Computing and Informatics, Vol. 31, 2012, 693–709

A NEW CONCURRENT CHECKPOINT MECHANISM FOR EMBEDED MULTI-CORE SYSTEMS

Jianwei Liao

College of Computer and Information Science
Southwest University of China
e-mail: liaojw@swu.edu.cn

Communicated by Bogdan Wiszniewski

Abstract. This paper presents a new transparent, incremental, concurrent checkpoint mechanism for embedded multi-core systems. It allows the checkpointed process (also called checkpointee) to continue running without stopping while checkpoints are set to a large extent. Through tracing TLB misses to block the accesses to target memory pages first time while dumping memory pages (the most time-consuming step when setting a checkpoint). At that time, a kernel thread, called checkpointer, copies the memory access target pages to the designated memory buffer for constructing a consistent state of the checkpointee, and then resumes the memory accesses. From the experimental results, in contrast to a traditional concurrent checkpoint system, the proposed mechanism reduces the downtime of the checkpointed process by more than 10.1 %. Moreover, the incremental checkpointing functionality has been implemented in this new concurrent checkpoint mechanism as well. Compared with full checkpointing, incremental checkpointing can reduce the checkpoint time more than 95.5 % and 89.2 % while the benchmark is the matrix multiplication at the checkpoint intervals of 10 seconds and 20 seconds, respectively.

Keywords: Concurrent checkpoint, reduced downtime, incremental checkpoint, embedded multi-core systems

Mathematics Subject Classification 2010: 68N25: Operating system, 68M25: Reliability, testing and fault tolerance

1 INTRODUCTION

The industry trends of shrinking device geometries, lower voltages and higher frequencies in modern processors and devices are expected to increase the rate of transient hardware faults. Despite the number of those faults is not high as that of software faults, they may collapse the operating system and make the whole system go to crash with very high probability [1, 2]. The operating systems should be reliable to recover the critical applications from such kinds of system crashes by resorting to various fault-tolerant techniques.

Embedded systems are always used in the long-time running cases; for the purpose of providing the high availability for some critical applications, checkpoint/restart mechanism is always used for saving the running state of the target process. Before saving the state of the checkpointed process, also called checkpointee, the conventional checkpoint/restart systems have to stop the checkpointed process for getting its consistent state. In other words, the checkpointee cannot keep running or providing service while setting the checkpoint, and that period is referred to as downtime in this paper.

However, a major part of the applications in the embedded systems are the interactive or real-time applications, which need rigid timing restrictions, such as a finish time and a response time; the downtime required while using traditional checkpoint mechanisms to set a checkpoint for them is always too long to be accepted. We have proposed a new concurrent checkpoint mechanism for real-time and interactive processes in our previous work [3], which employs tracking TLB misses to block the memory accesses until the target pages are copied to the designated memory buffer during setting the checkpoint. It allows the execution of the checkpointee to overlap with the dumping of memory address space without operating on page table. Moreover, it employs two buffers to store the whole address space and the original copies of the accessed pages, then it constructs the image file of the checkpointee by using the contents in these two buffers.

In order to reduce the checkpoint overhead and improve the usage of memory, based on our previous work (which uses two memory buffers and only supports full checkpointing [3]), this paper will propose and implement an improved Transparent, Incremental, Concurrent checkpoint mechanism with a small memory buffer called TIC-CKPT. While TIC-CKPT sets the checkpoints for the checkpointed process, the checkpointee keeps running until the occurrence of TLB miss triggered by the first access to the target page, then the checkpointer kernel thread first copies the memory access target page to the designated memory buffer, and then unblocks checkpointee's memory access request. Before dumping memory pages to the nonvolatile storage, the checkpointer checks whether those pages are in the designated buffer or not. If they are in the buffer, the original pages in the designated buffer will be used to construct a consistent state of the checkpointed process; otherwise, dumping them to the nonvolatile storage directly. Moreover, for the purpose of reducing the checkpoint time, TIC-CKPT supports incremental checkpointing, which means only the dirty pages after the previous checkpoint are saved to the nonvolatile

storage. As a result, the checkpoint time can be reduced to a great extent. Besides, TIC-CKPT only needs a small memory buffer to store the original copies of the accessed target pages during setting the checkpoint; it is suitable for memory-limited embedded systems to build fault-tolerant operating systems.

This paper focuses on the design and implementation of TIC-CKPT, which brings about a little limited impact on the runtime overhead of the target applications while setting full checkpoints and incremental checkpoints. The paper is organized as follows: Section 2 introduces the background knowledge and related work. The design and implementation of TIC-CKPT are described in Section 3. Section 4 presents the experimental results in evaluating the performance of TIC-CKPT. Finally, we present concluding remarks and the directions of future work.

2 BACKGROUND AND RELATED WORK

2.1 Traditional Checkpointing

Many of system-level checkpoint/restart systems have been implemented. BLCR [4] is a typical checkpoint/restart module for the Linux kernel developed and maintained by Berkeley Lab (USA); it supports x86, ARM and PPC systems running Linux 2.6.x kernels. Kernel-based Checkpoint/Restart System [5] is an active project issued by Oren Laadan; it is a kernel-based checkpoint/restart system for the Linux kernel. In fact, most of the traditional checkpoint systems [6, 7] including those mentioned above, need to stop the checkpointed process to ensure the consistent state of the checkpointee during setting the checkpoints.

2.2 Checkpointing Optimization

For the purpose of satisfying the strict timing requirements of real-time or interactive processes while making the checkpoints for them, several checkpoint optimization techniques have been proposed to decrease the checkpoint time. In the traditional checkpoint mechanisms, reducing the checkpoint time means the downtime of the checkpointed process can be reduced as well. Decreasing the content that needed to be saved to the nonvolatile storage is the main direction to reduce the checkpoint time, such as diskless checkpointing [8]. Moreover, incremental checkpointing is a well-known technique to reduce the checkpoint time for some long-time running processes which need multiple checkpoints during their lifetime. For example, space-efficient page-level incremental checkpointing [9] and other incremental checkpointing methods [10, 11] have been proposed successively. The core idea of the incremental checkpoint mechanism is to save the modified pages (i.e. dirty pages) in address space of the checkpointed process from the previous checkpoint. Compared with the number of all pages in the checkpointee's address space, the number of the modified pages is always smaller. Consequently, the downtime of the checkpointee while setting a checkpoint is also decreased.

2.3 Concurrent Checkpointing

However, the optimized techniques mentioned in Section 2.2 are solutions to the symptoms but not to the causes; they cannot reduce the downtime of the checkpointee fundamentally. As shown in our previous work [3], dumping memory address is responsible for the major part of the checkpoint time. If the execution of the checkpointee can overlap with the dumping memory address space, the downtime of the checkpointee due to setting the checkpoint can be decreased to a great extent in theory; this is motivation of the concurrent checkpoint mechanisms.

As a matter of fact, the concept of concurrent checkpointing in this paper is not a new theory. K. Li and J. S. Plank [12] proposed a low-latency, concurrent checkpoint system for parallel programs called the Concurrent Low-Latency (CLL) checkpoint system, which aims at overlapping the execution of the checkpointee with the dumping of memory address space, distinct from traditional checkpoint systems; the CLL checkpoint system works as follows:

1. Stop the checkpointed process.
2. Save the values of registers, thread information et cetera to the nonvolatile storage; although they did not mention that TLB entries should to be flushed, this operation should be done before the 3rd step.
3. Turn off all the access right bits in checkpointee's page table; then resume the checkpointee.
4. Copy the memory address space to a designated memory buffer concurrently with a kernel thread (called Copier). After copying a page, turn on the corresponding access bit. During the Copier copies the memory pages, the modified page fault handler blocks the write accesses, and invokes the Copier to copy the original target page to memory buffer first, then switch on the access right bit of the corresponding page table entry.
5. After the Copier saves the whole address space, another kernel thread called Writer stores the data in the memory buffer to the nonvolatile storage to form an image file which contains the original copies of the write target pages.

CLL checkpoint system works quite like copy-on-write technique; it allows setting checkpoints concurrently with the execution of the checkpointee, interrupts the checkpointee only for small, fixed amounts of time, and is transparent to the checkpointee. The original pages in the memory buffer will be used to construct the consistent state of the checkpointed process. Needless to say, CLL is quite suitable to set the checkpoints for real-time and interactive processes in embedded systems. However, the CLL checkpoint system needs too many extra memory accesses for setting and restoring all access right bits in the page table, which weaken the benefit brought by this kind of concurrent checkpointing directly. In addition, CLL concurrent checkpoint mechanism does not support incremental checkpointing, which is suitable for setting multiple checkpoints for the long-time running applications with quite short checkpoint time.

3 DESIGN AND IMPLEMENTATION OF TIC-CKPT

In this section, we present the design and implementation of TIC-CKPT, which does not require extra page table operations and supports the incremental checkpointing. Because the algorithms of the full checkpointing and incremental checkpointing are quite different, their algorithms are presented in Sections 3.1 and 3.2 separately.

3.1 TIC-CKPT Algorithm

The algorithm of the full checkpoint mechanism is shown in Figure 1, where checkpointer stands for the kernel thread that sets the checkpoint and works in privileged mode; buffer B is a designated memory buffer to save the values of registers, information of the thread et cetera and the copies of memory access target pages during saving address space.

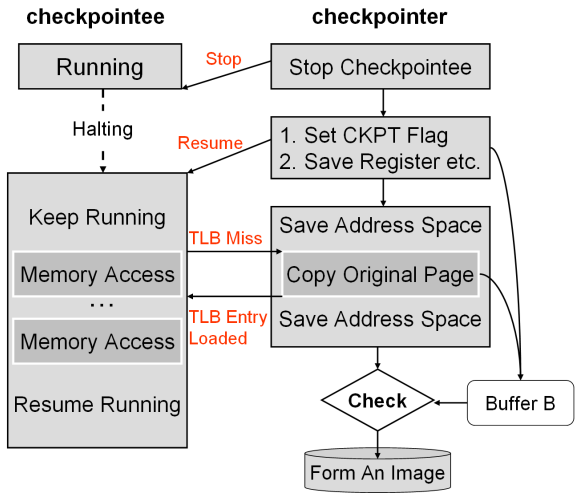

Fig. 1. TIC-CKPT architecture

Unlike the CLL checkpoint mechanism, TIC-CKPT works as follows:

1. Stop the checkpointed process by sending a "stop" signal.
2. Copy the values of registers and thread information to the designated buffer B rather than nonvolatile storage for decreasing the stop time of the checkpointed process.
3. Set the checkpoint flag to indicate that a checkpoint is being set now, and invalidate TLB entries.
4. Resume the checkpointee by sending a "continue" signal.

5. Save memory address space to the nonvolatile storage; meanwhile, if there is an access request to the target memory page first time during saving of the memory address space, since the TLB handler was modified to support concurrent checkpointing, that access request will be blocked until the original target page has been copied to buffer B; finally, the memory access proceeds as usual.

   While copying the address space to the nonvolatile storage, the checkpointer scans the virtual memory areas of the checkpointee's address space, gets the virtual address of every page, and checks whether the virtual address of the page is in buffer B or not. If not, then this page is saved to the nonvolatile storage directly. If the virtual address is in buffer B, that means there were memory accesses to this page after the starting of dumping memory address space, then the copy of the original page in buffer B will be moved to the nonvolatile storage. Therefore, only a quite small memory buffer is employed to store a list and the copies of the accessed pages rather than a big memory buffer for saving a copy of the whole address space; this property enables the TIC-CKPT to be applied in memory-limited embedded systems.
6. Clear the checkpoint flag after the checkpointee's address space is saved to the nonvolatile storage to represent the checkpointing is completed; an image file containing a consistent state of the checkpointee is constructed and saved into the nonvolatile storage.

Because both write and read requests to memory pages can be captured by tracing TLB misses [6, 13], before copying memory address space of the checkpointee, TLB should be invalidated (i.e. flushed), and as a result, every write or read to a page for the first time will cause a TLB miss. If the checkpoint flag is set, then the read or write request cannot be fulfilled until the original target page is copied to buffer B. It is different from the traditional concurrent checkpoint mechanism, there are no extra memory accesses brought by the operations on the page table.

In TIC-CKPT, saving memory address space is being processed concurrently with the execution of the checkpointed process to a great extent. Thus, it is necessary to block the checkpointee when copying the original access target page to buffer B before the first access request to that page. According to the locality of reference, compared with the number of pages in the whole address space, the number of original copies of the accessed target pages is much smaller.

Compared with copying the pages of the address space to memory buffer temporarily, dumping the pages to nonvolatile storage directly takes much longer time; in addition, before saving every page, TIC-CKPT has to check whether the page in memory address space is in buffer B or not (in Section 4.2, we will see TIC-CKPT needs much more time to set a checkpoint than non-concurrent checkpoint mechanism does); thus in order to manage buffer B much more effectively, PageList is introduced to reflect the pages in address space which were accessed after the start of saving the address space. Each node of the PageList contains the virtual address of a page and the corresponding page structure pointer as shown in Figure 2.

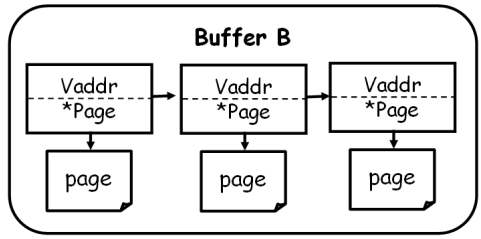

Fig. 2. PageList data structure

If there is a write operation to a page which has been saving to the nonvolatile storage, then the page saved to disk might be dirty. In order to prevent such exceptions, a Read-Copy Update lock is used while saving a page to nonvolatile storage.

3.2 Incremental Checkpointing

As mentioned in Section 2.2, incremental checkpointing is a widely used technique to reduce the checkpoint time. It saves the dirty pages after the previous checkpoint; there are two kinds of methods to keep track of the dirty pages. The first mechanism is using dirty bit. After setting a checkpoint, all the writable pages are cleaned as non-dirty. While the process writes the pages, the operating system will set the dirty bits in the corresponding page table entries. In other words, we can discern which pages are modified since the previous checkpoint by traversing the page table, then save the modified pages to the nonvolatile storage. The other mechanism is called bookkeeping [6]; it sets all writable pages as read-only after a checkpoint. There must be a page fault exception when the page has been written. Then, the modified page fault handler inserts the address of corresponding page to a designated data structure, such as a list. At last, the incremental checkpoint mechanism just needs to save the pages whose addresses are in the designated data structure.

Both mechanisms mentioned above require operating on the page table. In TIC-CKPT, incremental checkpointing is also supported; it provides a mechanism such as bookkeeping to track the dirty pages but without any extra operations on the page table. TIC-CKPT tracks the dirty pages by resorting to TLB modification misses (i.e. write violations). As mentioned before, both write and read accesses cause the TLB misses; in order to distinguish them, and track the write target pages only, the modified TLB handler clears the read/write bit of the page table entry before loading it into TLB for the first time. Therefore, a write access to that 'read-only' page leads to a page fault exception; then the virtual address of that page will be inserted into the designated data structure called Address List shown in Figure 3. After that, page fault handler works as normal and calls TLB handler to load the corresponding page table entry again with the original read/write bit. While setting incremental checkpoint, it iterates the Address List to obtain the virtual addresses of

the dirty pages after the previous checkpoint, then saves the corresponding physical pages to the nonvolatile storage. At last, an incremental checkpointing image file is formed which only contains a small part of pages in checkpointee's address space.

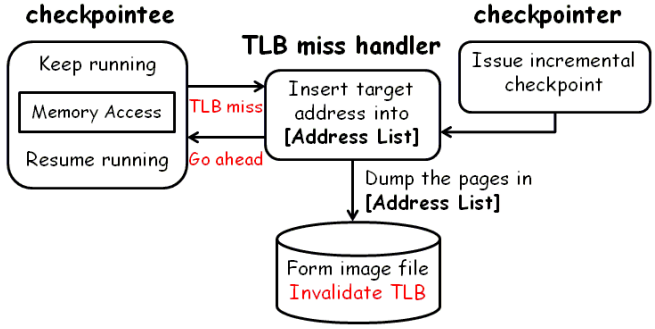

Fig. 3. Incremental checkpointing algorithm

For all incremental checkpoints, the checkpointer should save the snapshot of the values of registers and the thread information. Since we have discussed such operations in the last subsection, there is no illustration thereof in Figure 3. Quite unlike full checkpointing workflow discussed in Section 3.1, incremental checkpointing needs to invalidate all TLB entries before the ending of setting a checkpoint. In other words, invalidating TLB entries should be the last step to set an incremental checkpoint, and the motivation is to support tracking dirty pages for the next incremental checkpoint.

3.3 Implementation

TIC-CKPT is implemented as a Linux module in Linux kernel 2.6.28 with 16 new source files, more than 5 000 LOCs. The target architecture is SH4 platform [14]. There are 200 lines of source code modification in the TLB handler (the file is named tlb-sh4.c). In addition, there is an eight line patch that involves two files of the Linux kernel. Though we did not discuss the design and implementation of the restart mechanism in this paper, this functionality has been also implemented to verify the checkpoint functionality.

Li's proposed CLL checkpoint system is a typical concurrent checkpoint system, for the comparison experiments, we have implemented this checkpoint system in the Linux kernel for the SH4 architecture, but we need to declare this again although it has been mentioned in Section 2.3; the experimental Linux version of CLL concurrent checkpoint system assumes all memory write requests are legal. We admit that we can use two page tables to ensure the illegal write request cannot write the read-only memory page; however, not only the degrade of concurrency due to much more comparison should be processed, but also numerous of modifications in Linux kernel internals.

Moreover, since there are no traditional system-level checkpoint implementations that target on the SH4 architecture, we have also implemented a traditional checkpoint system for the SH4 architecture, which we called the non-concurrent checkpoint system.

In Section 3.1, we used the checkpoint flag to show whether the process is being checkpointed or not. In order to reduce the overhead of reading value of checkpoint flag from a global variable, we have defined a checkpoint bit to indicate the checkpoint is being set or not, employing an unused bit in the memory management control register on the SH4 architecture. Please note that after the checkpointing, we did not validate the TLB because on the SH4 architecture, invalidating TLB just means an operation that flushes all TLB entries rather than disabling the TLB. Maybe calling it flushing all TLB entries is much more proper, but in the programmer manual [14], this operation is named invalidating TLB, thus we use this term in this paper.

4 EXPERIMENTS AND EVALUATION

4.1 Experimental Platform and Benchmarks

In order to evaluate the performance of our proposed TIC-CKPT, we used a multiple core SH4 board as our experimental platform, called SH-4A [15]. It is a 32-bit RISC microprocessor that is upward compatible with the SH-1, SH-2, SH-3, and SH-4 microcomputers at instruction set code level. SH-4A has a quad-core CPU, each core with maximum operating frequency of 600 MHz, 128 MB of memory, 4 instruction TLB entries, 64 unified TLB entries with full-associative configuration and 1 000 BaseTx Ethernet. Network file system (4×160 GB, 7 200 rpm SATA hard disks equipped on the server side) has been adopted as persistent storage to save the root file system and the checkpointed image.

Before presenting the experimental results in this section, we will introduce the benchmarks used in evaluation experiments: Matrix multiplication (MAT), the matrix size, such as 256, means there are 256×256 elements in this matrix, the type of the element is double precision floating-point format; Pattern Matching (PM), means finding a place where one string exists within a string text file, using the KMP algorithm; Bubble Sorts (SORT), a typical stable sort algorithm; Fast Fourier Transforms (FFT), an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse; The Joseph Problem Algorithm (JPA), a well-known recursive algorithm. Moreover, a light-weight real-time benchmark called rt-benchmark [16], which runs simple real-time task periodically, is adopted to measure the longest stopped times by using different checkpoint mechanisms to set one checkpoint.

4.2 Overhead: Checkpoint Time

Figure 4 indicates the comparison of checkpoint times by using three checkpoint systems mentioned in Section 3.3 to set one checkpoint. We can see that checkpoint

times by using both TIC-CKPT and CLL checkpoint systems are almost 20 % longer than when using non-concurrent checkpoint system.

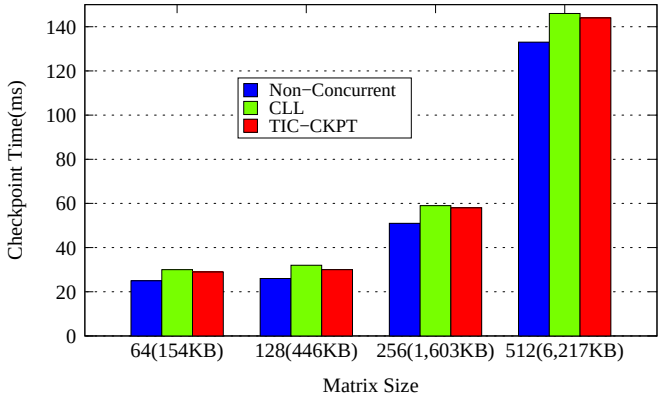

Fig. 4. Checkpoint time

Both TIC-CKPT and CLL checkpoint mechanisms check whether the pages are in the designated buffer or not before saving a page. Moreover, other operations such as invalidating TLB entries take up a part of the checkpoint time. Fortunately, the focus of our work is to allow the setting of the checkpoint and running of the checkpointed process to take place concurrently; therefore, although the checkpoint times introduced by TIC-CKPT and CLL are longer than in non-concurrent checkpoint system, the absolute stop time of the checkpointed process is much less. Such information will be presented in Section 4.3.

In addition, since the CLL checkpoint system sets all access right bits of the page table entries to be read-only before dumping memory and restores them after saving a page to non-volatile storage, it takes around 2 % more time to set a checkpoint than TIC-CKPT.

4.3 Reduced Downtime

In order to illustrate TIC-CKPT performs better than the CLL concurrent checkpoint system in reducing the downtime of the checkpointed processes, the metric called Percentage of Reduced Downtime (PRDT) is used to evaluate the reduced downtime while using various checkpoint systems to set the checkpoints. Let us define the downtime while using non-concurrent checkpointing to set a checkpoint as DT_{non}, and the downtime while using concurrent checkpointing to set a checkpoint as DT_{con}, then the percentage of reduced time is defined as:

$$PRDT = (DT_{non} - DT_{con})/DT_{non}. \tag{1}$$

In comparison with the non-concurrent checkpoint system, TIC-CKPT reduces 47.37–89.82 % of the downtime of the checkpointed process when the benchmark

is matrix multiplication with different sizes. In addition, compared with the CLL checkpoint system, TIC-CKPT reduces the downtime of the checkpointed processes by more than 10.1 % on our experimental benchmarks. Detailed results are shown in Figure 5.

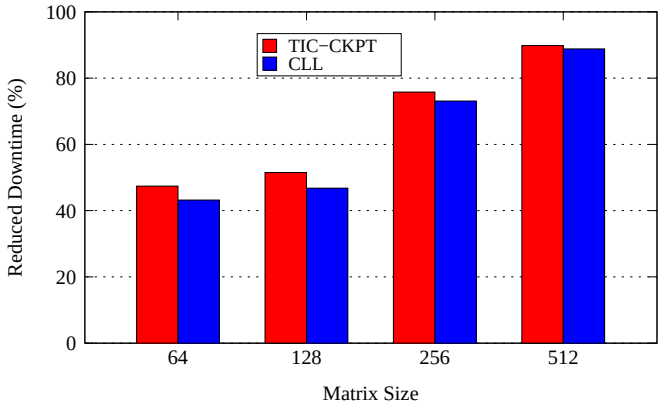

Fig. 5. Percentage of reduced downtime of MAT processes

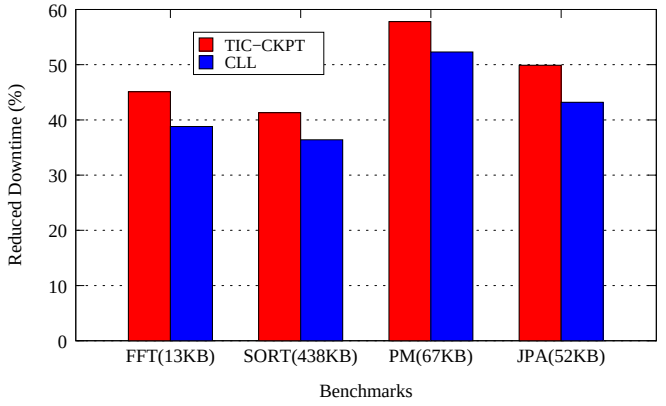

Fig. 6. Percentage of reduced downtime of compute-intensive processes

Some compute-intensive applications introduced in Section 4.1 are used to measure the performance of this concurrent checkpoint system. In fact, these applications keep the principle of locality in a certain degree. In Figure 6, the downtime of checkpiontees is reduced by more than 35.29 % and 41.18 % while using the CLL checkpoint system and the TIC-CKPT, respectively. Compared with the CLL checkpoint system, this new concurrent checkpoint system can decrease the downtime by more than 12.24 %. It is easy to see from Figures 5 and 6 that in contrast to the CLL

checkpoint system, the proposed TIC-CKPT has much more concurrency and can reduce much more downtime. This property is quite suitable for real-time and interactive processes; although the checkpoint time may become longer, the downtime of the checkpointee is reduced far more than that brought by the non-concurrent checkpoint systems.

Checkpoint System	Downtime [msec]	Execution Time [msec]	Real Runtime [msec]
Non-Concurrent	46.3	7.3	53.6
CLL	10.6	7.3	17.9
TIC-CKPT	9.7	7.3	17

Table 1. Setting a checkpoint for real-time benchmark

In addition, a real-time benchmark called rt-benchmark has been chosen to show TIC-CKPT can get much concurrency and meet the time constraint better than others. The downtime while setting a checkpoint for it with three checkpoint systems are reported in Table 1, where Execution Time means the average interval between the finish time and the expected scheduled time without any checkpoints; Real Runtime means the average interval between the finish time and the expected scheduled time while one checkpoint was set. This table shows that TIC-CKPT introduces the shortest downtime, that means with the proper deadline configuration for rt-benchmark, for instance, 17.5 ms in our experiments; then the task can complete before deadline even though a checkpoint has been set by using TIC-CKPT with quite high probability[1]. On the contrary, while using CLL and traditional checkpoint mechanisms to set a checkpoint, the task misses the deadline with high probability. In addition, if we set 19 ms as the deadline, based on our experimental data (the maximum of Execution Time we got is 8.8 ms), the task can complete before the deadline while using TIC-CKPT to set one checkpoint; however, while using CLL to set a checkpoint, the task might miss the deadline.

4.4 Incremental Checkpointing

We used the number of copied pages as performance parameters to evaluate the incremental checkpointing in TIC-CKPT. It is very clear, while the number of copied pages is becoming smaller, the time needed for setting a checkpoint is much shorter. The benchmark we used is matrix multiplication (the matrix size is 1 024, the average execution time on our experimental platform is more than 300 seconds since the experimental board SH-4A cannot support the double precision directly, all computation is emulated by software), time intervals for each checkpointing are 10 seconds and 20 seconds. The numbers of the saved pages while setting full checkpoints and incremental checkpoints are shown in Figures 7 and 8.

[1] Depends on when the scheduler chooses the task to run.

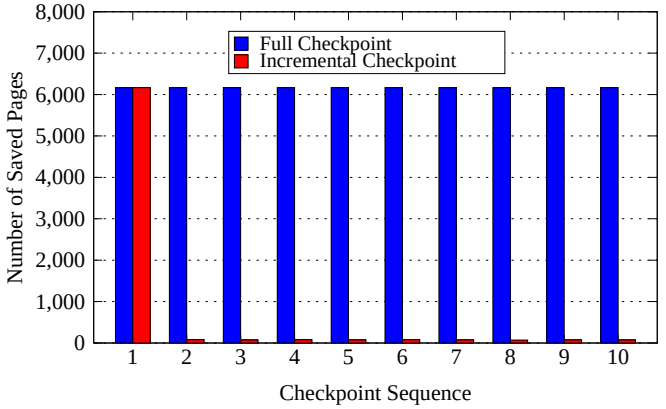

Fig. 7. Numbers of copied pages (10 seconds)

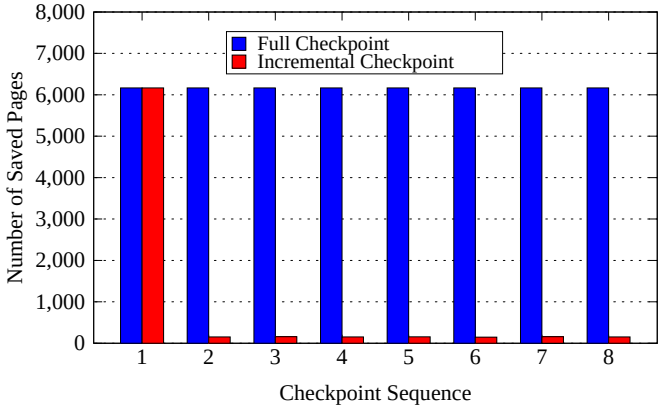

Fig. 8. Numbers of copied pages (20 seconds)

Except the first checkpoint, incremental checkpointing reduces the number of copied pages by more than 98.7 % and 97.4 % while the intervals between two checkpoints are 10 seconds and 20 seconds, respectively. As a matter of fact, after the initialization of two input matrices, matrix multiplication overwrites only the memory area belonging to the output matrix. Because the number of modified pages after the previous checkpoint is only a very small part of the whole address space, in contrast to the full checkpointing, the number of pages saved to the nonvolatile storage while using incremental checkpoint mechanism is much smaller than when using the full checkpoint mechanism.

Moreover, we have measured the checkpoint time for setting a full checkpoint and the second incremental checkpoint; the results are shown in Table 2. We can see from the table, compared with full checkpointing, incremental checkpointing

reduces the checkpoint time by more than 89.2 % when the time slice between two incremental checkpoints is 20 seconds, respectively; this is because the number of saved memory pages by using incremental checkpointing is much smaller than the number of saved pages by using full checkpointing.

Checkpoint Interval	Full Checkpoint [msec]	Incremental Checkpoint [msec]
Interval (10 seconds)	621.4	38
Interval (20 seconds)	621.4	67

Table 2. Time for setting a full and an incremental checkpoint for MAT ($1\,024 \times 1\,024$)

The experimental results in this section also show that while the time becomes longer, the number of copied pages is becoming bigger. It is obvious that checkpoint interval is the key parameter to incremental checkpointing; while the interval is bigger than a threshold, incremental checkpointing might not perform better than full checkpointing. Fortunately, for the purpose of saving the latest state of the running process, it is unacceptable that full checkpointing techniques do not make a checkpoint until a major part of memory pages has been modified. In general, the time interval for setting incremental checkpoints could be the same as that for setting full checkpoints.

4.5 Restart Mechanism

All three checkpoint systems employed in the evaluation experiments have the same restart mechanism for restoring the checkpointed process from the full checkpoint images. Therefore, we present the restart time from full checkpoint by using TIC-CKPT only in this section. Besides, the restart time from incremental checkpoints by using TIC-CKPT is presented in this section as well.

Figure 9 shows the restart times by using the full checkpoint and incremental checkpoints. In the figure, X axis represents the time point for setting checkpoints (for example, while TIC-CKPT makes an incremental checkpoint per 10 seconds, '50' means there are 4 incremental checkpoint image files and 1 full checkpoint image file) then the restarting should reload the latest state from all 5 checkpoint image files. We can conclude from the experimental results that it takes much more time when restarting from several incremental checkpoint image files compared with restarting from one full checkpoint image only, because in the case of restarting with the incremental checkpoints, not only the incremental checkpoint image files, but also the full checkpoint image file should be opened and read by the kernel thread to restore the latest and consistent state of the checkpointed process.

In addition, another problem with the incremental checkpointing is the occupation of disk space for storing the checkpoint image files [17]. Therefore, a full checkpoint after several incremental checkpoints should be employed to release the disk space occupied by incremental checkpoint image files and avoid the unacceptable restart time.

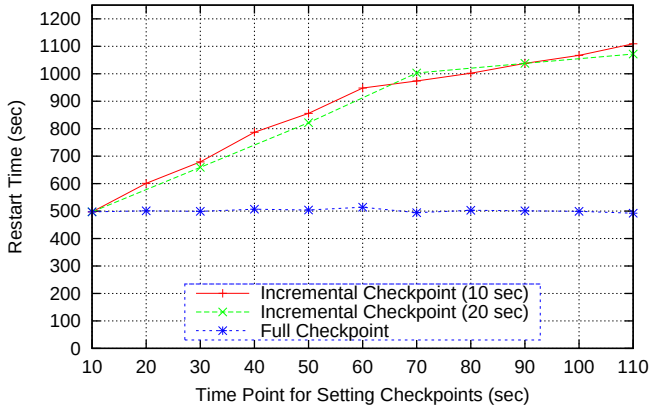

Fig. 9. Restart overhead

5 CONCLUSIONS

A new system-level, transparent, concurrent checkpoint mechanism for multi-core embedded systems called TIC-CKPT has been designed, implemented and evaluated in this paper. This mechanism allows the checkpointed process to keep running while setting the checkpoints for it to a certain degree without any extra operations on the page table and extra hardware. Much more exactly, the most time-consuming step of setting a checkpoint (i.e. dumping address space) is overlapping with the running of the checkpointed process. In addition, in order to reduce the checkpoint time for setting checkpoints for long-time running processes which may need multiple checkpoints during their life time, incremental checkpointing has also been proposed and implemented in TIC-CKPT.

The experimental results show that in contrast to non-concurrent checkpoint, for our selected benchmarks, the downtime of the checkpointed process can be reduced by 50–90 %. In addition, compared with the CLL checkpoint system proposed by Kai Li et al., the downtime can be reduced by around 10.1 %. For this reason, TIC-CKPT is suitable for real-time and interactive processes which have stringent timing requirements, such as a finish time or a response time. Besides, from the results of experiments on incremental checkpointing, except for the first checkpoint for the benchmark of matrix multiplication, incremental checkpointing in TIC-CKPT can reduce the checkpoint time by more than 95.5 % and 89.2 %, while the time slices between two checkpoints are 10 seconds and 20 seconds, respectively. Though restarting from incremental checkpoints takes longer time, we can obtain more benefits from incremental checkpointing because of the many checkpoints during the execution of the applications, but the restarting is rare. In general, setting a full checkpoint after several incremental checkpoints is employed to release the disk space occupied by incremental checkpoint files and avoid the unacceptable restart time.

Although TLB misses and loads are transparent to IA-32 and IA-64 platforms, some other architectures, such as MIPS and SuperH, can employ TIC-CKPT mechanism since the loading of page table entries to TLB is handled by the operating system; Sparc and Power PC have hashed page tables that act as extended TLBs, so every TLB miss causes a fault, which is handled by the operating system. Therefore, TIC-CKPT can also be implemented in the operating systems targeted for these architectures with some minor modifications by tracing the hashed page tables. The current implementation of TIC-CKPT only supports single process applications, it cannot set the checkpoints for the multi-process applications. We need to complete this checkpoint system to support checkpointing the multi-process and multi-thread applications in the near future.

Acknowledgement

This work was partially supported by Scientific Research Fund for Doctor of Southwest University of China (No. SWU112025).

REFERENCES

[1] Borkar, S.: Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. IEEE Micro, Vol. 25, 2005, No. 6, pp. 10–16.

[2] Rashid, L.—Pattabiraman, K.—Gopalakrishnan, S. (Eds.): Towards Understanding the Effects of Intermittent Hardware Faults on Programs. Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSNW '10), Boston, June 2010, pp. 101–106.

[3] Liao, J.—Ishikawa, Y.: A New Concurrent Checkpoint Mechanism for Real-Time and Interactive Processes. In Proceedings of 2010 IEEE 34th Annual Computer Software and Applications Conference (Compsac '10), Seoul (Korea), July 2010, pp. 47–52.

[4] BLCR. Available on: `https://ftg.lbl.gov/projects/CheckpointRestart/`.

[5] Kernel based checkpoint/restart. Available on: `https://www.linux-cr.org/`.

[6] Gioiosa, R.—Petrini, F.: Transparent, Incremental Checkpointing at Kernel Level: A Foundation for Fault Tolerance for Parallel Computers. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05), WA, USA 2005, pp. 1–9.

[7] Wang, C.—Scott, S.-L.: Proactive Process-Level Live Migration in HPC Environments. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC '08), Texas (USA) 2008, pp. 1–12.

[8] Plank, J. S.—Li, K.—Puening, M. A.: Diskless Checkpointing. IEEE Trans. Parallel Distrib. Syst., Vol. 9, 1998, No. 10, pp. 972–986.

[9] Yi, S.—Hong, J.: Adaptive Page-Level Incremental Checkpointing Based on Expected Recovery Time. In: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC '06, pp. 1472–1476.

[10] Naksinehaboon, N.—Scott, S. L.: Reliability-Aware Approach: An Incremental Checkpoint/Restart Model in HPC Environments In: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, CCGRID '08, pp. 783–788.

[11] Mehnert-Spahn, J.–Schoettner, M.: Incremental Checkpointing for Grids. In: Proceedings of the Linux Symposium 2009, Canada, July 2009, pp. 201–208.

[12] Li, K.—Naughton, J. F.—Plank, J. S.: Low-Latency, Concurrent Checkpointing for Parallel Programs. IEEE Trans. Parallel Distrib. Syst., Vol. 5, 1994, No. 8, pp. 874–879.

[13] Li, Y.—Lan, Z.: A Fast Restart Mechanism for Checkpoint/Recovery Protocols in Networked Environments. In: Proceedings of The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '08), Alaska, USA, June 2008, pp. 217–226.

[14] SH-4 CPU Core Architecture. Available on: `http://www.shell-storm.org/papers/files/768.pdf`.

[15] SuperH RISC engine Family. Available on: `http://www.renesas.com/`.

[16] rt-benchmark: A light weight real-time benchmark. Availaible on: `http://nerdvar.com/prog/rt-benchmark.c`.

[17] Sangho, J.—Shin, S.: Space-Efficient Page-Level Incremental Checkpointing. In: Proceedings of the 2005 ACM Symposium on Applied Computing (SAC '05), New Mexico, USA, March 2005 pp. 1558–1562.

Jianwei Liao received his Ph. D. degree in computer science from the University of Tokyo, Japan in 2012. Now he works for the College of Computer and Information Science, Southwest University of China. He published several articles in international peer reviewed journals and IEEE conferences as the first author. His research interests include dependable operating systems and parallel file systems.