# The Smart Energy Management of Multithreaded Java Applications on Multi-Core Processors

**Kuo-Yi Chen**
*Department of Computer Science and Information Engineering,*
*National Formosa University,*
*Taiwan, R. O. C.*
*E-mail:kuoyichen@gmail.com*

**Fuh-Gwo Chen**
*(*Corresponding author)*
*Department of Computer Science and Information Management,*
*Hung Kuang University,*
*Taichung, Taiwan*
*E-mail: fgchen@gmail.com*

Received 21 March 2012
Accepted 12 July 2012

Multi-core processors are becoming widely deployed. However, more cores consume more power. A power saving technique for multi-core systems based on the observation of critical sections is proposed. Since only one thread works in a critical section, other threads on other cores that access the same resource would be busy waiting. The proposed power-saving technique leads to the energy savings (11 to 15 percent) and the lowest values of Energy Delay Product as compared with the other power-saving techniques.

*Keywords*: Power-saving, Java, Synchronization, Multi-core processors, Multi-threaded applications, JVM

## 1. Introduction

The significant advances of VLSI technology lead to multi-cores on a physical chip nowadays. With the decreasing prices, multi-core processors are widely deployed in both server and desktop systems. The performance of multi-threaded applications could be improved on multi-core based systems because the workload of threads could be dispatched to cores, which work in parallel.

More cores imply more power consumption. [1] Though power-saving technique is important to reduce energy wastages of processors, the performance of applications should be maintained when power-saving techniques are applied. Therefore, to find a technique to save power and keep the performance of applications is an important issue with the use of multi-core processors.

The most adopted power saving technique for current multi-core processors is the ability of dynamic frequency tuning which is based on Dynamic Voltage and Frequency Scaling (DVFS). Many studies use DVFS to adjust the frequency of processor cores, and then, to save power. [2] These researches could be classified into two groups: *profiling* and *performance monitors*.

The power-saving techniques of *profiling* rely on the measurement of the behaviors of applications first, and then, analyze the measured results and find an approach to tune the frequency of processors. However, the profiling techniques require extra costs, such as code analysis and special instructions inserted into the target applications. Due to the significant overhead of extra costs, the profiling power-saving techniques are not applied widely. [8] [9]

The frequency of multi-core processors could be tuned according to the information collected by the hardware *performance monitors* (PMs). The particular behavior of applications, such as CPU usages, could be observed by PMs in run-time. Based on the observations of PMs, the phases of applications could be determined, and then the frequency of processor cores could be tuned to save power without significant overhead. [5] [6]

The power-saving techniques based on PMs are simple and effective, thus they have been already applied on various systems, such as power governors of Linux. One drawback of PMs is that phases are identified after their information is collected by PMs. The information is always one step behind. Thus the phases of applications cannot be determined by PMs precisely. The inaccurate engage/disengage timing of phases could lead to inappropriate CPU frequency tuning, and result in the degradation of system performance and energy wastages.

In order to improve the limits of profiling and PMs power-saving techniques, we propose an approach that takes *critical sections* as targets to reduce the power consumption of multi-threaded applications. Threads use critical sections to synchronize the access of shared resources. A thread enters a critical section for accessing a shared resource exclusively. It is worth noting that only one thread, hence one core, can work in a critical section, and other cores are busy waiting, if they are to access the same resource. Thus critical sections could be considered as a special phase of multi-threaded applications. Moreover, due to the busy waiting cores could lead to energy wastages, critical sections could be a power-saving points of multi-threaded applications.

In order to exploit the power-saving potential of critical sections, our motivation is to develop a power-saving technique based on the detections of critical sections using the run-time information of a *Java virtual machine* (JVM). In Java, [3] all instructions of applications have to be interpreted by a JVM. The behavior of an application could be observed precisely with very little overhead in run-time, including critical sections.

Furthermore, the use of run-time information of a JVM can improve the disadvantages of profiling and PM power-saving techniques. First, the run-time information is readily available to a JVM, the extra cost of profiling could be reduced. Secondly, due to phases of applications could be detected precisely by the run-time information of a JVM, the disadvantages of ambiguous phase determinations of PM power-saving techniques could be also improved. [10]

The research steps of this study are performed as follows. Firstly, the behaviors of critical sections in Java are studied. Secondly, the technique of critical section detection is developed by the run-time information which is readily available in a JVM. Thirdly, the power-saving technique of critical sections is proposed. Finally, the proposed approach is compared with other power-saving schemes to demonstrate its performance.

The experimental results show that power-saving techniques of critical sections leads to significant energy reductions (11 to 15 percent) with the multi-threaded Java benchmarks, which is better than the other power-saving techniques (6 to 9 percent). Moreover, the performance degradation of power-saving techniques of critical sections is only one percent, which is much better than the use of other power-saving techniques (7 to 12 percent). As a result, the power-saving technique of critical sections can lead to the lowest values of *Energy Delay Product* (EDP) among the other power-saving techniques without additional costs. A preliminary version of this paper is published. [7] As compared with, [7] we enhance the detection of critical section detections to bytecode levels, and introduce more power-saving opportunity with new power-saving algorithms.

The organization of the research is as follows. Section 2 describes the opportunity of power saving in critical sections. The synchronization methods of Java are described in Section 3, with the proposed approach of critical section detections. The proposed power saving algorithm is shown in Section 4. The experiment set up and experimental results are shown in Section 5, followed by the conclusion in Section 6.

## 2. The power-saving opportunity of critical sections

When a shared resource is accessed by a thread exclusively, the particular period could be considered as in a critical section. No matter how many processor cores are available, if the threads on all cores want to enter the same critical section simultaneously, only one core can work and other cores are busy waiting. This observation leads the power-saving opportunity as follows: the frequency of the cores that are busy waiting could be minimized to reduce energy wastages. As a comparision, we consider a single core case first and assume all threads execute round robin as shown in Figure 1. Due to the time sharing between threads, the critical sections would not lead to the purely busy waiting status.

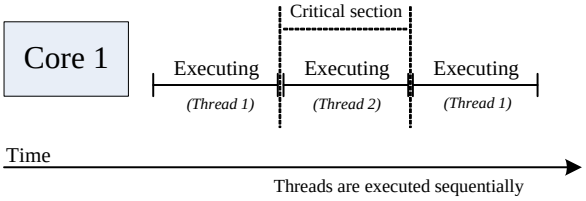


Figure 1. Critical section in the configuration of a single core.

On the other hand, the power-saving opportunity of critical sections could be observed with the use of multi-core processors with multi-threaded applications. Consider an extra case in Figure 2, a multi-threaded application (thread 1 to 4) is executed in parallel and all threads want to enter the critical section for accessing the same resource at the same time, and only thread 1 succeeds. The other threads (thread 2, 3 and 4) become busy waiting on thread 1 to exit the critical section. After thread 1 exits critical section, thread 2 enters critical section and access the same resource. That shows only a thread is executing in a critical section among the threads which access the same shared resource.

This observation shows in a critical section, the busy waiting cores could be tuned to save power without performance degradations. For example, in Figure 2, the frequency of busy waiting cores (core 2, 3 and 4) can be minimized to reduce the energy wastages. Moreover, the performance of applications would be maintained because only busy waiting cores are tuned.

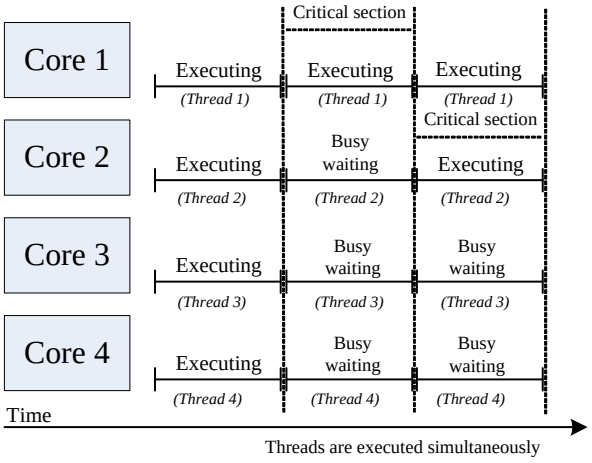


Figure 2. Power-saving opportunity of multi-core processors.

## 3. The synchronization methods of Java

The scoped lock is widely used as a synchronization mechanism to guarantee the atomic access of shared resources. The scoped lock could be considered as the automatic lock obtaining and releasing mechanism. In Java, a scoped lock is declared by the use of the keyword, *synchronized*. In the sample code, shown in Figure 3, the method Test() is declared as synchronized in line 2. With this declaration, the Test() method can be guaranteed its atomic access.

```
1: public int Test() {
2: synchronized(this) {
3:         try {
4:                 //do something
5:         } catch (Exception e) {
6:                 //without releasing lock explicitly
7:                 return -1;
8:         }
9:         }
10:        return 0;
11: }
```

Figure 3. The sample code of scoped locks in Java

Java synchronization methods not only guarantee the atomic access of shared resources, but also lead to the critical sections, which are the power-saving points of multi-threaded applications. As all instructions (bytecodes) of Java applications have to be interpreted by the Java interpreter, the synchronization codes can be observed before they are executed. Therefore, the critical sections can be detected in advance to exploit the power-saving opportunity.

Since critical sections are the power-saving point of multi-threaded applications which are executed on multi-core systems, the development of critical sections detection is important. The critical section detection is based on the use of synchronized keyword and Java *bytecode interpreter*.

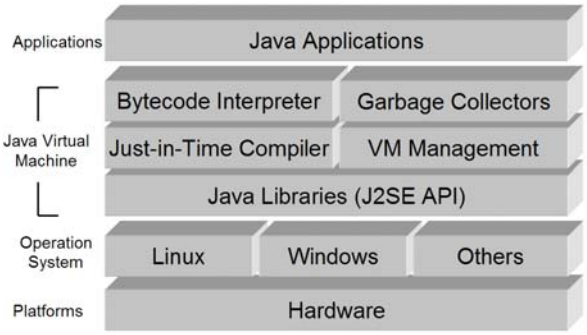


Figure 4. The structure of Java virtual machines

The bytecode interpreter is an important software component of a JVM, whose structure is in Figure 4. In order to reach platform independency, Java applications are compiled to the class files with *bytecodes*, the special instruction set of Java. All instructions of Java applications have to be simulated by the bytecode interpreter. Hence, the particular bytecode patterns could be identified before they are actually fired on hardware.

The keyword, *synchronized*, would be compiled to the particular bytecode patterns. Moreover, critical sections would be generated by these particular known bytecode patterns in run-time. Thus critical sections could be identified in advance before they are actually executed on hardware by ehancing the interpreter to match the patterns. In Java, the use of synchronized keyword would be compiled to two bytecode patterns for Java objects and methods. The bytecode pattern of synchronized Java objects is explained first. A piece of sample code with the use of *synchronized* keyword for Java objects and its compiled bytecodes are shown in Figure 5 and Figure 6.

```
1: public class test {
2:         public static void main(String[] args) {
3:                 synchronized(new Object()){
4:                         int i = 0;
5:                 }
6:         }
7: }
```

Figure 5. The sample code of synchronized objects

As shown in Figure 5, the use of synchronized keyword would guarantee the atomic access of the shared resource, Object, in line 3. That means a critical section of this shared resource would be generated in run-time. In order to create a critical section for this synchronized object, the line 3 in Figure 4 would be compiled to the particular pattern of a pair of bytecodes, *monitorenter* and *monitorexit*, as shown as line 6, 11 and 15 in Figure 6.

```
1: new  #2; //class Object
2: dup
3: invokespecial#1; //Method java/lang/Object."":()V
4: dup
5: astore_1
6: monitorenter
7: iconst_0
8: istore_2
9: nop
10: aload_1
11: monitorexit
12: goto 23
13: astore_3
14: aload_1
15: monitorexit
16: aload_3
17: athrow
18: nop
19: return
```

Figure 6. The bytecodes of synchronized objects

When the *monitorenter* bytecode is translated by the interpreter, a lock would be acquired atomically for this synchronized object. Thus the *monitorenter* bytecode

could be used to indicate the start of a critical section. The lock would be released by two *monitorexit* bytecodes. They could be used to indicate two possible ends of this critical section. The possibility of two ends of this critical section is due to the exception handling of Java. Based on the observation of this particular bytecode pattern, a pair of *monitorenter* and *monitorexit*, the critical section of a Java object can be detected before it is executed on hardware.

On the other hand, a Java method could be accessed by multiple threads, thus the atomic access of Java methods is important in multi-threaded applications. The bytecode pattern of synchronized Java methods is different from Java objects. In order to use this pattern, a piece of sample code with the use of synchronized keyword for Java methods is shown in Figure 7.

```
1: class Company{
2:         private int sale = 0;
3:         synchronized public void sell(int qty){
4:                 int sum = sale;
5:                 sum += qty;
6:                 sale = sum;
7:         }
8: }
```

Figure 7. The sample code of synchronized methods

As shown in Figure 7, the use of synchronized keyword guarantees the atomic access of Java method, *sell*, in line 3. That means a critical section would be generated when this method is accessed in the run-time. In order to notify the JVM to create a critical section for this synchronized method, this method would be compiled with a special value, *ACC_SYCHRONIZED*, for its property flag. This property flag would be checked by a JVM when this method is invoked.

When a synchronized method is invoked, a monitor would be acquired by the current thread to guarantee the atomic access of this synchronized method automatically. This monitor would be released whether this method invocation completes normally or abruptly. When the executing thread owns the monitor, other threads cannot acquire it. Thus the atomic accesses of synchronized Java methods could be reached.

Based on the study of synchronized bytecode patterns, the pair of *monitorenter/monitorexit* and property flag, ACC_SYNCHRONIZED, critical sections could be identified by a bytecode interpreter. Thus the algorithm of detection of critical sections is proposed as follows. As shown in Algorithm 1, a critical section could be detected by recognitions of the two-bytecode pattern. Thus the accurate engage/disengage timing of a critical section could be identified. With the accurate timing information of a critical section, the frequency of busy waiting processor cores could be adjusted in advance, and then the power-saving opportunity of critical sections can be exploited.

**Algorithm 1** The critical section detections.

**Require:** The monitorenter bytecode *MonEnter*
**Require:** The monitorexit bytecode *MonExit*
**Require:** The property flag *ACC_SYNCHRONIZED*
  **if** Current Bytecode = *MonEnter* **then**
    *Critical sections of Objects* $\rightarrow$ *Start*
  **end if**
  **if** Current Bytecode = *MonExit* **then**
    *Critical sections of Objects* $\rightarrow$ *End*
  **end if**
  **if** Property flag = *ACC_SYNCHRONIZED* **then**
    **if** Method is invoked **then**
      *Critical sections of Methods* $\rightarrow$ *Start*
    **end if**
    **if** Method is completed **then**
      *Critical sections of Methods* $\rightarrow$ *End*
    **end if**
  **end if**

Secondly, the proposed critical section detection algorithm also improves the two disadvantages of the PM power-saving approach. First, application's phases must be identified after the information appears in PMs. It is always one step behind. Secondly, the actual engage/disengage time of a given phase is not known. These disadvantages are due to the nature of using PMs which are at the lower level of a computer system. Conversely, the proposed critical section detection uses the run-time information which is from higher levels (e.g. bytecode patterns), thus these limitations can be improved.

Since the power-saving point, critical sections, can be detected before they are actually executed on hardware, the busy waiting periods of processor cores can be identified. Thus the energy wastages of busy waiting periods could be reduced by CPU frequency tuning.

## 4. Power-saving algorithms of critical sections

Based on the critical section detections, the accurate engage/disengage timing of a critical section could be determined. Thus the frequency of busy waiting processor cores could be minimized to save power. The power-saving algorithm of critical section (CS) detections is shown as follows.

**Algorithm 2** Power-saving algorithm of CS detections.

**Require:** The start of a critical section *CSstart*
**Require:** The end of a critical section *CSend*
**Ensure:** The executing core *Ce*
**Ensure:** The other busy waiting cores *Cw*
  **if** Critical section = *CSstart* **then**
    $Ce \leftarrow Maximum\ CPU\ Frequency$
    $Cw \leftarrow Minimum\ CPU\ Frequency$
  **end if**
  **if** Critical section = *CSend* **then**
    $Cw \leftarrow Maximum\ CPU\ Frequency$
  **end if**

In algorithm 2, the *CSstart* and *CSend* can be determined by the bytecode patterns in the run-time. Thus the power level of busy waiting cores could be adjusted based on the precise engage/disengage timing of critical sections. Due to the accurate timing, the frequency can be adjusted correctly, and then reduce more energy wastages than the use of PMs. Moreover, because critical sections can be detected in run-time, this power-saving algorithm works without the extra costs of profiling.

## 5. Experiments and evaluations

The experiment platform is has a four-core processor, Intel Q6600 Quad-Core CPU. The frequencies of each core can be adjusted independently from 1.8 to 2.93 GHz. The operating system is Fedora Core 8 with kernel version 2.6.24. The experiments use the HotSpot of OpenJDK. The latest version, OpenJDK 1.7, is built for experiments.

Five widely used multi-threaded Java benchmarks are used in this study. They are *Eclipse*, *Hsqldb*, *Lusearch* and *Xalan* from *Dacapo*, and *SPECjbb2005*. These multi-threaded benchmarks are of different types of workloads, and they could represent the common features of general applications.

For Example, Hsqldb executes a number of transactions against a model of a banking application via a JDBCbench-like in memory. The text searching and XML transformation of Lusearch and Xalan are usually seen in text editors and browsers. The SPECjbb2005 processes the complete business logic with multiple warehouses. Based on experiments of these benchmarks, the performance and power consumption of multi-threaded applications could be observed and compared

In order to compare the proposed power-saving algorithms with other power-saving approaches, four different power schemes are used to compare in this experiment. First, two static CPU frequencies are used as the control group, the maximum and minimum CPU frequencies. They are mapped to the *performance* and *power-saving* governors in Linux. Secondly, two widely used Linux power-saving governors, *conservative* and *ondaemon*, are used as PM power-saving approaches. The ondaemon governor changes CPU frequency based on processor usages as the events of performance monitors. On the other hand, the conservative governor can be considered as a more gradual on-demand.

To evaluate the power consumption of these benchmarks, the measurement of CPU power consumption is an important issue. A generic dynamic power measurement of CMOS circuits is used in these experiments. The dynamic power consumption of processors can be expressed as follows.

$$P = C \times V_{dd}^2 \times f \quad (1)$$

In Equation 1, *C* is the effective switching capacitance, *Vdd* is the supply voltage and *f* is the executing frequency. It is worth noting that the observation of lower power consumption might not lead to better power efficiency. In general, the worst performance of applications usually could be observed while the lowest processor frequency is applied. The worst performance usually leads to the longest execution time, and then results in significant power consumptions. The performance of power-saving techniques cannot be evaluated appropriately only by the values of power consumption.

To evaluate the performance of power-saving schemes appropriately, the executing time and power consumption of benchmark applications should be considered at the same time. Therefore, EDP is used as

a comprehensive measurement in experiments. The value of EDP is the product of energy and execution time of applications. The lower value of EDP usually indicates the better performance of power-saving techniques. With the use of EDP, the power consumption and execution time of benchmarks could be estimated at the same time and the performance of different power-saving techniques could be evaluated appropriately.

The power consumption is shown in Figure 8. In the worst performance of benchmarks, the use of minimum CPU frequency leads to the significant power consumption among all benchmarks. On the other hand *conservative* and *ondaemon* lead to slight reductions of power consumption (6 to 9 percent). It is worth noting that the proposed power-saving technique leads significant reductions of energy (11 to 15 percent), which is better than the use of *conservative* and *ondaemon*. This observation shows that the proposed power-saving algorithm not only keeps benchmark's performance well, but also reduces energy wastages significantly.

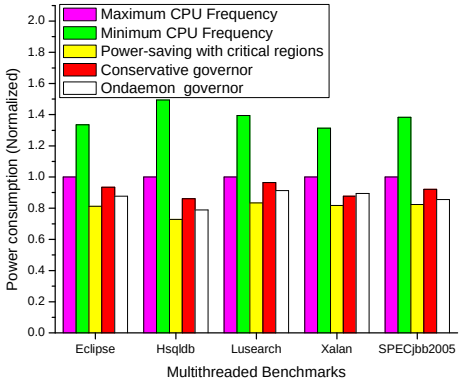


Figure 8. The Power consumption of power-saving techniques

Finally, the EDP values are shown in Figure 9. The lowest EDP values (82 percent) could be observed with the use of proposed power-saving technique. On the other hand, the EDP values of *conservative* (94 percent) and *ondaemon* approaches (96 percent) are higher than the proposed approach. Based on the comparison of the EDP values, the performance of the proposed approach is better than other power-saving approaches.

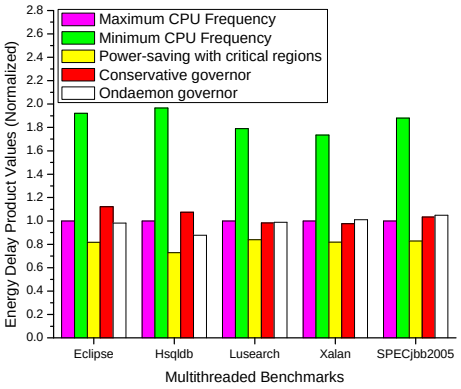


Figure 9. The EDP values of power-saving techniques

According to these experimental results, three advantages of the proposed approach could be concluded. First, while other approaches suffer from performance degradation, the proposed approach keeps the performance of benchmarks well. Secondly, significant energy wastage could be reduced by the proposed approach, which is better than the use of other power-saving approaches. Finally, the lowest EDP values indicate that the comprehensive performance of the proposed power-saving techniques. Thus the goals of this study, power-saving and very little performance degradation, are reached.

## 6. Conclusion

In this research, the power-saving opportunity, critical sections of multi-threaded applications, is studied. Due to atomic accesses of the shared resource, only one processor core can work in a critical section, if all the threads on cores are to access the same shared resource, while other processor cores are busy waiting. These busy waiting cores do not improve system performance, but waste energy. We propose the approach to identify such critical sections in advance and hence to reduce the energy wastages by tuning the busy waiting cores into the lowest frequency.

The proposed approach with other related approaches are implemented on Hotspot. Five widely used multi-threaded Java benchmarks are used to evaluate the performance of power-saving techniques.

The experimental results show that the proposed approach leads to well performance maintenance, significant reductions of energy wastes and the lowest EDP values among other power-saving approaches. These experimental results demonstrate the effectiveness of proposed power-saving technique, and show its better performance among the other power-saving approaches.

## 7. Acknowledgments

This work was supported by the Taiwan National Science Council (NSC) under Grant No. 101-2220-E-001-001.

## 8. References

[1] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1277-1284, 1996.

[2] T. Seki, *et al.*, "Dynamic voltage and frequency management for a low-power embedded microprocessor," *IEICE Transactions on Electronics*, vol. 88, p. 520, 2005.

[3] T. Lindholm and F. Yellin, *Java virtual machine specification*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[4] M. Weiser, *et al.*, "Scheduling for reduced CPU energy," *Mobile Computing*, pp. 449-471, 1996.

[5] C. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," *ACM SIGPLAN Notices*, vol. 38, pp. 38-48, 2003.

[6] M. Curtis-Maury, *et al.*, "Prediction-based power-performance adaptation of multithreaded scientific codes," *IEEE Transactions on Parallel and Distributed Systems,*, vol. 19, pp. 1396-1410, 2008.

[7] Kuo-Yi Chen, Fuh-Gwo Chen and Ting-Wei Hou, "The power-saving approach by critical section detections of Multi-cores Embedded Systems", in *Proc. of International Conference on Mechanical and Electronics Engineering* (ICMEE 2010), vol.1, pp. 117-121, Kyoto, Japan. August 1-3, 2010.

[8] Flinn, J. and Satyanarayanan, M., "PowerScope: A tool for profiling the energy usage of mobile applications", *Mobile Computing Systems and Applications*, WMCSA'99. Second IEEE Workshop, pp. 2-10, 1999.

[9] Simunic, T. and Benini, L. and De Micheli, G. and Hans, M, "Source code optimization and profiling of energy consumption in embedded systems", *Proceedings of the 13th international symposium on System synthesis*, IEEE Computer Society, pp. 193-198, 2000.

[10] Kuo-Yi Chen, Chin-Yang Lin, Tien-Yan Ma and Ting-Wei Hou, "A Power-saving Technique for the OSGi Platform", *IEICE Transactions on Information and Systems*, Vol.E95-D, No.5, pp.1417-1426, May, 2012