Multithreading Prioritization Concurrently By using an effective Dynamic Slicing Algorithm

Maysoon A. Mohammed[1, 2], Mazlina Abdul Majid[1], Mohammed Adam Ibrahim[1], Balsam A. Mustafa[1]

[1] Faculty of Computer Systems & Software engineering, Universiti Malaysia Pahang, Kuantan, Pahang, Malaysia

[2] Mechanical Engineering Department, University of Technology, Karrada, Baghdad, Iraq

E. mails: saiftrt@yahoo.com, mazlina@ump.edu.my, adamibrahim@ump.edu.my, balsam@ump.edu.my

Abstract: Lately, multithreading evolved into a standard way to enhance the processor usage and program efficiency. The dynamic program slice can be that component of an application that ''affects'' the working out of an adjustable regarding interest throughout program performance over a certain system input. Dynamic program slicing describes an amount of program slicing techniques that depend on program execution and may even significantly decrease the size of an application slice simply because run-time data, accumulated during program execution, is used in order to figure out system slices. Three related methods to multithreading prioritization are introduced in this paper. One of the problems with multithreading is the concurrent of the operations where in this research we expand an effective dynamic slicing algorithm to priorities multithreaded concurrently. The algorithm would compute exact slices in multithreaded execution circumstances. The similar priority threads having the highest priority might perform in a synchronized way with no deadlock with another thread. The results of this work would help the threads with the same priorities to be executed simultaneously and reduce processing time.

Keywords: Dynamic slicing, multithreading, prioritization, and CPTA.

1. INTRODUCTION

Multitasking is a task covering concurrent execution of several programs at the same time. In programming term this phenomena is called multithreading. Nowadays, the computers became very fast for a human to handle the interpretation of the switching mechanism over the parallel execution of several threads on a single processor. So the interpreter would do the switching among the threads. A multi-threaded program starts with one thread of execution, and alternative threads could also be created afterward. Generally, a multithreaded program consists of one main (global) thread and many other threads. The other threads are initiated by the main thread. For example, a multithreaded program consists of one main thread and other threads named X, Y and Z. The main thread considers the start and initial execution point that the threads created by it. Furthermore, the local threads X, Y and Z could be a main thread to other threads and so on, this illustrated in figure 1 below:

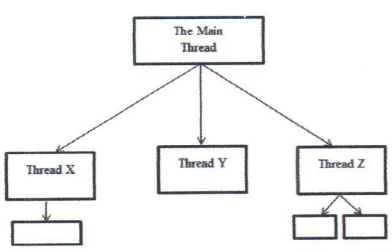


Figure 1: Thread Initialization

Most of the applications that we use through the PC or even mobile are multi-threaded, that means more than one task will be implemented concurrently. For example, when using the Internet, the network browser implemented by more than one thread which may be a thread to display the images and another thread to retrieve data from the network. Another example, the Microsoft word application could have multi threads for displaying graphics, responding to keystrokes from the user, and performing spelling and grammar checking in the background. Also, For instance, when a user wants to print and read a document from files in the same time, this can perform by executing multiple threads in a parallel execution. 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. One of the solutions of the concurrent operations is to divide the complex program to segments or simpler tasks and execute them in parallel. This would assist to speed up the execution of the complex program to decrease the execution rate extremely. CPU would process the execution threads that have priorities in a sequential order. Depending on the nature of the processing of the thread there are three given degrees of priority maximum, normal and minimum priority. For example, background tasks such as screen personalize should be assigned with lowest priority and an error detecting or file updating tasks is given the highest priority. Normally, thread with the highest priority would be run first followed by the thread with the lowest priority. In the case of more than one thread have the maximum priority the CPU time cannot partitioned among threads having the same priority leading to blocking situation. So we expand a dynamic slicing algorithm called Concurrent Priority Threads Algorithm (CPTA) to guarantee three things: smooth execution of multiple threads, speed up the work of CPU, and prevent the CPU from harm due to the influx of multithreaded with the same priority, In this paper we focuses on the smooth run of multiple threads to reduce the processing time.

2. BASIC PRINCIPLES

Traditionally such reactive software systems have been designed as shared-memory multi-process or multi-threaded programs [1]. One of the available options for the user depending on the scale of the experiment is to create a control scheme from scratch, using a programing language and operating system of choice, with the use of helpful guides and various libraries [2]. The program P has all the statements that belong to a program slice which affect the value of variable v at proper point p [3].

Dynamic Slicing is more benefit in program debugging, testing, program understanding, software maintenance, and present with android systems (e.g., [4], [5], [4], [6]) and [7].

This paper is focused on modifying an effective dynamic slicing algorithm for multithreaded programs with the same priority threads. The algorithm would help in effective way of more than two threads that have been supported with the maximum 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 catalyze the rate of execution with thread priority at the top side.

3. RELATED WORK

This section presents the methods that related with multithreading prioritization and make a comparison among these methods in respect to four categories (time, aspect, accuracy and kind of system) as illustrated in table 1.

a. ACE Method

The ACE method is depending on three resources the Application, the Control Unit, and External data. The idea is each thread has its own priority and the issue unit serves the threads according to their priorities. The priority p of a thread i is computed as follows: pi = min(ai, ei) + ri when ri = min(mi, bi + si). bi is the lowest or base priority of thread i; mi is the highest priority and si is an offset to adapt the current priority by the application, user defined, or the Control Unit. ei is an external data limited by ai to the priority assumed by application and control unit, even to a new bounded but higher maximum value. When the operating system want to rapid run to external interrupt routines or to release a critical resource as fast as could, then the feature of arise a priority boost for a short time will be useful in this case [8]. Figure 2 shows the range for single and multiple threads priority can use at run time.

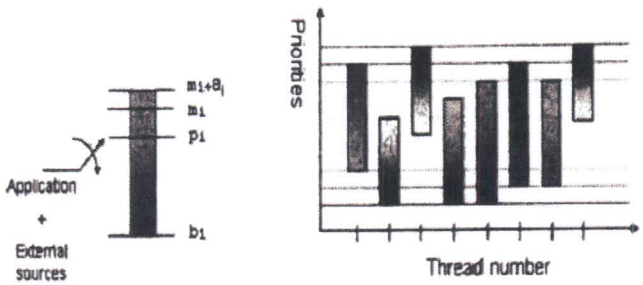


Figure 2: Range of Priority for single and multiple threads [8]

b. Semaphore coprocessor

It is an extension for the ACE method by putting some sort of semaphore coprocessor since yet another impact in which threads affect each other and also which usually makes use of the external input pertaining to handling the thread priority. Selecting the semaphore coprocessor covers the two factors, thread stalls due to synchronization between software threads and also synchronization that has a hardware unit. Equally are usually related in numerous parts of embedded systems and also servers [9]. The semaphore coprocessor, shown in figure 3, handles a configurable variety of resources. Each resource has a queue plus a token Si, i ∈ 0..k. The lock and also unlock instructions usually are put into the instruction set of the processor so the issue unit passes all instructions towards the coprocessor. A certain token is requested by a lock instruction. When the instruction arrives and if the token is available then the coming instruction will be served at once. Else, it is buffered and the thread is paused until the occupying thread releases the token. If a thread asks for a token and this token is allocated by another thread, then to avoid the long pause for the waiting thread the priority of the allocation thread will be raised. There are two directions to modify the priority:

1. A static value ei would increases the priority so that the priority of a thread is always will be the highest if there is at least one thread requiring the same token, or

2. It is increased by a dynamic value ei. In this case, ei depends on the number of threads waiting for the token. So, ei = c * #queue where c is constant and #queue is the current queue length [9].

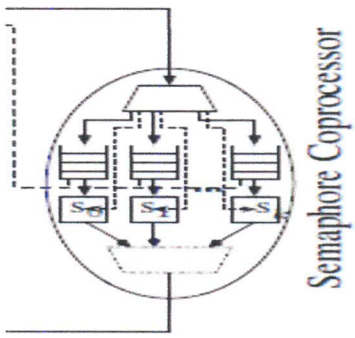


Figure 3: Semaphore coprocessor [9]

c. DYNAMIC SLICINGTHREADS

The programming languages which have an object-oriented base are more suitable than the other languages to dynamic slicing technique. A dynamic slicing criterion specifies the input, and takes into account different occurrences of a statement in an execution trace [10]. The input and the program statement are belongs to the run trace which in turn belongs to dynamic slicing criterion. 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 [11].

The execution of many threads concurrently started with a global thread can be done by abstraction of two kinds of dependence named Dynamic thread dependence graph (DTDG) and Priority multithreaded dependence graph (PMDG).

Table 1: Comparison among multithreading prioritization methods

Methods / Findings	ACE	Semaphore Coprocessor	Dynamic Slices
Time	Reduce time with very low cost	Reduce time with resource increasing	Reduce time processing
Aspect	S.W., M.W. and H.W.	H.W.	S.W.
Accuracy	YES	YES, with more needs for processor resources as for semaphore	YES

In this research we focused on the fourth method which is dynamic slicing threads.

4. PROPOSED ALGORITHM

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 Concurrent Priority Threads Algorithm (CPTA) and would be computing all the threads in the multithreaded program with the same assigned priority. The threads would be synchronized 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 [12]. 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.

Concurrent Priority Threads Algorithm steps:

Input: Threads in a priority based multithreaded program, dynamic slicing criterion <S, v, E, i>. Output: Processed Threads with highest 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 5 and Y=1 to 5;

(d) if(threadX==MAX_PRIORITY)

(e) then Print thread X processing

(f) if(threadY==MAX_PRIORITY)

(g) then Print threadY processing

(h) if(threadX and threadY==MAX_PRIORITY)

(i) then Print threadX and threadY processing

(j) else Print threadX pre-empted; threadX.suspend()

(k) else Print threadY pre-empted; threadY.suspend()

Step 3: Resume the suspended threads

(a) threadX.resume();Print threadX processed

(b) threadY.resume();Print threadY processed

Step 4: End

5. EXPEREMINTAL RESULTS

The purpose of the experiment of the algorithm for Priority Multithreading dependence graph is to reduce the processing time of the CPU by taking just the statements that have effect on the thread segment. Also, for the execution stage of the thread the algorithm will increase the run rate in respect to the highest priority threads.

Figure 4 is a bar graph represents the threads with the corresponding priorities; we can see that the threads starts from the highest for thread Z and decreases for all threads until reach the lowest priority for thread A (i.e. the execution is sequential). The statements for this graph in the program take the code:

```
73  class ThreadPriorityDemo
74  {
75  public static void main(String args[])
76  {
77  A threadA=new A();
78  B threadB=new B();
79  C threadC=new C();
80  X threadX=new X();
81  Y threadY=new Y();
82  Z threadZ=new Z();
83  threadZ.setPriority(Thread.MAX_PRIORITY);
84  threadY.setPriority(threadX.getPriority()+1);
85  threadX.setPriority(threadC.getPriority()+1);
86  threadC.setPriority(threadB.getPriority()+1);
87  threadB.setPriority(threadA.getPriority()+1);
88  threadA.setPriority(Thread.MIN_PRIORITY);
```

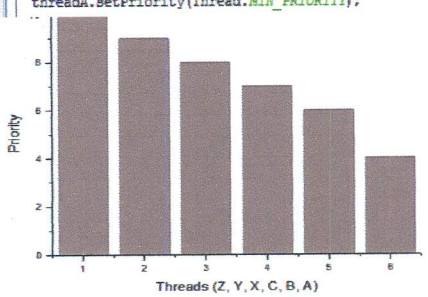
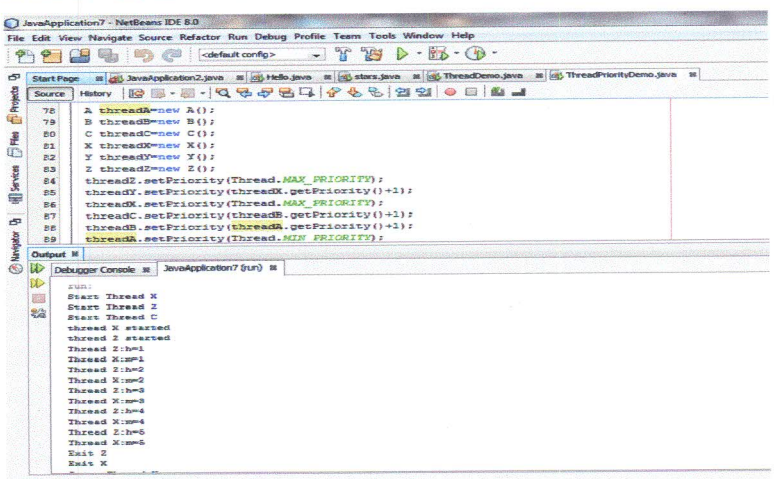


Figure 4: priorities of the threads

Moreover, if we have two threads with the same maximum priority as thread Z with thread X, the code and the graph would be as below.



Here, first we specify the priorities for the threads so that gave thread Z and X the maximum priorities, the other threads with lower priorities. Second, we could get this result that the two threads Z and X with the maximum priorities will be executed concurrently and then the other threads with the lower priorities after two times of execution.

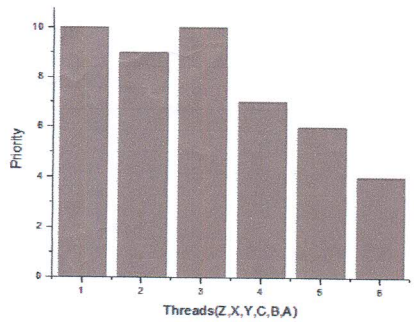


Figure 5: priorities concurrently for threads Z and X

6. CONLUSION AND FUTURE WORK

Dynamic slicing algorithm would help the threads with the same priorities to be executed simultaneously, where the CPU allocate equal time for execution to the threads with the highest priorities. It guarantees the rapid execution rate to the program, and less time consuming by providing the critical resources for the threads. Also it has the flexibility to change the priorities of the threads without harm the CPU or slow down the path of running. Moreover, 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. For the future work, we attend to add more threads with two CPUs, and make the threads dependable on each other.

7. REFERENCES

[1] M. Emmi, A. Lal, and S. Qadeer, "Asynchronous programs with prioritized task-buffers," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 2012, p. 48.

[2] N. Sinenian, A. B. Zylstra, M. J.-E. Manuel, J. A. Frenje, A. D. Kanojia, J. Stillerman, et al., "A multithreaded modular software toolkit for control of complex experiments," Computing in Science & Engineering, vol. 15, pp. 66-75, 2013.

[3] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in Proceedings of the first workshop on Hot topics in software defined networks, 2012, pp. 79-84.

[4] D. Binkley, S. Danicic, T. Gyimóthy, M. Harman, Á. Kiss, and B. Korel, "Theoretical foundations of dynamic program slicing," Theoretical Computer Science, vol. 360, pp. 23-41, 2006.

[5] R. Gupta, M. J. Harrold, and M. L. Soffa, "An approach to regression testing using slicing," in Software Maintenance, 1992. Proceerdings., Conference on, 1992, pp. 299-308.

[6] K. B. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," Software Engineering, IEEE Transactions on, vol. 17, pp. 751-761, 1991.

[7] L. LIU, R. GU, and B. XU, "Design and Implementation of Multi-Thread Interaction Based on Android," in International Conference on Management and Engineering (CME 2014), 2014, p. 11.

[8] A. Doring and M. Gabrani, "On networking multithreaded processor design: hardware thread prioritization," in Circuits and Systems, 2003 IEEE 46th Midwest Symposium on, 2003, pp. 520-523.

[9] C. Albrecht, A. C. Doring, F. Penczek, T. Schneider, and H. Schulz, "Impact of coprocessors on a multithreaded processor design using prioritized threads," in Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on, 2006, p. 7 pp.

[10] N. Sasirekha, A. E. Robert, and D. M. Hemalatha, "Program slicing techniques and its applications," arXiv preprint arXiv:1108.1352, 2011.

[11] N. Walkinshaw, M. Roper, and M. Wood, "The Java system dependence graph," in Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on, 2003, pp. 55-64.

[12] D. P. Mohapatra, R. Mall, and R. Kumar, "An overview of slicing techniques for object-oriented programs," Informatica (Slovenia), vol. 30, pp. 253-277, 2006.