Volume 3, Issue 12, December 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Efficient Dynamic Slicing Algorithm for Prioritizing Synchronized Multithreaded Programs

Divanshi Priyadarshni Wangoo
Research Scholar, CSE & IT, ITM University
Gurgaon, India

Dr. Supriya P.Panda
Associate Professor, CSE &IT, ITM University
Gurgaon, India

Abstract- This paper proposes an efficient slicing algorithm for prioritizing synchronized multithreaded programs. The algorithm would compute precise slices in multiple thread execution environments. The equal priority threads having the maximum priority would execute in a synchronized way without preempting the other threads. Synchronization method is used in programs when a thread is reading a record from a file and at the same time another thread wants to update the same file. In Java programming language, a monitor is created for the thread that calls the method first and remains till the thread holds the section. During this time no other thread can enter the code involving the synchronized section. Dynamic slicing algorithm would synchronize the allocation and deallocation of the resources to the threads with assigned priorities in multithreaded environment. This would facilitate efficient usage of resources by all the threads and would make the resources available for further allocation. It would also assist in the prevention of deadlock mechanisms where more than two threads try to gain control on the same resource the same time. Moreover, the assigned CPU time to threads will exhibit in equal accordance with the maximum priorities assigned. Thus, execution of multithreaded programs would accelerate at a valuable rate.

Keywords- Dynamic Thread Dependence Graph (DTDG), Priority Multithread Dependence Graph (PMDG), System Dependence Graphs (SDG), Lines of Code (LOC), multithreading, dynamic slicing

I. INTRODUCTION

Multitasking is a phenomena involving simultaneous execution of several programs at the same time. In programming terminology this phenomena is called multithreading. The parallel execution of several threads on a single processor as in most of the day to day computers is so fast that a human would be unable to interpret the switching mechanism behind it. The switching of control between the threads is done by the interpreter. Java language has a powerful multithreading mechanism that helps in fast execution of programs with multiple threads. The programs running multiple threads simultaneously come under the category of concurrent programs. Normally a multithreaded program consists of one main thread and several other threads. The purpose of the main thread is to initiate other threads in the same program. For example, a multithreaded program consists of one main() thread and other threads namely X, Y, Z. The main() thread module marks the starting execution point and also initializes the created threads in the program. This concept is illustrated in Fig.1 below:

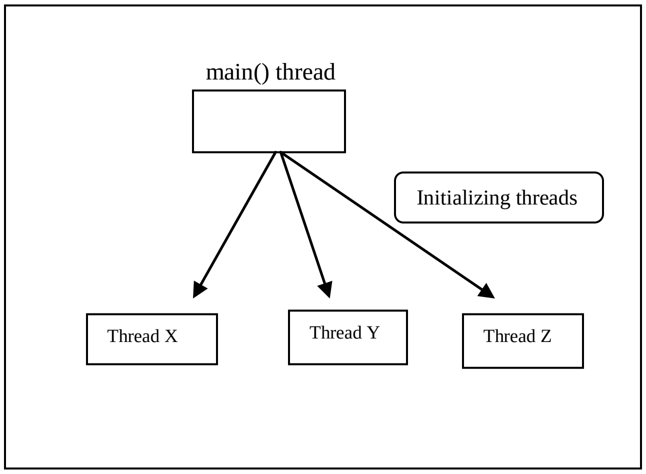


Fig. 1 Execution flow of a multithreaded program

There are a lot of programming tasks that require multithreading mechanisms in their programs. For example, when a user wants to print a document and in the meanwhile reads a document from another file, then this task can be accomplished by executing multiple threads in a parallel environment. Thus, one thread would target at printing of the document and another thread would correspond to the process of reading a document from a file. Through multithreading a programmer can divide a complex program containing concurrent operations into simpler threads and run them in parallel. This would aid in faster running of the complex program at a tremendously reduced execution rate. Generally multithreaded programs are considered as running single-threaded programs in parallel. A single threaded program consists of a single flow of control. Threads are assigned priorities so that all the running threads are processed by the CPU in the sequential order. For this scheduling threads are given maximum, minimum or normal priority based on the nature of the processing of the threads. For example, background tasks such as screen repainting should be assigned with least priority and a file updating task is given the highest priority. In this way the highest priority thread would be executed first in comparison with the other threads. There are also situations where threads in a program come under the same assignment of the maximum priority. In such situations the CPU processing time is not divided among the same priority threads at the same time leading to blocking situation. Although some java programming environments suspend the blocking threads but it is not a reliable method for real time employment. So, for the smooth running multithreaded programs that have multiple flow of controls, there is a need for some mechanism to overcome this blocking state of threads. Such a mechanism is dynamic slicing which is a slicing technique which lays its base in the construction of System Dependence Graphs (SDG).

This paper is focussed on developing an efficient slicing algorithm for multithreaded programs with equal priority threads. The algorithm would assist in effective scheduling of two or more than two threads that have been equipped with the highest priority. Multithreaded programs that require locking and unlocking of the critical section resources run in a synchronized way. The algorithm would compute precise slices by taking dynamic execution traces in the multithreaded program. The slices would correspond to those statements in the program that are affected by the dynamic slicing criterion. Dynamic slicing is a beneficial technique that would help in the synchronization of multithreaded programs and would catalyse the rate of execution with thread priority at the top side.

II. MULTITHREADING IN OBJECT- ORIENTED PROGRAMS

Object-oriented programs have features such as polymorphism, abstraction, message passing, encapsulation, etc that are present in almost every language coming in the limelight of object-oriented programming languages such as C++, java, C# ., etc. There is a unique feature in java called multithreading that aims at running multiple flow of controls in the underlying program. This enables parallel processing on a single processor. The processor would switch between threads and make every thread processing complete from start to the end of the program. The threads share the flow of execution on the single processor. Threads in java are created in the form of objects encapsulating the method called run(). Usually threads are created in two ways, either by defining a class extending the Thread class in java.lang.Thread package that would override its run() method, or by defining a class that implements the Runnable interface. Either of the two methods can be used depending on the requirement of the class the user wants to create.

A thread has a lifecycle that represents various states that a thread undergoes from its creation to destruction state. Normally a thread is in one of the five states- newborn, runnable, running, blocked and dead state. The newborn state deals with the creation of the threads but the thread is not scheduled for running. The newborn thread can be scheduled using the start() method or killed using the stop() method. In the runnable state the thread is ready for execution and joins the queue of threads waiting for the availability of the processor. In the running state the processor is available to the thread for its execution. A thread is in the blocked state when it is unable to enter the runnable or running state. When the run method of a thread is completely executed the thread is said to be in a dead state. The dead state also signifies the end of the thread lifecycle. The lifecycle mechanism of a thread enables the various methods that can be invoked on the thread in any one state like yield() method in the runnable state and sleep() in the running state.

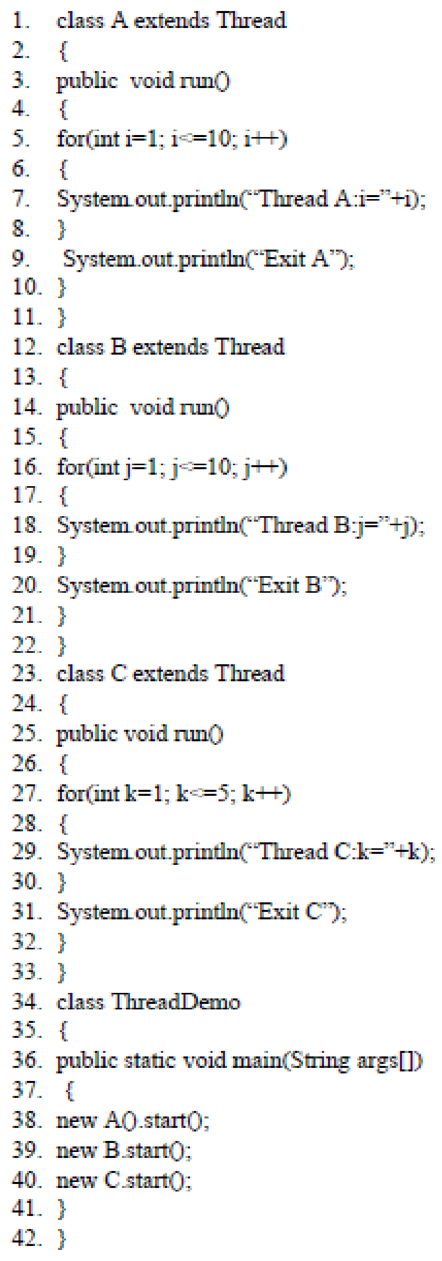
III. DYNAMIC SLICING EMPLOYING THREADS

Dynamic slicing technique works for the programming languages having an object-oriented base. A dynamic slicing criterion specifies the input, and takes into account different occurrences of a statement in an execution trace [1]. Usually the dynamic slicing criteria consists of the execution trace consisting of the input and the program statement.The technique works by computing of slices on a special type of graph called System Dependence Graph based on a particular slicing criterion <s, V>, where s is the program statement and V is the program variable for which the slice is being computed [2]. Multithreaded programming is a unique feature in java language that simultaneously executes several threads initiated by a main thread in the program. This can be accomplished by the construction two new types of dependence graphs called Dynamic Thread Dependence Graph (DTDG) for normal threaded programs and Priority Multithreaded Dependence Graph (PMDG) for prioritize threads in multithreaded programs .

Although researches have been done for the construction of multithreaded dependence graph and thread dependence graphs for synchronization programs in multithreaded environments [3], [4]. There remains further scope for implementation for the priority based synchronization using the dependence graphs. Thus this paper introduces the two graphs i.e., DTDG and PMDG which will be taken as input for the algorithm based on the nature of the multithreading program.

A. Dynamic Thread Dependence Graph (DTDG)

The Dynamic Thread Dependence graphs (DTDG) is a directed System Dependence Graph(DSDG) where the nodes represent the threads and statements executed in a program, directed straight lines represent the control dependencies, directed dotted lines represent the data dependencies and dotted dashed lines represent the parameter bindings and the invocation of the initialized threads by the main() thread. The directed dashed lines depict the creation of thread class objects. The multithreading program and its DTDG are depicted in Fig.2 and Fig.3 respectively.



```
1.  class A extends Thread
2.  {
3.  public  void run()
4.  {
5.  for(int i=1; i<=10; i++)
6.  {
7.  System.out.println("Thread A:i="+i);
8.  }
9.   System.out.println("Exit A");
10. }
11. }
12. class B extends Thread
13. {
14. public  void run()
15. {
16. for(int j=1; j<=10; j++)
17. {
18. System.out.println("Thread B:j="+j);
19. }
20. System.out.println("Exit B");
21. }
22. }
23. class C extends Thread
24. {
25. public void run()
26. {
27. for(int k=1; k<=5; k++)
28. {
29. System.out.println("Thread C:k="+k);
30. }
31. System.out.println("Exit C");
32. }
33. }
34. class ThreadDemo
35. {
36. public static void main(String args[])
37.  {
38. new A().start();
39. new B.start();
40. new C.start();
41. }
42. }
```

Fig.2 A simple multithreaded program in java

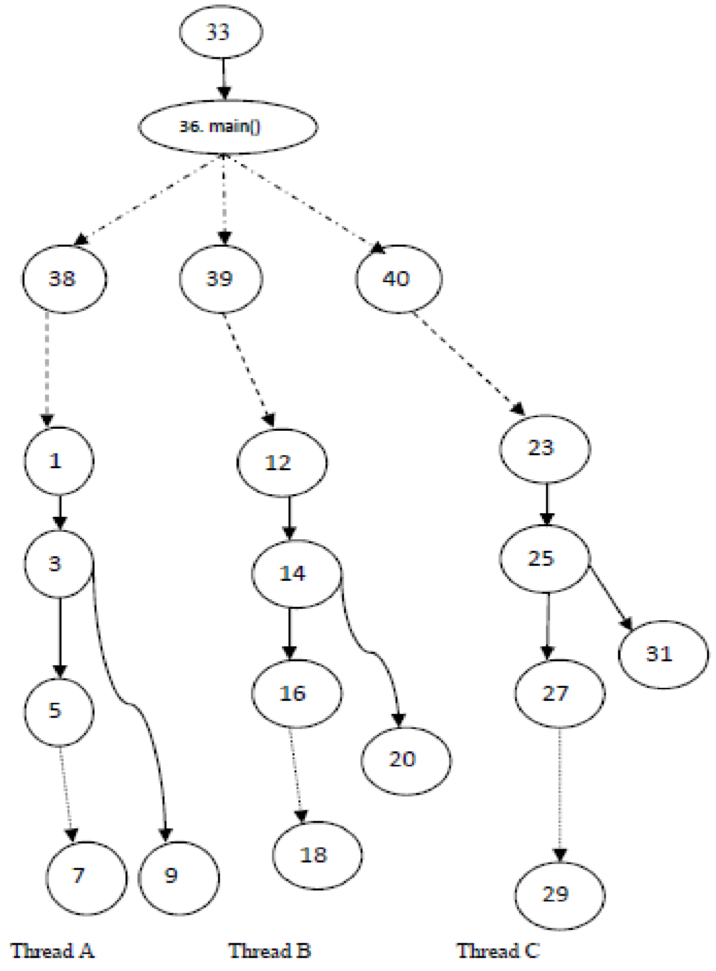


Fig.3 DTDG of Fig.2

B. Priority Multithread Dependence Graph (PMDG)

A Priority Multithreaded Dependence Graph (PMDG) is a directed System Dependence Graph (SDG) with directed arcs having different dependencies. The directed solid lines represent control dependencies, directed dotted lines represent data dependencies, directed dashed lines represent the creation of thread class objects, thread priority and thread execution, and dotted dashed lines represent procedural & parameter bindings. The java program representing use of priorities in threads and its PMDG are depicted in Fig.4 and Fig.5, respectively.

```
1.   class A extends Thread
2.   {
3.   public void run()
4.   {
5.   System.out.println(“threadA started”);
6.   for(int i=1; i<=5; i++)
7.   {
8.   System.out.println(“Thread A:i=”+i);
9.   }
10.  System.out.println(“Exit A”);
11.  }
12.  }
13.  class B extends Thread
14.  {
15.  public  void run()
16.  {
17.  System.out.println(“threadB started”);
18.   for(int j=1; j<=5; j++ )
19.  {
20.  System.out.println(“Thread B:j=”+j);
21.  }
22.  System.out.println(“Exit B”);
23.  }
24.  }
25.  class C extends Thread
26.  {
27.  public void run()
28.  {
29.  System.out.println(“ThreadC started”);
30.  for(k=1;k<=5;k++)
31.  {
32.  System.out.println(“Thread C:k=”+k);
33.  }
34.  System.out.println(“Exit C”);
35.  }
36.  }
37.  class ThreadPriorityDemo
38.  {
39.  public static void main(String args[])
40.  {
41.  A threadA=new A();
42.  B threadB=new B();
43.  C threadC=new C();
44.  threadC.setPriority(Thread.MAX_PRIORITY);
45.  threadB.setPriority(threadA.getPriority()+1);
46.  threadA.setPriority(Thread.MIN_PRIORITY);
47.  System.out.println(“Start Thread A”);
48.  threadA.start();
49.  System.out.println(“Start Thread B”);
50.  threadB.start();
51.  System.out.println(“ Start Thread C”);
52.  threadC.start();
53.  System.out.println(“End of main thread”);
54.  }
55.  }
```

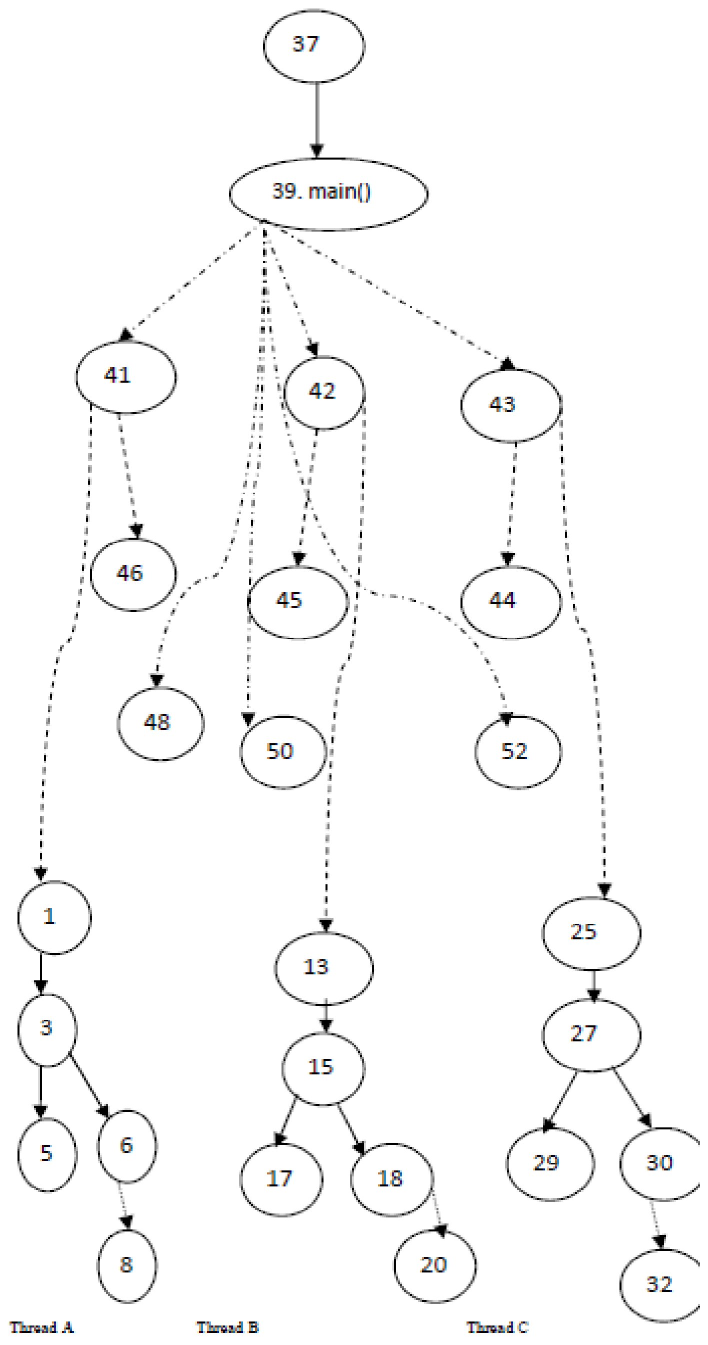


Fig.4 A java program representing use of priorities in threads

Fig.5 PMDG of Fig.4

C. Algorithm for Priority multithreaded programs

The slicing algorithm for the multithreaded programs based on priority takes the execution trace of the program at run time. The algorithm computes all the threads that have the maximum priority for CPU scheduling task. The threads with the minimum priority are executed last after the high priority threads. The algorithm is called Equalize Priority Threads Algorithm(EPTA) and would be computing all the threads in the multithreaded program with the same assigned priority. The threads would be synchronised sequentially for the processing by the CPU scheduler.
The dynamic slicing criterion taken for this algorithm is <S, v, E, i> where S is the statement in the program, v is the variable or object used in the specified statements, E is the execution trace with the input *i* provided dynamically at run time [5]. This algorithm makes sure that the threads having maximum priorities are processed equally by the CPU by concurrent suspending and resuming of threads. The threads can also be sent to the sleep state for a specified period of time in milliseconds.

Equalize Priority Threads Algorithm(EPTA)

Input: Threads in a priority based multithreaded program, dynamic slicing criterion <S, v, E, i>.

Output: Processed Threads with maximum equal priorities.

```
Step 1: Construct the PMDG of the given priority based multithreaded program

Step 2: Take the slicing criterion <S, v, E, i>, where the input i is the maximum priority thread taken dynamically at run
time and repeat the following

(a)  Repeat while threadX.stop(); and thready.stop();
(b)  threadX.start(); and thready.start();
(c)  for X=1 to 10 and Y=1 to 10;
(d)  if(threadX==MAX_PRIORITY)
(e)  then Print thread X processing
(f)  if(threadY==MAX_PRIORITY)
(g)  then Print threadY processing
(h)  else Print threadX  pre-empted; threadX.suspend()
(i)  else Print  threadY pre-empted; threadY.suspend()
Step 3: Resume the suspended threads
(a)  threadX.resume();Print threadX processed
(b)  threadY.resume(); threadY processed
Step 4: End
```

D. Execution slice traces in the PMDG

The execution of the maximum priority threads using dynamic slicing criterion takes into consideration only those statements in the program that have an impact on the thread segment. The implementation of Equalize Priority Threads Algorithm for program in Fig.4 works as under.
Suppose the execution trace taken for the program is <44, threadC> with input 10, then the affected statements would be 43, 25, 27, 29, 30, 32, 34. The number of statements under consideration reduces to minimum number of statements and reflect only those statements that are actually affected by the slicing criterion. Thus dynamic slicing algorithm proves to be less time consuming. The execution rate in case of maximum priority threads also increases by incorporation of the Equalize Priority Threads Algorithm in the running stage of the thread. Table 1 gives the execution trace slices based on the dynamic slicing criterion for all the three threads in the program of Fig.4. The table also gives the lines of code taken for the program and gives the actual number of statements actually affected by the slicing criterion. Last column gives the status of the precision attained by using slicing criterion in the program. The precision is Yes if attained otherwise No is the Output.

TABLE I

Thread priority	**Execution trace slices**	**LOC**	**Statements affected**	**Precision attained**
ThreadC	<44, threadC>, i=10	55	43, 25, 27, 29, 30, 32, 34	Yes
ThreadB	<45, threadB>, i=10	55	42, 13, 15, 17, 18, 20, 22	Yes
ThreadA	<46, threadA>, i=10	55	41, 1, 3, 5, 6, 8, 10	Yes

IV. CONCLUSIONS

Dynamic slicing algorithm aids in the acceleration of threads in synchronized multithreaded programs with equally assigned priorities. Threads with equal maximum priorities would be allocated equal CPU time for implementation. It induces faster rate of execution of the programs. This stimulates the concurrency of the multithreaded programs by availing the critical section resources for other threads. Also, the occurrence of deadlocks is belittled to an estimable track that would not impede the path of execution. Thus, the slicing algorithm proves to be an enhancement for the multithreaded programs using synchronization mechanism for its efficient usage of resources.

REFERENCES

[1] N.Sasirekha,A.Edwin Robert and Dr.M.Hemalatha ,” Program Slicing Techniques and its Applications “, International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3,pp 50- 64, July 2011

[2] Donglin Liang, Mary Jean Harrold,“Slicing Objects using System Dependence Graphs”, International Conference on Software Maintenance,Washington, D.C, pp. 358-367,November 1998.

[3] Zhang Guangquan & Rong Mei, “An Approach of Concurrent Object-oriented Program Slicing Base on LTL Property”, International Conference on Computer Science and Software Engineering, pp 650-653, 2008

[4] Jianjun Zhao, “Multithreaded Dependence Graphs for Concurrent Java Programs”, IEEE, pp 13-23, 1999

[5] Durga Prasad Mohapatra, Rajib Mall and Rajeev Kumar, *“An Overview of Slicing Techniques for Object-Oriented Programs”*, Informatica30 , pp.253-277,2006.