Dynamic Interrupt Controller and Conflict Management for Transactional Memory in Embedded System

Jun Young Moon[1], **Jun Gil Ahn**[2] **and Jong Tae Kim**[1,2]
[1]Department of IT Convergence, Sungkyunkwan University, Suwon, South Korea
[2]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, South Korea

Abstract – *In hardware transactional memory system, selecting an interrupt handling mechanism is the one of problems. To handle interrupts occur in transactions, all systems need special mechanisms but these require more hardware or software resources, so this is not acceptable to the embedded system that has limitations. In this paper, we proposed interrupt handling process and interrupt controller that distributes interrupts to proper core in the system. Then we present performance metrics for the system and how transactional memory policies affect to the metrics. Simulation results show that roughly 80% of interrupts that occur in transaction can be improved and eager conflict detection, polite and polka contention managements are proper polices for the proposed system.*

Keywords: Transactional memory, parallel computing, interrupt handling, contention management, conflict detection

1 Introduction

In multi-core system, the complexity of lock-based synchronization is well-known problem to almost programmers. The transactional memory (TM) is proposed to solve the problem [1]. TM system traces all memory operations in critical section and records these in special buffers. If the critical section is committed, all memory operations is written to memory. In contrast, if the critical section is aborted, information about the critical section in the buffer is discarded and TM system retries the critical section.

There are some problems to implement a real hardware transactional memory (HTM) system for embedded systems. One of the problems is how to handle interrupts in transaction time that is interval between transaction start and end. In an embedded system, decreasing response time of user inputs is critical to the performance of whole system, because almost applications implemented by an embedded system want to response to users as soon as possible. As a result how to handle interrupts in HTM for embedded systems is more important than for high performance computer systems.

The traditional method to handle interrupts that occur in transaction time is that the interrupts have to be pended or the transaction has to be aborted [2]. This is simple and easy to implement by hardware. But this method has disadvantages because the average response time is decreased so this is not applied to embedded systems directly.

To solve the problem, some types of TM use special mechanisms [3, 4, 5]. Nested institutes a nested transaction concept, so the interrupt handler can be executed in transaction time [3]. To implement nested transaction, the complexity of cache structures and the size of buffers is more increased. VTM can suspend transactions so interrupt can be handled in transaction time [4]. To support that, it uses virtualization concept and need additional hardware controller, memory spaces and software layers to save and restore overflowed or suspended transaction data. MetaTM uses stacked transaction concept to handle interrupts [5]. By using transaction push-pop primitives, interrupt handler can suspend and restore transaction information. Those solutions always need more hardware buffer to store transaction state and complex TM controller. As a result, those also have disadvantages to embedded systems. Because embedded systems that always want to less cost, power and size and the systems have more strict criteria to hardware and software resource than high performance computer. Eventually, HTM for embedded system need interrupt handling mechanism that is easy to implement and has short response time.

In this paper, we focused on the interrupt handling process in TM for embedded systems. First, we present simple interrupt controller for TM to be used on embedded system. Second, we proposed some performance metrics for interrupt handling in the proposed controller and evaluate performance of TM systems by using the metrics. Then we analyze how many interrupt handlers can be improved and how choosing policies affect performance of interrupt handling and propose the best policy for the suggested interrupt handling process.

In section 2, we present the interrupt handling process in the proposed system and suggest the performance metrics. In section 3, we analyze the effects of contention management policies and conflict detection methods to proposed metrics. In section 4, we explain the evaluation environment to check tendencies explained in section 3. In section 5, we describe the result of experiments. In section 6, we discuss the conclusion of this paper.

2 Interrupt handling

In an embedded system, overheads needed to implement HTM system should be small. That means architecture of the HTM should have only essential components that has to be remained in the architecture to implement transaction concept. For example, transaction buffers and modified buses should be always in the architecture. In contrast, supporting to handle interrupts that happen in transaction time is not an essential part to the system. Because, by using pending interrupt concept, all interrupts in transaction time always can be handled correctly. Only drawback is increasing response time. To reduce that, we modify an interrupt controller that almost embedded systems already have.

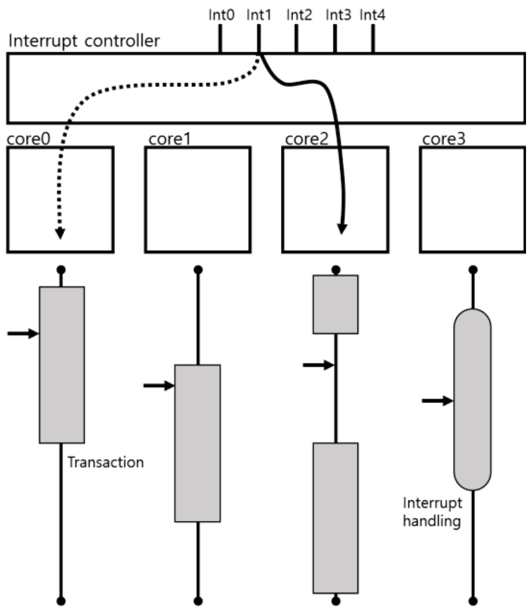

Figure 1. Dynamic interrupt distribution for TM

2.1 Interrupt controller

Because the proposed architecture cannot support handling interrupts happened in transaction time, interrupts should be sent to empty core that doesn’t handle interrupt or execute transaction at that time as much as possible to decrease response time. To find empty cores, the interrupt controller read TM status register that each core already have. The TM status registers have the information about whether each core execute a transaction. Interrupt status of each core is stored in interrupt controller registers. Almost interrupt controller in multi-core system can distribute interrupts to each cores and select what core is a target processor for a specific interrupt. As a result it may be not expensive that overhead to implement the proposed interrupt controller.

Figure 1 shows the concept of the suggested interrupt controller. The proposed interrupt controller can know whether cores are busy or empty. In figure 1, the state of core0, core1 and core3 are busy because core0 and core1 execute a transaction and core 3 handles an interrupt. But the core2 is empty core because it does not execute any transaction or handle an interrupt at that time. If the interrupt 1 is happened at that time, the interrupt controller try to send the interrupt to the core 0 that is decided by target register statically. But, the interrupt controller know the core0 is busy so that it modifies the target core for the interrupt 1 from core0 to core2 dynamically to decrease the response time for the interrupt 1.

2.2 Performance metrics

To evaluate the interrupt handling performance of the suggested TM system, there are three types of performance metrics.

- The ratio of occurring interrupt in transaction time. If the number of interrupt is few, the average response time of all interrupts is decreased.

- The interrupt response time of occurring interrupt in transaction time. Because the proposed system using the interrupt pending mechanism, interrupts that occur in transaction time have to wait the transaction end if there are not empty core.

- The number of empty cores at occurring an interrupt in transaction time. If existence probability is high, the average response time of interrupts is decreased.

The last is important to the suggested controller since it impacts to the first and second metric. If the existence probability of empty cores is increased, the ratio of interrupts that occurs in transaction time and how much the second metric affects to the total performance of interrupt handling are decreased. By using the metrics, we find the effective TM policies for the proposed interrupt handling system.

3 TM policies

In TM system design, choosing contention management and conflict detection are the most important steps, because the performance of almost TM systems is sensitive to the policies [6]. Since finding the most effective policy for the proposed architecture is important.

3.1 Existing TM policies

There are popular contention management policies are proposed in [6]. The polite policy uses the randomized back off increased exponentially. The karma policy has priority concept to determine what transaction will be aborted and uses fixed back off. The timestamp policy also employ fixed back off, but priority of transaction is start time of each transaction. The karma and polite combine to form the polka policy. The priority in the karma and back off in the polite are used in the polka policy. The conflict detection mechanisms are eager and lazy [7]. The lazy conflict detection checks whether there is a conflict between transactions at only commit time. But the eager conflict detection detects a conflict at every memory operations in transaction.

3.2 Analyzing the effect of TM policies

3.2.1 The ratio of occurring interrupt in transaction

The first performance metric of a specific system is influenced by how long a core execute transactions. If the execution time of transaction is long, the probability of occurring interrupt in transaction is increased.

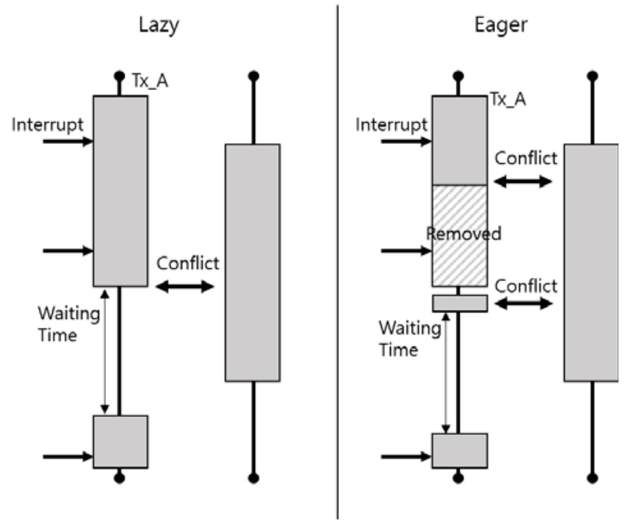

Figure 2 Effect of conflict detection to the first metric

From the perspective, the lazy conflict detection method have more disadvantages than the eager method. The features of the eager is that it checks whether there is a conflict at every memory operations in transaction so that aborting is happened earlier than the lazy method. In figure 2, the length of CPU time that execute transactions in the eager is shorter than the lazy. So the number of interrupt in the lazy case is higher than in the eager case.

The efficiency of the contention management policies also can be analyzed by using the same perspective. The proportion of the time for executing transaction to whole CPU time is influenced by what contention management policy is used in TM, especially what back off policy is employed in each policy. The contention management policy that use fixed back off execute more transactions in certain time than other policies that use exponentially increased back off. Therefore the polite and polka policy that use the back off time increased exponentially have advantages to the first metric.

3.2.2 The interrupt response time of occurring interrupt

Average length of transactions affects to the second performance metric since interrupt happened in a transaction is pended in the proposed system. The average length of transactions of the lazy is longer than that of the eager methods. Contention management policy does not affect interrupt response time. Even if the proposed system selects any contention management policy, the length of transactions is not changed.

3.2.3 The number of empty cores

Factors that decide the value of the third metric are similar to the factors of the first metric. How many CPU times are used to execute transactions is important factor to this metric. Because the smaller the amount of CPU time that execute transactions or interrupt handler is, the higher the probability of existence empty core is. As section 3.2.1 explained, the eager conflict detection method, polite and polka contention management policies that use exponentially increased back off time have advantages.

4 Implementation

To evaluate tendencies of transactional memory policies in the proposed interrupt handling system and how many interrupts can be improved by the proposed interrupt handler, we make an ESL (Electronic system level) platform that support transactional memory concept. The proposed system's target is an embedded system therefore the evaluation platform uses four ARM Cortex-A9 that is provided by OVP (Open virtual platform) [8]. This core model uses the transaction level modeling and doesn't support pipeline and cache system so that the model assumes execution time of all instructions is just one-cycle.

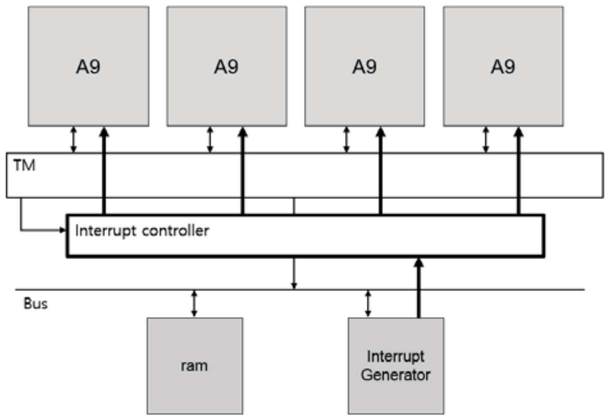

Figure 3 Evaluation platform

Except the cores, other platform components that are transactional memory controller, interrupt controller and ram are implemented by using SystemC and OSCI TLM-2.0 [9]. The transactional memory controller can change internal mechanism among contention management policies and conflict detection methods. Supported contention management policies are the polite, karma, timestamp and polka. The conflict detection is lazy or eager. The controller only supports the lazy version management. So, the controller can supports eight types of transactional memory policy.

There are two types of workload for evaluating the TM system. One is an interrupt workload. In this paper, we implement an interrupt generator module that makes randomized interrupts that are sent to each cores in figure 3. Another is transaction program that is executed on each cores. In this paper, we use the counting benchmark [2] and the number of transactions that is needed to complete the benchmark is set to $4x10^5$. Additionally, to check effects of size of transaction, we simulate the benchmark as we change the size to 16, 32, 52 or 92 bytes.

5 Simulation Results

Figure 4 shows the proportion of pended interrupt that occurs at transaction time to normal interrupt that is handled instantly. The size of transaction used to simulations is set to 32 bytes. The average performance of eager is 39.7% and that of lazy is 19.8%. As we explained in section 3.2.1, the performance of eager method is better than that of lazy in every contention management policies. Between the contention management policies, the polite is better than the others because that employs exponential back off. The polka policy also use same back off mechanism of the polite but the performance of the polka is lower than that of the polite. The reason of the phenomenon is that the polka policy uses the priority concept in karma policy, so the probability of increasing waiting time is low.

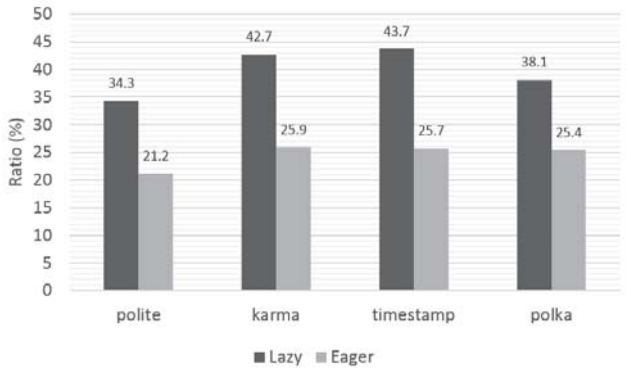

Figure 4 Proportion of interrupt in transaction time

Table 1 shows length of response time that is needed to handle an interrupt that occurs at transaction time. The unit of length is the number of instructions. In almost contention management cases and all transaction size, the eager methods is better than the lazy. There are not difference between contention management policies except the polite-eager case. The reason of polite-eager case is this uses exponential back off. If there are many conflict between transactions in the polite, the average of back off time is increased continuously. The longer the average of back off becomes, the lower probability of conflict is. As a result, the eager is being similar to the lazy. The polka has also same tendency but that is not critical as the polite because the former uses priority concept.

		16	32	52	92
lazy	polite	21.9	50.0	85.0	150.0
	karma	21.1	47.9	83.5	152.0
	timestamp	21.0	47.5	84.5	150.0
	polka	21.4	44.3	77.5	143.8
eager	polite	20.6	44.0	77.0	150.0
	karma	20.7	35.0	57.6	95.0
	timestamp	20.5	36.0	56.0	95.0
	polka	20.2	37.4	62.4	108.3

Table 1 Average response time of interrupt in transaction time

Figure 5 shows how many interrupts can be sent to empty core when the interrupt occurs at transaction time. The y-axis of figure 5 is the percentage of existence of empty core. The size of transaction used to simulations is set to 32 bytes. In lazy system, 72% of pended interrupts can be sent to other empty core. In eager, 97% of interrupt handler for interrupt occurs at transaction time can be improved. In addition, the polite and polka is better than the karma and timestamp.

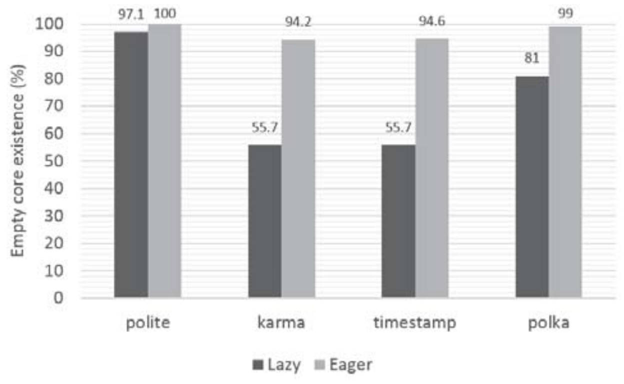

Figure 5 Existence probability of empty cores

If the size of transaction is increased, CPU time for executing transaction is also increased. As a result, the probability of existence is also decreased. Figure 6 shows the result of simulations that use various size of transaction and the lazy conflict detection. Karma, timestamp and polka has same tendency except the polite policy. The polite policy has a unique tendency that the bigger the size of transaction become, the higher the probability of existence is. The phenomenon is explained by using the same reason used to why the first metric of polite is higher.

According to the simulation results, the proposed interrupt controller can improve the performance of interrupt handling. But there are some performance gap in what contention management and conflict detection method are selected to TM system. Between conflict detection methods, the eager method for conflict detection has many advantages about the proposed system. Every contention management has better performance when that is combined with eager.

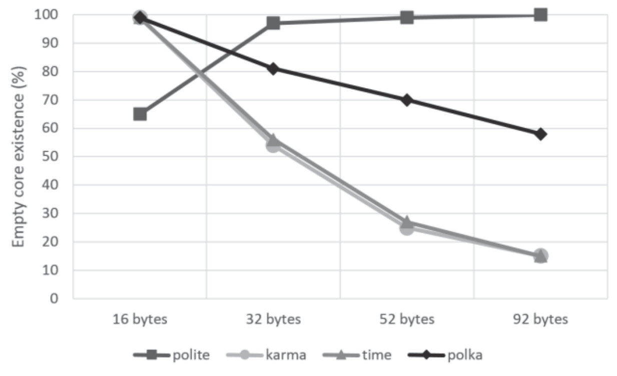

Figure 6 Probability of existence tendency in various transaction size

Among contention management policies, the polite and polka have advantages because those employ exponential back off mechanism. In addition, required hardware resource to implement the polite is smaller than that of the polka because the polite does not use any priority concept so that the polite is proper contention management to the proposed system. The effect of the size of transactions also shows the polite is more appropriate than the polka. But, the proposed result is only evaluated about the interrupt handling performance. So, other types of performance metric should be considered such as elapsed time, abort rate and so on.

6 Conclusion

In this paper, we proposed interrupt handling mechanism in TM for embedded system. The proposed mechanism uses interrupt controller that has transaction information about each cores and distributes interrupts to empty core by using the information. Then we present the performance metrics to evaluate proper contention management and conflict detection method for the proposed system. Also we explain the features of each policies that affect to the proposed performance metrics. Then we implement the ESL platform that has Cortex-A9 quad-core, transactional memory controller and interrupt controller and evaluate the effect of each policies.

The simulation results present that 72% of pended interrupts can be improved at lazy on average and 99% of pended interrupts also can be improved at eager at the size of transaction is 32 bytes. But improvement amounts are affected by the size of transaction and what policies is used by the transactional memory controller. By using the simulation results, we conclude that the eager has advantages to handling interrupts. In addition, the exponentially increased back off time has advantages so that the polite and polka contention management policies are proper to the proposed transactional memory system to handle interrupts effectively. Between them, the polite is better because the implementation cost is lower and the performance tendency about the size of transactions is better.

7 References

[1] Harris, T. Cristal, A., Unsal, O. S., Ayguade, E. Gagliardi, F., Smith, B. and Valero, M. "Transactional memory : An overview"; IEEE Micro, Vol. 27., Issue 3., 8-29, 2007.

[2] Herlihy, M. and J. E. B. Moss. "Transactional memory : Architectural support for lock-free data structures" ; ISCA 93 Proceedings of the 20th annual international symposium on Computer Architecture, 289-300, May 1993.

[3] Moss, J. Eliot B., and Antony L. Hosking. "Nested transactional memory : model and architecture sketches" ; Science of Computer Programming, Vol. 63., Issue 2., 186-201, Dec 2006.

[4] Rajwar, Ravi, Maurice Herlihy, and Konrad Lai. "Virtualizing transactional memory"; ISCA 05 Proceedings of the 32nd International Symposium on Computer Architecture, 494-505, Jun 2005.

[5] Ramadan, H. E., Rossbach, C. J., Porter, D. E., Hofmann, O. S., Bhandari, A., and Witchel, E. "MetaTM/TxLinux : tranasctional memory for operating system"; ACM SIGARCH Computer Architecture News, Vol. 35., Issue 2., 92-103, May 2007.

[6] Scherer III, W. N. and Scott, M. L. "Advanced contention management for dynamic software transactional memory"; Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing., 240-248, 2005.

[7] Rachid Guerraoui and Paolo Romano. "Transactional Memory. Foundations, Algorithms, Tools and Applications". Springer International Publishing, 2015.

[8] Imperas Inc. Open Virtual Platform World. http://www.ovpworld.org/.

[9] Initiative, Open SystemC. "OSCI TLM-2.0 language reference manual." Version JA32, http://www.systemc.org, 2009.