International Journal of Control and Automation
Vol.10, No.9 (2017), pp.177-190
http://dx.doi.org/10.14257/ijca.2017.10.9.15

A Dynamic Analysis Tool for Real-time Characteristics Related to Cache Hit-ratio of Real-time Applications*

Hyang Yeon Bae[1], Ok-Kyoon Ha[2†] and Yong-Kee Jun[1]

[1]*Department of Informatics, Gyeongsang National University,*
[2]*Department of Aeronautics & Software Engineering, Kyungwoon University*
baehy06@gmail.com, okha@ikw.ac.kr, jun@gnu.ac.kr

Abstract

It is important to guarantee the reliability of applications for real-time systems such as avionics. Unfortunately, existing profilers that measure the performance of applications cannot provide the information related to the cache hit-ratio of real-time applications. This paper presents a dynamic analysis tool for analyzing real-time characteristics related to the cache hit-ratio in real-time applications. This tool consists of a monitor module, which traces every function to find periodic functions, and an analyzer module, which analyzes the information of the function executions of the application, such as the thread identifier, caller, callee, number of calls, average time of execution, whether the signal callback or not, average periodic, violation of periodicity, and cache read miss rate in real-time applications

Keywords*: Analysis tool, periodic function, profiler, real-time applications*

1. Introduction

Real-time software demands both functional and temporal accuracy. Software requirements for temporal accuracy include requirements for time constraints. Failure to meet these deadline requirements for real-time software can lead to not only to degradation in software quality, but also, in worst-case scenarios, a major loss of life and/or property. It is therefore important to satisfy time constraint requirements [1]. Various solutions have been applied to hardware and software to guarantee real-time performance [2]. However, performance degradation caused by external factors is difficult to prevent and predict as well [3]. One of the external factors that make it difficult to predict performance degradation is the cache hit-ratio change due to memory access by multiple threads of parallel execution. When multiple threads executed in parallel schedule access a wide range of memory, the locality may degrade. When the cache locality is degraded, the cache hit-ratio decreases and the program performance deteriorates at the same time [4].

Furthermore, owing to the characteristics of multiple threads of parallel execution, which make it difficult to predict a time to execute, the cache hit-ratio may be degraded by unexpected timing. This degradation may increase the risk of violating a demand for the software's deadline constraints. Therefore, it is important to analyze problems in advance to improve performance. In order to analyze potential problems and guarantee the quality of the software in advance during the development process of the application, dynamic testing is performed. Dynamic testing is a technique that runs an executable

Received (November 28, 2016), Review Result (February 15, 2017), Accepted (March 10, 2017)
*This paper is a revised and expanded version of a paper entitled "A Dynamic Analysis Tool for Periodic Task in Real-time Applications" presented at ASEA 2016 conference, November 25, 2016, Jeju, Korea.
† Corresponding Author

ISSN: 2005-4297

program to test for problems that can occur during actual execution. This results in a large temporal and spatial overhead, but less false alarms.

Among the existing tools, there are dynamic testing tools such as Vtune [5], Valgrind [6], HPCToolkit [7], and CC-Analyzer [8] that can analyze cache hit-ratios together. These existing tools provide application execution information such as memory leaks, thread contention, call graph, as well as cache hit-ratio. However, these tools do not provide information related to real-time properties. In this paper, we propose a dynamic analysis tool that can find tasks whose cache hit-ratio violates the timing constraints of real-time applications during the execution of the application. The proposed dynamic analysis tool inputs a binary program that can be executed using the PIN framework [9], which is an analysis tool with dynamic instrumentation. It then generates the execution time of the function whose real-time properties have been violated owing to the cache hit-ratio during execution. The analysis tool is subcategorized into a monitor module and an analyzer module. The software analysis tools include a thread number, caller, callee, call count, average execution time, thread and signal callback distinction, average execution cycle, cycle violation frequency, and the rate of cache hit and miss, so that the section where the problem occurs can be easily identified.

2. Background

This chapter explains the general outlines of multi-core and the importance of timing constraint requirements for real-time applications. It also describes experimentally the performance problems that can occur when cache locality deteriorates, and explains the profiler that can analyze cache hit-ratio and profiler problems.

2.1. Real-Time Application

In real-time application, real-time means to respond within a given response time. The basic requirement of the difference between real-time software and general software is the predictability and timeliness of software operation. Predictability is the requirement that the overall performance of the software and the performance and timing of individual tasks must be predictable, while timeliness refers to the requirement to respond within a specified deadline. Since these real-time requirements are difficult to satisfy in a general operating system (OS), an application is developed using a real-time OS [10] with preemptive scheduling.

Real-time applications perform real-time tasks. Real-time tasks are classified as hard, soft, and firm tasks according to the strictness of timing constraints (*i.e.*, deadline constraints). A hard real-time task must be performed within the confines of a very stringent timing constraint, thus a dedicated environment is essential. A hard real-time task that performs sensor data input and operation control of safety-critical systems has very strict deadline constraints. Since a hard real-time task is used for safety-critical operations, such as actuator servos or sensor inputs in aircraft, failure to complete a task within a specified deadline can lead to catastrophic consequences, such as personal injury or property damage.

A soft task, such as keyboard or mouse input/output, monitor output, *etc.*, may fail to meet timing requirements, but the results are not fatal and the computed results are not insignificant. A firm task, such as video, audio, image processing, *etc.*, does not cause a serious problem, although the computed result becomes insignificant if the task is not completed within a given time [11,12]. Real-time applications with hard real-time tasks are required to guarantee real-time performance because they have serious consequences if real-time properties are violated. Real-time applications with soft and/or firm tasks do not cause serious problems. However, since these result in the degradation of quality, it is important to meet deadline constraints of all kinds in real-time applications.

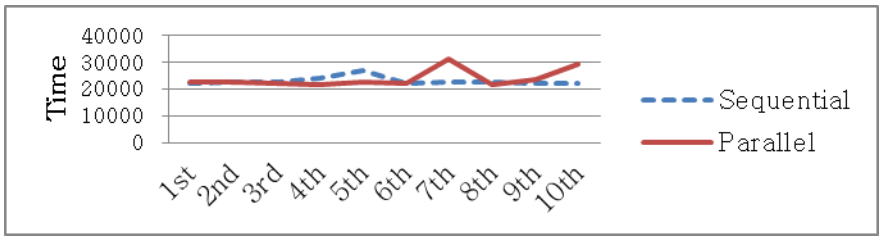

Figure 1. Execution Time Graph of Function that Reads 64 * 64byte Array in Total 4MB Cache

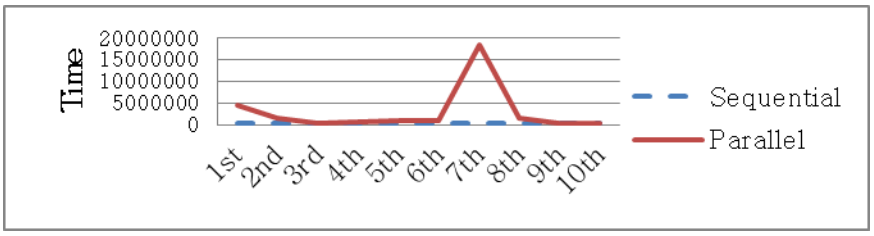

Figure 2. Execution Time Graph of Function that Reads 256 * 256-byte Array in Total 4MB Cache

2.2. Cache Memory

Cache memory is the small amount of SRAM in the CPU, and cache memory is used to reduce the difference between the CPU's fast processing speed and the relatively slow read/write speed of memory. By duplicating frequently read data into the cache memory, it is possible to reduce the time required to access the underlying slower storage memory, thereby enabling high-speed processing.

To improve cache efficiency, locality is important. Locality falls into two categories, temporal and spatial locality. Temporal locality refers to the time interval between re-access to the location of the once accessed memory is close. Spatial locality refers to the access location of the successively read data is stored physically close [13]. Figure 1 and Figure 2 are 4 MB in total. In order to show the cache hit-ratio that varies in accordance with cache locality, and the performance that varies according to the cache hit-ratio in the environments of L1 cache (128 KB) and the L2 cache (512 KB), in Figure 1 and Figure 2, respectively, by using a program "Sequential" with a relatively low cache miss rate and a synthetic program "Parallel" with a relatively high cache miss rate, an experiment was carried out to read 4 KB and 64 KB shared memory, respectively.

The "Sequential" is a synthesis program that induces sequential reading of one shared memory by synchronizing multiple threads of parallel execution. "Parallel" is a program that allows multiple threads of parallel execution to read a shared memory

simultaneously without synchronization. When "Parallel" reads a large amount of data, it is likely to significantly reduce the cache hit-ratio by changing the contents of the shared cache by other cores.

The experimental results shown in the graphs demonstrate that the execution result of "Parallel" was relatively uneven in the case of reading 4 KB of memory. When reading 64 bytes of memory, the execution time was not just uneven but also had significantly longer execution time. Especially in the seventh experiment, when the cache hit-ratio greatly decreased due to changes in shared cache content made by other cores performing reading operations, a significant execution delay occurred because it was also influenced by background processes. The experiment also showed that memory accesses of multiple threads of parallel execution lowered the cache hit-ratio. As a result, an increase in the fluctuation range of execution time may lead to unexpected performance degradation

2.3. Profiler

In software development, software writers need tools to test and analyze whether their programs meet given requirements and to identify code errors. A profiler (*i.e.*, code profiler) is a kind of tool used to facilitate testing. A profiler's function is to measure the temporal and spatial performance of the software and provide program execution information so that the measured problems can be solved. A profiler is mainly used to measure memory and CPU usage, memory leaks, program execution time, synchronization errors, and frequency of use of specific commands and functions. It also provides insights about program behavior and its impact on the system during program execution, such as a call graph, *etc.*

Table 1. Existing Dynamic Profiler that can Monitor Cache Hit Ratio

Tool	Environment	Monitored Target			GUI
		Cache-Level		Real-Time Application	
VTune	x86	L3	O	X	O
HPC Toolkit	-	L3	X	X	O
Cachegrind	ARM,PPC, x86	L2	O	X	O
*CC-Analyzer	x86, Xeon Phi	L3	X	X	O

Table 1 lists the dynamic profilers that can analyze the application's cache information. Vtune is a commercial profiler developed by Intel Corporation. In addition to analyzing problems, some solutions are presented through the analysis. The HPCToolkit can provide information about the CPU operation using the Hardware Performance Counter (HPC) API. Cachegrind is a tool built into Valgrind, a dynamic binary instrumentation framework. This tool measures the CPU usage of the target application and draws a call-graph. The CC-Analyzer is a tool that measures the cache hit-ratio and miss rate, and analyzes the causes using the dynamic binary instrumentation framework.

Existing tools such as those in Table 1 can provide information on cache hit-ratios. However, because conventional analysis tools analyze general applications, they cannot provide data necessary in real-time applications. Therefore, in this paper, in order to determine the interval in which the real-time property is violated by the cache hit-ratio, an analysis tool is presented that can provide information such as execution time, execution cycle, and cache hit-ratio of the actual function by tracking the operation of the application.

3. Design

It is important to ensure the real-time properties of real-time applications. However, real-time properties may be violated due to implementation or external factors. One external performance degradation factor is the cache hit-ratio. Existing analysis tools that detect and optimize the cache hit-ratio do not provide the functions that can verify real-time properties. In this chapter, in order to find the interval in which the real-time property is violated by the cache hit-ratio, an analysis tool is presented that can provide information such as execution time, execution cycle, and cache hit-ratio of the actual function by tracking the operation of the application. The analysis tool, seen in Figure 3, is designed to generate the analysis information by inputting the executable target

applications. The output is largely classified into three segments: basic information, cache misses, and real-time properties.

3.1. Basic Information

Basic information includes a thread identifier, function name, caller, and number of calls. The analysis tool collects and analyzes cache miss and real-time data based on these three data. The "number of calls" is the most basic function execution information that can be collected by simply tracking the execution of function. This value is a predictable value of how the function will affect the overall program performance. If the number of execution times is small, the quality improvement of entire program will be insignificant even if the quality of the function is improved. Therefore, this information is an index to know which function can be improved in order to improve the quality of application with the greatest efficiency.

3.2. Cache Miss

The cache-miss ratio shows the ratio of memory accesses that causes data cache miss. This information represents only the miss rate for the application's data read access. The cache-miss rate (LMissRate) per function (f) can be obtained using the number of read accesses (LAcc) and the number of times the memory accessed for reading did not exist in the cache (LMiss).

$$\mathrm{LMissRate}(f_n) = \left(100 * \sum_{n=1}^{x} \mathrm{LMiss}\left(f_{end(n)} - f_{start(n)}\right)\right) / \sum_{n=1}^{x} \mathrm{LAcc}\left(f_{end(n)} - f_{start(n)}\right)$$

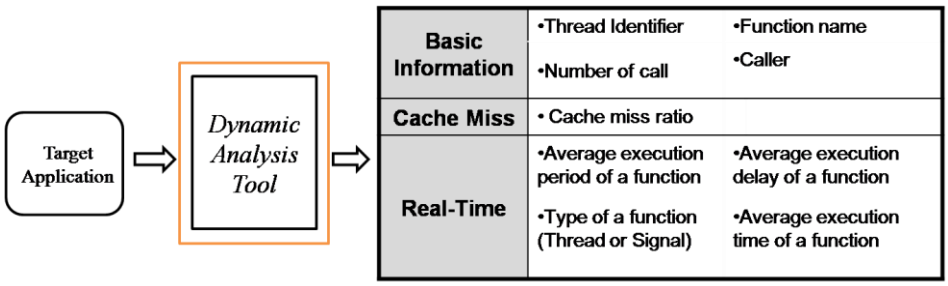

Figure 3. Output Information of Our Analysis Tool

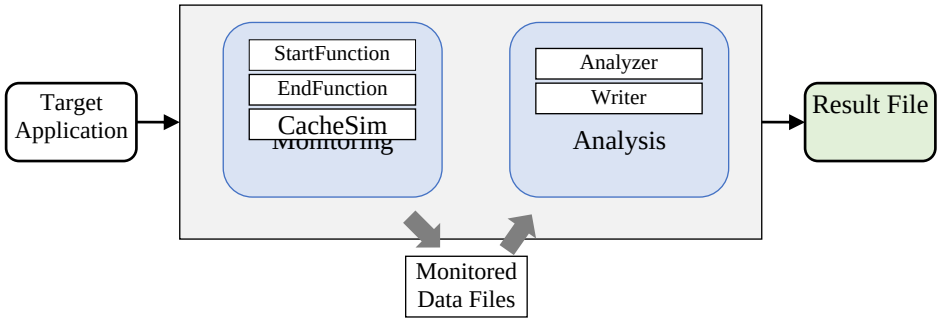

Figure 4. The Architecture of Our Analysis Tool

Here, *n* refers to the execution of the same function. For each execution of the function, both the number of read accesses and the number of times that the memory accessed by

the read did not exist in the cache are measured at the start and end points of the function, and by adding the all the measured values.

3.3. Real-Time

With regard to information with real-time properties, the average execution period, average execution delay, and average execution time of a function are numerical values that can determine whether the function violates deadline constraints. The average execution period of a function can be found mainly in the callback function of the Timer Signal where the occurrence cycle is set or in the loop where there is little difference in the external interference or the method of computation. With regard to average execution delay information of the function, when a delay of 5% or more is generated based on the average execution cycle at the time of measurement, the delay time is recorded and the number of delay times is counted.

The average execution delay of a function measures the start and end time of a function, then calculates the difference, and then calculates the average by dividing the total execution time by the total execution times of the function, which is the sum of all these values. The function type information distinguishes between a signal callback function and a function which is a simple thread. An operating system that uses preemptive scheduling can use this information to predict the priorities of the performed functions to a certain extent.

3.4. Design of the Analysis Module

The analysis tool consists of a monitoring module that collects application execution data and an analysis module that analyzes the collected data and generates meaningful information. The Monitor module's StartFunction and EndFunction are monitoring functions that run at the start and end of the function, respectively.

Both monitoring functions take a thread identifier and an identifier of the function to be analyzed as a parameter, and can determine which function to analyze. Both monitoring functions record the time of measurement, cache access count, cache miss count, and cache hit count in the "Monitored Cache Datafile" using the "CacheSim."

The "CacheSim" is implemented to simulate a cache. The "CacheSim" has the cache size, replacement policy, and the size and number of blocks; it records the number of cache accesses, misses, and hits. The Analysis module is an analyzer module that analyzes the collected data and generates it in the form of user-understandable information.

The "Analyzer" opens the "Monitored Data File" to analyze the syntax. The record of the file can distinguish whether it is the start point data or end point data of the function. This information is used to obtain information about the start and end points of the function. Then, by distinguishing the data with the same thread number, caller, and function name, it stores the data collected. At the same time, the number of calls to the function is counted, and the average execution time, average execution cycles, execution delay, and cache miss ratio are analyzed. This operation is repeated until all the log files are read. When the data analysis is completed, the analyzed data is generated in the form of a table.

4. Verification and Validation

4.1. Verification of Analysis Tool

The analysis tool was implemented using g++ 4.8.4 and Intel PIN 3.0 version on a Linux Kernel 3.13 environment. To run the analysis tool, type the following command:

pin -t tool.so - ./[application]

Pin is a tool that runs the analysis tool developed using the PIN framework [14]. The "-t" option allows the analysis tool to run in 32-bit if the program's execution environment is 32-bit, and 64-bit if it is 64-bit. "Tool.so" is the executable file of the analysis tool. Enter the symbol "--: to distinguish between the pin command option and the target application of the analysis, and enter the executable name of the target application.

The results are shown in the table in Figure 5. The table column entries are as follows:

TID	Caller	Name	NofC	AET(us)	IsF[Signal]	AF(us)	AToFV(us)	NofV(us)	AofCM(%)
0	frame_du...	register_tm_clones	1	7	F[T]	0	0	0	0
0	__libc_csu_...	main	1	3198128	F[T]	0	0	0	0.645251
0	__libc_csu_...	frame_dummy	1	0	F[T]	0	0	0	0
0	__do_globa...	deregister_tm_clones	1	7	F[T]	0	0	0	0
3	_Z12aperio...	_Z5func3v	6	3083	F[T]	0	0	0	3.34383
2	_Z12aperio...	_Z5func2v	5	1852	F[T]	0	0	0	2.54675
1	_Z12aperio...	_Z5func1v	3	2589	F[T]	0	0	0	2.31467
0	_GLOBAL_...	_Z41__static_initializati...	1	233459	F[T]	0	0	0	2.13287
0	main	_Z13timer_handleri	29	39427	T[S]	100044	112	1	2.52923
2		_Z13timer_handleri	1	158518	F[S]	0	0	0	3.4047
3		_Z12aperiodic_f3Pv	1	317384	F[T]	0	0	0	0.413878
2		_Z12aperiodic_f2Pv	1	240655	F[T]	0	0	0	0.1318
1		_Z12aperiodic_f1Pv	1	222020	F[T]	0	0	0	0.108063
0		_start	1	0	F[T]	0	0	0	0

Figure 5. An Example of the Analysis Report for the Target Application

- TID : Thread ID
- Caller: Function that called the function
- Name: Function that was called
- NofC: The number of calls for the function
- AET: Average execution time (*i.e.*, run time)
- IsF: If the function is executed periodically, output T; output an F symbol when it is not executed periodically. In addition, if it is a signal callback function, output [S], when it is not, output [T].
- AF: This is an average execution cycle. This value is generated only when the periodic execution of the function is detected.
- AtoV: Execution time significantly exceeding the average value in the average cycle. This value is generated only when the function is executed periodically.
- NoF: The number of times the mean value was significantly exceeded in the average cycle. This value is generated only when the function is executed periodically.
- AofCM: Average of cache miss rates during function execution.

Each column can be sorted in ascending or descending order by item, and the item can be hidden or displayed through the check box on the right.

By using the average execution cycle and execution time that the analysis tool represents, the matter of intended execution timekeeping can be confirmed. When a problem in the average cycle or execution time occurs, by checking the cache hit-ratio of corresponding function, the causes of cache hit-ratio can be confirmed.

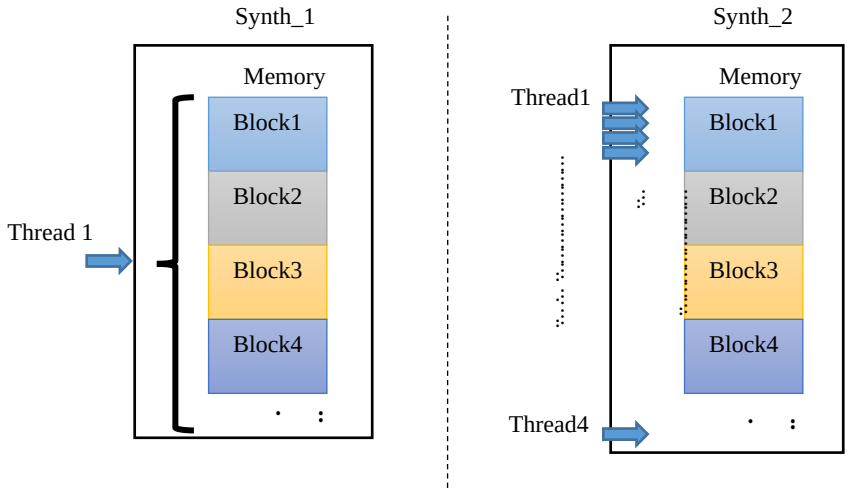

Figure 6. (Synth_1) A Synthetic Program with a Relatively High Cache Hit Rate, (Synth_2) a Synthetic Program with a Relatively Low Cache Hit Rate

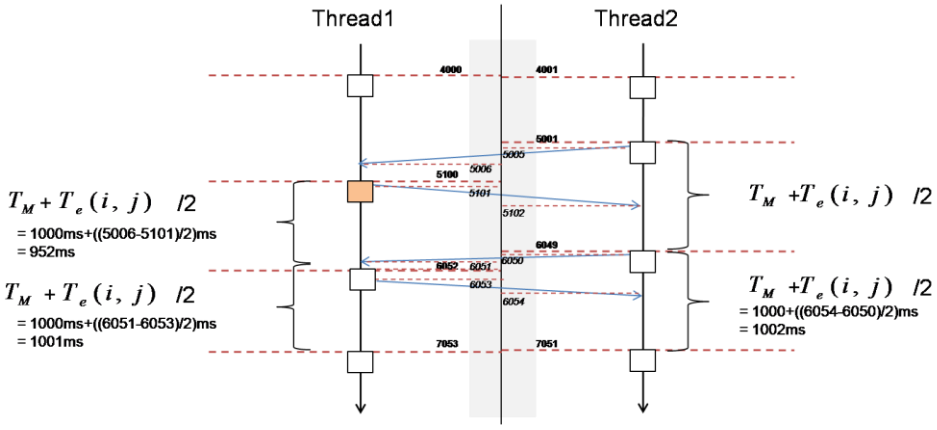

Figure 7. A Method to Adjust Execution Time Error of the Synthetic Program

4.2. Verification Procedure of the Analysis Tool

To test the accuracy of this tool, we used a synthesis program that periodically executes a Timer Callback function that deliberately increases the cache miss rate, and a synthesis program that has a thread that periodically executes an arithmetic function. The experiment was performed on an Ubuntu14.04 operating system with an RT-Linux Kernel 3.20 patch.

Figure 6 shows the operation of the synthetic program that intentionally generates a cache miss in a periodically executed Timer Callback function, and the synthetic program that performs relatively few cache misses, in order to compare the two programs. The image on the left side of Figure 6 is the synthetic program with few cache misses made for comparison. This program was implemented to read the contiguous memory space while the task is executed once, and to end the program after periodically executing it 30 times. Most of the time, because the uploaded block of memory was already present in the cache, it was configured so that the cache hit-ratio was not significantly reduced even if the task was re-executed and the memory was reloaded.

TID	Caller	Name	NofC	AET(us)	IsF[Signal]	AF(us)	AToFV(us)	NofV(us)	AofCM(%)
0	frame_du...	register_tm_clones	1	7	F[T]	0	0	0	0
0	__libc_csu_...	main	1	3198128	F[T]	0	0	0	0.645251
0	__libc_csu_...	frame_dummy	1	0	F[T]	0	0	0	0
0	__do_globa...	deregister_tm_clones	1	7	F[T]	0	0	0	0
3	_Z12aperio...	_Z5func3v	6	3083	F[T]	0	0	0	3.34383
2	_Z12aperio...	_Z5func2v	5	1852	F[T]	0	0	0	2.54675
1	_Z12aperio...	_Z5func1v	3	2589	F[T]	0	0	0	2.31467
0	_GLOBAL_...	_Z41__static_initializati...	1	233459	F[T]	0	0	0	2.13287
0	main	_Z13timer_handleri	29	39427	T[S]	100044	112	1	2.52923
2		_Z13timer_handleri	1	158518	F[S]	0	0	0	3.4047
3		_Z12aperiodic_f3Pv	1	317384	F[T]	0	0	0	0.413878
2		_Z12aperiodic_f2Pv	1	240655	F[T]	0	0	0	0.1318
1		_Z12aperiodic_f1Pv	1	222020	F[T]	0	0	0	0.108063

Figure 8. Analysis Results of the Synthetic Program that Read Sequential Space and Lead Relatively High Cache Hit Ratio

TID	Caller	Name	NofC	AET(us)	IsF[Signal]	AF(us)	AToFV(us)	NofV(us)	AofCM(%)
3	_Z5func3v	_Z13timer_handleri	12	130809	F[S]	0	0	0	24.5762
3	_Z12aperio...	_Z13timer_handleri	5	110775	F[S]	0	0	0	20.3163
0	main	_Z13timer_handleri	18	114708	T[S]	116456	-11690	7	18.8856
3		_Z12aperiodic_f3Pv	1	2205695	F[T]	0	0	0	17.426
0	__libc_csu_...	main	1	2419949	F[T]	0	0	0	16.6396
0	__libc_csu_...	__x86.get_pc_thunk.bx	1	2734254	F[T]	0	0	0	16.2035
3	_Z12aperio...	_Z5func3v	6	1412	F[S]	0	0	0	5.0098
2	_Z12aperio...	_Z5func2v	5	3307	F[T]	0	0	0	2.80105
0	__libc_csu_...	__do_global_dtors_aux	1	37896	F[T]	0	0	0	2.56741
0	__do_globa...	_Z13timer_handleri	1	36559	F[S]	0	0	0	2.56734
0	_start	__libc_csu_init	1	239388	F[T]	0	0	0	2.09955
0	_GLOBAL_...	_Z41__static_initialization_a...	1	233607	F[T]	0	0	0	2.09472
0	frame_du...	_GLOBAL__sub_I_test	1	234450	F[T]	0	0	0	2.09459
1	_Z12aperio...	_Z5func1v	3	2048	F[T]	0	0	0	1.52824

Figure 9. Analysis Result of the Synthetic Program that Read Non Sequential Space and Lead Relatively Low Cache Hit Ratio

The program represented by the image shown in the right of the Figure 6 was implemented to periodically repeat it 30 times, and then terminate like the program shown on the left. However, it is a program implemented to read non-contiguous memory space. It is likely that the memory block that is read is not in the cache, and when it is re-executed, the contents have already been replaced with another memory in the same block; hence, the cache data must be replaced again. If the cache hit-ratio is lowered, the cost of revisiting the memory and the replacement cost may occur, and the execution time may be longer than when the cache hit-ratio is low.

Figure 7 is the synthesis program in which the timing of execution of two computational threads that are executed periodically is synchronized. The quality required in this program demands that the sum of the input computation period and the error tolerance for the period must be less than the sum of the computation period entered and the error generated per computation "j" of thread "i". Each thread delivers the time of its computational execution to the partner thread. If the difference between the execution timing of the partner thread and that of its own operation is checked, the execution time error can be known. If the error tolerance is out of range, half of the error is added to the input calculation execution cycle of both threads to match each other's execution timing and normalize the cycle.

TID	Caller	Name	NofC	AET(us)	IsF[Signal]	AF(us)	ATofV(us)	NofV(us)	AofCM(%)
1	_Z16Server…	_Z15Com1ComputationP8S…	100	142	T[T]	103633	2729	2	0.00528701
3	_Z16Server…	_Z15Com2ComputationP8S…	100	74	T[T]	104281	2534	2	0.00230947
0		_start	1	0	F[T]	0	0	0	0
0	_start	__libc_csu_init	1	238496	F[T]	0	0	0	2.09637
0	__libc_csu_…	__x86.get_pc_thunk.bx	1	11256558	F[T]	0	0	0	0.771588
0	__libc_csu_…	_init	1	904	F[T]	0	0	0	0
0	_init	__x86.get_pc_thunk.bx	1	0	F[T]	0	0	0	0
0	__libc_csu_…	frame_dummy	1	0	F[T]	0	0	0	0
0	frame_du…	register_tm_clones	1	6	F[T]	0	0	0	0
0	frame_du…	_GLOBAL__sub_I_fp1	1	233562	F[T]	0	0	0	2.09141
0	_GLOBAL_…	_Z41__static_initialization_a…	1	232720	F[T]	0	0	0	2.09154
0	__libc_csu_…	main	1	10932666	F[T]	0	0	0	0.626064

Figure 10. Analysis Results of the Synthetic Program which Inserted 10*ms* Delay at Once in Two Times during Execution

TID	Caller	Name	NofC	AET(us)	IsF[Signal]	AF(us)	ATofV(us)	NofV(us)	AofCM(%)
1	_Z16Server…	_Z15Com1ComputationP8S…	99	48	T[T]	104121	5241	21	0
3	_Z16Server…	_Z15Com2ComputationP8S…	100	107	T[T]	104019	0	0	0.00531915
0		_start	1	0	F[T]	0	0	0	0
0	_start	__libc_csu_init	1	238845	F[T]	0	0	0	2.14084
0	__libc_csu_…	__x86.get_pc_thunk.bx	1	11212948	F[T]	0	0	0	0.754321
0	__libc_csu_…	_init	1	905	F[T]	0	0	0	0
0	_init	__x86.get_pc_thunk.bx	1	0	F[T]	0	0	0	0
0	__libc_csu_…	frame_dummy	1	0	F[T]	0	0	0	0
0	frame_du…	register_tm_clones	1	7	F[T]	0	0	0	0
0	frame_du…	_GLOBAL__sub_I_fp1	1	233900	F[T]	0	0	0	2.13273
0	_GLOBAL_…	_Z41__static_initialization_a…	1	233058	F[T]	0	0	0	2.12969
0	__libc_csu_…	main	1	10921249	F[T]	0	0	0	0.618537

Figure 11. Analysis Results of the Synthetic Program with Thread Performs Periodic Computation

4.3. Verification Result

Each synthesis program was analyzed using an analysis tool. It is shown that the intended result in the synthesis program was expressed by a numerical value that can be understood by the user using the analysis tool.

4.3.1. Synthetic Program that Induces Cache Hit-Ratio Degradation

Figure 8 shows the results of analysis of the synthetic program with relatively high cache hit-ratio sorted by name. In the case of the "Z13timer_handler," which is a timer callback function intended to execute the cycle, it is shown that it is a signal callback function executed 29 times with a priority preemption during the execution of thread 0. The average execution time is approximately 39 ms and was executed with a cycle of approximately 100 ms. This function has one cycle violation, but since the cache miss rate was approximately 2.5%, it was not caused by the cache miss; it can be hypothesized that this was caused by external factors.

The "'Z13timer_handler" was executed once during the execution of thread number 2, but because of the nature of tool to measure by thread, it was not recognized as a periodic function because the cycle could be measured in one run. However, since the execution time was 150 ms, it can be hypothesized that this execution affected the execution cycle of thread 0.

Figure 9 illustrates the analysis results of synthetic program with relatively low cache hit-ratio sorted by name. As a result of analysis, it was possible to see that the cache miss rate of the "Z13timer_handler" became much higher than the synthesis program had intended. The signal callback function must be completed within 100 ms at a cycle of 100 ms, but the execution time was longer than the cycle and it was out of the execution cycle. The measurement method of the analysis tool is as follows: if the previous execution

point and the next execution point are continuously similar to each other, the periodicity violation frequency may be higher than the measured seven times, because it is not regarded as a cycle violation even if it deviates much from the user's intended cycle. Since the cache miss rate became relatively higher in the "Z13timer_handler" function, the cause of increase in the execution time can be seen as a cache miss rate. In addition, in the case of the "Z13timer_handler" which was executed by interrupting the execution of another function in thread 3, because the execution time was not fixed owing to a high cache miss rate, the execution timing greatly deviated. Thus, it was not measured as a periodic function.

4.3.2. Synthetic Programs with Threads that Computes Periodically

A synthetic program with threads that computes periodically and operates every 100 *ms* was implemented to repeat a total of 100 times. The "Z15Com1ComputationP85" and "Z15Com2ComputationP85" are arithmetic functions performed periodically. Figure 10 accurately shows that '"Z15Com1ComputationP85" and "Z15Com2ComputationP85" operated for approximately 100 *ms*. Since each cycle had two cycle violations but a very low cache miss rate, it can be hypothesized that a cache miss occurred because of an external factor with a higher priority than the thread.

Figure 11 is the result of inserting a delay of 10 *ms* once per two executions in "Z15Com1ComputationP85." It shows that a delay occurred at "Z15Com1ComputationP85." If a delay occurs, the synthesis program immediately recovers and reduces the error so that there is no significant difference in the average execution cycle, leading to stable cyclical performance. The analysis results show that the average delay of approximately 5 *ms* occurred due to the recovery of about half of the delay, immediately after the delay occurred.

5. Conclusion

Since the breach of real-time properties of real-time applications may not only degrade the application quality but also cause damages to human life or property, the section that violates the real-time properties must be found and optimized. In this paper, we implemented an analysis tool that shows a function that takes a longer time to execute the application, periodic error, and cache hit-ratio of the function that is executed periodically. The analysis tool consists of a monitoring module that collects the execution data of the application by tracking functions and commands, and an analyzer module that analyzes the collected data and generates the execution cycle, execution time, cache hit-ratio, and information about the function executed in the application.

To test the accuracy of this tool, we used a synthetic program consisting of a thread that reads the memory at regular intervals, and a synthetic program that executes the computation periodically. The synthetic program consisting of a thread that reads memory every fixed period was divided into a program with a relatively high cache hit-ratio and a program with a relatively low cache hit-ratio. Experimental results showed that the synthetic program with a high cache hit-ratio had relatively short execution time and no violation of execution time. However, in the case of the synthetic program with a low cache hit-ratio, the results showed a relatively long execution time, and many ratios far out of the execution cycle. The synthesis program that executed the computation periodically showed that it can also accurately analyze the periodic execution of the thread. The analysis tool presented in this paper traces all executions of the applications and can provide information about the executed function, the execution cycle of the executed function, and the cache hit-ratio. However, the amount of overhead is large because the amount of system resources consumed during the execution of the analysis tool is great. Thus, there is a difference in the execution time of the actual application and the analysis results.

In order to reduce overhead in the future, it is necessary to develop an analysis tool that can report results that more identical to the actual performance using technology with less overhead than Dynamic Binary Instrumentation. In addition, improvements in displaying the execution time, execution period, and other information for each task in a parallel image are needed to help users understand them more fluently.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2060082).

References

[1] J. Nowotsch and M. Paulitsch, "Quality of service capabilities for hard real-time applications on multi-core processors", in Proceeding of the 21st International conference on Real-Time Networks and Systems, **(2013)**, pp. 151-160.
[2] J. Choi, E. Jee, H. Kim and D. Bae, "A Case Study on Timing Constraints Verification for Safety-Critical, Time-Triggered Embedded Software", in J. of Korean Institute of Information Scientists and Engineers, vol. 38, no. 12, **(2011)**, pp. 647-656.
[3] C. Christoph, F. Christian, G. Gernot, G. Daniel, M. Claire, R. Jan, T. Benoît and W. Reinhard, "Predictability considerations in the design of multi-core embedded systems", in Proceedings of Embedded Real Time Software and Systems, **(2010)**, pp. 36-42.
[4] M. Tolubaeva, Y. Yonghong and C. Barbara, "Predicting Cache Contention for Multithread Applications at Compile Time", in Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, **(2014)**, pp. 624-631.
[5] J. Reinders, "VTune performance analyzer essentials", in Intel Press, **(2005)**.
[6] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation", in Proc. of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, **(2007)**, pp. 89-100.
[7] A. Laksono, B. Sinchan, F. Mike, K. Mark, M. Gabriel, M. John and T. Nathan, "HPCToolkit: Tools for performance analysis of optimized parallel programs", in Concurrency and Computation: Practice and Experience, vol. 22, **(2010)**, pp. 685-701.
[8] R. Wang, Y. Gao and G. Zhang, "Real time cache performance analyzing for multi-core parallel programs", in Cloud and Service Computing (CSC), 2013 International Conference on. IEEE, **(2013)**, pp. 16-23.
[9] C. K. Luk, R. Muth, R. Cohn, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation", in Proceeding of the 2005 ACM SIGPLAN conference on Programming language design and implementation, **(2005)**, pp. 190-200.
[10] M. Barabanov, "A linux-based real-time operating system", in Diss. New Mexico Institute of Mining and Technology, **(1997)**.
[11] G. Buttazzo, "Hard real-time computing systems: predictable scheduling algorithms and applications", in Springer Science & Business Media, vol. 24, **(2011)**.
[12] M. Cho, S. Lee, W. Lee, G. Jeong, Y. Kim and C. Lee, "Deterministic Real-Time Task Scheduling", in Journal of Contents Association, vol. 7, no. 1, **(2007)**, pp. 73-82.
[13] B. Jacob, N. Spencer and W. David, "Memory systems: cache, DRAM, disk", Morgan Kaufmann, **(2010)**.
[14] A. R. Bernat and B. P. Miller, "Anywhere, Any-time Binary Instrumentation", in Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools, PASTE 2011, ACM, **(2011)**, pp. 9-16.

Authors

Hyang Yeon Bae, received the BS degree in Informatics from Gyeongsang National University (GNU), South Korea. She is now enrolled MS degree from the Department of Informatics in GNU. She worked as an engineer of avionics software in Korea industry for four years. Her research interests including parallel/distributed programming, embedded system programs, and software testing and debugging. Ms. Bae is a member of Korea Society of Computer Information (KSCI)

Ok-Kyoon Ha, received the BS degree in Computer Science under the Bachelor's Degree Examination Law for Self-Education from National Institute for Lifelong Education, and the MS and PhD degree in Informatics from Gyeongsang National University (GNU), South Korea. He is now a professor of department of aeronautics & software engineering in Kyungwoon University, South Korea. He worked as the manager of IT department in Korea industry for several years. His research interests including parallel/distributed programming, software testing and debugging, embedded system programs, dependable software, and software development activities for avionics. Dr. Ha is a member of Korean Institute of Information Technology (KIIT), Korea Institute of Information Scientist and Engineers (KIISE), and Korea Society of Computer Information (KSCI).

Yong-Kee Jun, received the BS degree in Computer Engineering from Kyungpook National University, and the MS and PhD degree in Computer Science from Seoul National University. He is now a full professor in the Department of Informatics, Gyeongsang National University, where he had served as the first director of GNU Research Institute of Computer and Information Communication (RICIC), and as the first operating director of GNU Virtual College. He is now the head of GNU Computer Science Division and the director of the GNU Embedded Software Center for Avionics (GESCA), a national IT Research Center (ITRC) in South Korea. As a scholar, he has produced both domestic and international publications developed by some professional interests including parallel/distributed computing, embedded systems, and systems software. Prof. Jun is a member of Association for Computing Machinery (ACM) and IEEE Computer Society.