  return $variable->{'value'};
           }
      }
    }
    elsif($data->{$i}{'type'} eq 'OUTPUT')
    {
```

```
        foreach $variable (@{$data->{$i}{'read'}})
  {#print $variable;
                if($variable->{'name'} eq $var && $variable->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  return $variable->{'value'};
        }
                elsif($variable->{'name'} eq $var && length($variable->{'value'}) == 0)
        {
        }
        elsif($variable->{'name'} eq $var && $variable->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  return $variable->{'value'};
        }
  }
}
elsif($data->{$i}{'type'} eq 'INPUT')
{
        foreach $variable (@{$data->{$i}{'write'}})
  {
                if($variable->{'name'} eq $var && $variable->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  return $variable->{'value'};
        }
                elsif($variable->{'name'} eq $var && length($variable->{'value'}) == 0)
        {
        }
        elsif($variable->{'name'} eq $var && $variable->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  return $variable->{'value'};
        }
  }
}
elsif($data->{$i}{'type'} eq 'ASSIGNMENT')
{
        my $return = 0;
        my $validValue = 0;
        foreach $variable (@{$data->{$i}{'read'}})
  {

                if($variable->{'name'} eq $var && $variable->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  $return = 1;
                  $validValue = $variable->{'value'};
        }
        elsif($variable->{'name'} eq $var && length($variable->{'value'}) == 0)
        {
        }
        elsif($variable->{'name'} eq $var && $variable->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
        {
                  $return = 1;
                  $validValue = $variable->{'value'};
        }
  }
        foreach $variable (@{$data->{$i}{'write'}})
  {
```

```
                    if($variable->{'name'} eq $var && $variable->{'value'} =~ /^(\d+\.?\d*|\.\d+)$/)
            {
                    return $variable->{'value'};
            }
                    elsif($variable->{'name'} eq $var && length($variable->{'value'}) == 0)
            {
            }
            elsif($variable->{'name'} eq $var && $variable->{'value'} !~ /^(\d+\.?\d*|\.\d+)$/)
            {
                    return $variable->{'value'};
            }

      }
          if($return == 1)
      {
        return $validValue;
          }
    }
   }       }
sub findPostfix
  {
     $expression = $_[0];
     $stack = $expression->{'stack'};
     $infix = $expression->{'infix'};
     $postfix = $expression->{'postfix'};
     $top1 = 0;
     $rank = 0;
     $j=0;

    for($i=0;$infix[$i] ne '#';$i++)
     {
           if($infix[$i] =~ /^(\d+\.?\d*|\.\d+|\w+)$/)
        {
             $rank+=1;
         $postfix->[$j++]=$infix[$i];
        }
       elsif($infix[$i] eq '(')
        {
         $top1++;
         $stack->[$top1] = '(';
        }
       elsif($infix[$i] eq ')')
       {
            while($stack->[$top1] ne '(')
        {
              $postfix->[$j++]=$stack->[$top1--];
          $rank--;
          if($rank<1)
           {
            exit(0);
           }
         }
         $top1--;
       }
      else
```

```
{
        switch($infix[$i])
  {
    case ['-','+']
     {
                while(1)
      {
                        if($stack->[$top1] eq '#' || $stack->[$top1] eq '(')
            {
                        last;
            }
                        $postfix->[$j++]=$stack->[$top1--];
                        $rank--;

                if($rank<1)
                {
$top $rank $j ");
                        exit(0);
                }
      }
       $top1++;
      $stack->[$top1]=$infix[$i];
    }
    case ['*','/']
    {
      if($stack->[$top1]=='^' || $stack->[$top1]=='*' || $stack->[$top1] == '/')
                {
                while($stack->[$top1] ne '+' && $stack->[$top1] eq '-' )
                        {
                                if($stack->[$top1] eq '#'|| $stack->[$top1] eq '(')
                        {
                                        last;
                                }

                        $postfix->[$j++]=$stack->[$top1--];
                        $rank--;
                        if($rank<1)
                        {
                                #       print("\n\n Invalid Infix Expression : ");
                                exit(0);
                }
                        }
      }
      $stack->[++$top1]=$infix[$i];
    }
    case '^'
    {
      $stack->[++$top1]=$infix[$i];
    }
  }
 }
}
while($stack->[$top1] ne '#')
{
 $postfix->[$j++]=$stack->[$top1--];
```

```
        $rank--;
          if($rank < 1)
          {
           exit(0);
          }
        }
        $expression->{'rank'} = $rank;
        $expression->{'top'} = $top1;
 }

sub evaluate
  {
    my $expression = $_[0];
    my $stack = $expression->{'stack'};
    my $infix = $expression->{'infix'};
    my $postfix = $expression->{'postfix'};
    my $top = \$expression->{'top'};
    my $rank = \$expression->{'rank'};
    my $j=1;
    my $temp = 0;
    my @array = ();

   for($i=0;$i < scalar(@$postfix);$i++)
   {
    if($postfix->[$i] =~ /^(\d+\.?\d*|\.\d+)$/)
     {
       $array[$j++]=$postfix->[$i];
     }
    else
     {
            $j-=2;
       switch($postfix->[$i])
        {
          case '+'
               {
          $temp=$array[$j]+$array[$j+1];
          $array[$j++]=$temp;
          }
          case '-'
               {
          $temp=$array[$j]-$array[$j+1];
          $array[$j++]=$temp;
               }
          case '*'
               {
          $temp=$array[$j]*$array[$j+1];
          $array[$j++]=$temp;
          }
          case '/'
               {
          $temp=$array[$j]/$array[$j+1];
          $array[$j++]=$temp;
          }
          case '^'
               {
```

```
            $temp= $array[$j]**$array[$j+1];
            $array[$j++]=$temp;
                }
          }
        }
    }
      return $array[--$j];
}
```

REFERENCES

[1] Alexander Aiken and David Gay. Barrier inference. In POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 342–354, New York, NY, USA, 1998. ACM Press.

[2] Banerjee, U., B. Bliss, Z. Ma, and P. Petersen, "A Theory of Data Race Detection." *Proc. of Workshop on Parallel and Distributed Systems: Testing and Debugging* (PADTAD), pp. 69-78, ACM, Portland, USA, July 2006.

[3] Banerjee, U., B. Bliss, Z. Ma, and P. Petersen, "Unraveling Data Race Detection in the Intel Thread Checker." *Proc. of Workshop on Software Tools for Multi-core Systems* (STMCS06), Manhattan, New York, NY., USA, March 2006.

[4] Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, Massachuesetts: The MIT Press, 2008.

[5] Dagum, L., Menon, R., "OpenMP: An Industry-Standard API for Shared Memory Programming," *Computational Science and Engineering*, 5(1): 46-55, IEEE, January-March 1998.

[6] David Callahan and Jaspal Sublok. Static analysis of low-level synchronization. In Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging, pages 100–111. ACM Press, 1988.

[7] David Callahan, Ken Kennedy, and Jaspal Subhlok. Analysis of event synchronization in a parallel programming tool. In Proceedings of the Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, pages 21–30, Seattle, Washington, March 1990.

[8] Dinning, A., and E. Schonberg, "An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection," *2nd Symp. on Principles and Practice of Parallel Programming*, ACM, pp. 1-10, March 1990.

[9] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[10] Nagappan, N., Ball, T. "Static Analysis Tools as Early Indicator of Pre-release Defect Density"15-21 May 2005 Page(s):580 – 586

[11] Ogasawara, H., Aizawa, M., Yamada, A. "Experiences with Program Static Analysis". Page(s):109 – 112, IEEE, Nov. 1998

[12] OpenMP.
https://computing.llnl.gov/tutorials/openMP/

[13] OpenMP C/C++ Manual.
http://www.openmp.org

[14] Parallel Lint Overview.
http://software.intel.com/sites/products/documentation/studio/composer/en-us/2009/compiler_c/bldaps_cls/common/bldaps_svover.htm

[15] Savage, S., Burrows, M., Nelson, G., Sobalvarro, P., and Anderson , T. "Eraser: A Dynamic Data Race Detector for Multi-threaded Programs." *ACM Transactions on Computer Systems*, 15(4):Page(s):391–411, 1997.

[16] Sreedhar, V. C., Zhang, Y., Gao, G. R. A New Framework for Analysis and Optimization of Shared Memory Parallel Programs.(Jul 2005). Unviersity of Delaware.

[17] *Static code analysis - Wikipedia, the free encyclopedia*. 30 Aug. 2009. 30 Aug. 2009 http://en.wikipedia.org/wiki/static_code_analysis.

[18] Stephen P. Masticola and Barbara G. Ryder. Non-concurrency analysis. In Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, pages 129–138, San Diego, California, May 1993.

[19] Tor E. Jeremiassen and Susan J. Eggers. Static analysis of barrier synchronization in explicitly parallel systems. In Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques, PACT '94, pages 171–180, Montr´eal, Qu´ebec, August 1994. North-Holland Publishing Company.

[20] *VivaMP - a tool for OpenMP*. Aug 2009. http://www.viva64.com/content/articles/pvs-studio-articles/ ?f=vivamp.html&lang=en&content=pvs-studio-articles.

[21] Yu, Y., Rodeheffer, T., and Chen, W. 2005. RaceTrack: efficient detection of data race conditions via adaptive tracking. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles* (Brighton, United Kingdom, October 23 - 26, 2005). SOSP '05. ACM, New York, NY, 221-234. DOI
http://doi.acm.org/10.1145/1095810.1095832

[22] Yuan Lin. Static nonconcurrency analysis of openmp programs. In FIRST INTERNATIONAL WORKSHOP on OpenMP, 2005.