ISSN: 0975-766X
CODEN: IJPTFI
Available Online through *Research Article*
www.ijptonline.com

OPTIMIZED L3 CACHE PARTITIONING FOR VIRTUALIZED ENVIRONMENT WITH GPOS AND RTOS

R.Ezhilarasie* and Dr.A.Umamakeswari
School of Computing, SASTRA University, Thanjavur-613401.
Email: ezhil@cse.sastra.edu

Received on 06-08-2016 *Accepted on 10-09-2016*

Abstract

The concept of caching the memory to speed up the memory transactions is not new. Cache memory at different levels help reduce the memory access time by different factors depending on the location of the cache in the system. Talking about a virtualization environment, wherein more than one OS run on a hypervisor, each OS has a dedicated L2 cache for its core and there exists a L3 cache or last level cache that is shared by all the OSes running on the hypervisor. Taking a particular case of a system with one Real-Time Operating System and one General Purpose Operating System sharing the last level cache, competition may arise. Problems like, inability to meet the deadlines by the processes in RTOS and performance aberrations in GPOS, can creep in due to inefficient sharing of last level cache. Cache Partitioning or CAP and Page-table Prefetching or PTP are two methods that overcome the mentioned problems one each. CAP approach is biased towards the GPOS whereas PTP approach favors the RTOS. Hence, there is a need for an approach that attempts to minimize the latencies in RTOS, minimizing cache misses for GPOS processes as well. Nesting PTP in CAP (PIC) is the proposal of this paper. To quantitatively depict the latency differences the Cyclictest has been used, which is a program that notifies the minimum, average and maximum latencies by a process. Improving the RTOS response without affecting the GPOS by proper sharing of Last Level Cache is the aim.

Keywords: Cache; Virtualization; RTOS; latency; cyclictest;

I. Introduction

Virtualization has rooted itself in all known fields of software and hardware. Para-virtualization and full virtualization being its two major types. Para virtualization has a collaborative interplay of the hypervisor and the OSes running on it, as the different virtual OSes are aware of each other's presence. This reduces the complexities which otherwise the

hypervisor would solely have to bear. On the other hand, in a full virtualized environment the running virtual machines are completely unaware of other's existence. The hypervisor takes care of multiplexing the resources between competing operating systems. Now, since the word competing has appeared, lets understand the gravity of it. The operating systems running on a hypervisor need memory space, processor cycles, IO space and cache. Since the very idea of virtualization is to accommodate more than one operating system in a single hardware system, it gets obviated that the guest operating systems will have to share the hardware resources. This sharing mechanism is crucial for the performance of each operating system. To elucidate this, here goes an example of a resource that will be shared by running operating systems – Cache memory. Let it be duly informed that the guest operating systems or the virtual machines may run on dedicated cores in a multi-core processor system. Each core has one dedicated cache called the level 2(L2) cache and a shared L3 cache also called as last level cache(LLC). Caching reduces the number of main memory accesses hence reducing the overall instruction execution time. So if one virtual machine fills up the cache with its data the other virtual machine will suffer from a large number of cache misses, degrading its performance. This situation is also popularly known as 'cache pollution'.

This paper focuses on running an RTOS and a GPOS on a single hardware machine with the help of full virtualization. A Real time OS is primarily concerned about its task deadlines being met. Since this RTOS would think that it is the only OS running, it would have its tasks scheduled with caches in consideration but with no competition for them. In such a case if the co-existing GPOS pollutes the L3 cache then the chances of RTOS tasks missing their deadlines would hike up. In mission-critical applications real-time performance of the RTOS should not be compromised. Partitioning the cache statically in a particular ratio is one solution, which has been implemented in a method called CAP- cache partitioning. It makes use of page coloring technique to bind the data from one core to a limited portion on the L3 cache. Although this would mean that a portion of LLC will permanently be booked for GPOS and the RTOS would not be able to use it, in case it uses-up its own allotted portion and needs more, even though the GPOS might not be completely using it either. Another method, Page-table Prefetching (PTP), was proposed in a paper[1] which overcame the issue in CAP. Page tables with entries of the addresses of instruction from RTOS are pre-fetched and kept in the cache, ensuring no cache misses. Although PTP seems perfect for the RTOS, GPOS performance may get hampered significantly. This paper proposes a method which nests PTP in CAP (PIC) to incorporate best of both the existing methods. KVM (Kernel Virtual Machine) facility provided by linux as a loadable kernel module is selected to support the virtualization framework in this paper. KVM is the hypervisor on which one

RTOS and a GPOS are going to be mounted as virtual machines. With a little modification in the memory allocator module of the linux kernel, incorporation of PIC in the system will be possible.

II. Problem statement and AIM

A RTOS and a GPOS combination is going to be a common sight soon, with a range of applications requiring a general-purpose OS for normal features and a real-time OS for timely deterministic response. A system for stock market which would want a real-time OS to deal with the pouring real-time data and a general purpose OS to handle security features and others, can serve a use case.

A. Cache pollution problem

The Virtual machine manager segregates the different guest virtual machines but the response gets reduced. To check the same we have used cyclictest which measures the latencies of real-time OS. A comparison is made with the native linux. While using the cyclictest the GPOS and the RTOS are pinned to two different cores. The native linux and the guest RTOS both are PREEMPT-RT patched. 10,000 times repetition of the cyclictest on both proves our assumption of guest RTOS getting degraded in performance. VMM strives to isolate the hardware resources for different VMs but the last level cache is shared among the cores and hence is shared by the different operating systems running as virtual machines on the KVM hypervisor. This LLC is important in reducing the data fetching time by the processes. If inefficiently shared, the GPOS might conquer on a large portion of the LCC space leaving the RTOS with increased cache misses, hence degrading the performance. The vice-versa is also possible, in which case the GPOS will be the victim. This is what is called as cache pollution, which needs to be addressed to maintain the performance of the overall virtualized system

B. The existing approaches

Cache Partitioning or CAP is method that uses page coloring technique to statically divide the area in last level cache for each core running fluctuate in real-time. It is possible that the real-time OS is hungry for cache whereas the general purpose OS is not even utilizing the allotted portion, but still RTOS cannot access the unused cache space on the other side.

Page-Table Prefetching or PTP is another cache optimization method that strives at eliminating the drawbacks of CAP. This is a dynamic approach which prioritizes RTOS over its counterpart. The data from RTOS gets pre-fetched into the LLC, hence the data fetching time will be reduced improving the response time. Though any data from RTOS can be kept in the LLC, page table entries are selected to be pre-fetched. The data is paged and is referenced through

page table entries which are resolved using a TLB (Translation look-aside buffer). Since a TLB is very small in size, it can become a bottleneck. This is the reason for selection of page table entries and not just any data from RTOS to be placed in the last level cache. PTP ensures that RTOS process does not have to be on the longer route, all the way to the main memory, most of the time, but it overlooks the other guest, which is a GPOS, completely. GPOS performance may decline significantly due to this negligence. CAP and PTP, as seen here, are effective for GPOS and RTOS in that order. Apparently, none of the two can singularly address woes of both the guests. This renders them unpopular in market.

III. The Nesting Approach

To overcome the biasness of CAP and PTP towards GPOS and RTOS respectively, this paper proposes to combine the two methods constructively. The upcoming section will describe the nesting approach in detail.

A. Introduction to KVM

Kernel Virtual Machine is hypervisor inbuilt in linux kernel as a loadable module. This feature provided by linux makes it easy to use a virtualized environment on this platform. Intel-VT and AMD-V, are the two extensions of modern Intel and AMD processors that facilitate the functioning of the KVM. KVM is a kernel module which comes loaded in the memory of the linux kernel that utilizes the processor. A number of virtual machines can be created and run on this hypervisor. These virtual machines can be different operating systems, which don't have to be homogenous in nature. A slightly modified version of QEMU is used a user-mode driver. Virtual machine manager or VMM is the popular GUI to manage virtual machines on the KVM. VMM strives to isolate the hardware resources for different VMs. The two virtual machine here are the RTOS and the GPOS which are pinned to one core each.

The two virtual machine here are the RTOS and the GPOS which are pinned to one core each. Dedicating a separate core will help in implementing page coloring technique which can be taken advantage of.

B. Page Coloring

Page Coloring is the mechanism using which the contiguous pages of a certain 'color' can be placed adjacent to each other in the cache too. Most commonly used cache type today is set-associative cache type RTOS and GPOS exclusively. Page coloring marks the set of instructions from one core as one color. Thus while mapping the addresses from main memory to the set-associative cache, the pages with same color get placed together in one set. So if the two guests are on two different cores, which will be ensured while creating the virtual machine, the addresses from the RTOS will not try to take the space allotted for GPOS pages. This solves the problem of cache

pollution just fine, except that any predetermined ratio may prove inappropriate as RTOS's needs for cache can. The cache is divided into sets. The data coming from the memory identifies the set to which it should go with the help of a cache set index, which is nothing but a certain fixed number of bits in the physical address. The virtual memory is translated to the physical memory using some techniques called as paging. Segmentation is also used in case of still larger virtual memory. These pages are again identified with the help of a page index, which is also a certain fixed number of bits in the physical address. The page index and the cache set index are both part of the physical address. They happen to intersect always.

This intersecting set of bit is what is refered to as the page color. Due to its uniqueness it can be used as an identifier. This identification is used to segregate the pages coming from different cores to two separate spaces in the LLC. The page coloring is used by making modifications in the memory allocator module of the native linux kernel. Most commonly used type of cache used is set-associative type. Certain bits of physical address indicate the cache set index.

These cache set index bits intersect with the page index. These intersecting bits are called as the page color. Hence the pages from one core can always be identified as one page color, therefore can be segregated from the pages of the other core. This segregation helps in statically dividing the LLC space for the two cores. This is going to be the base part of the source program which encompasses the CAP strategy. To implement it a loadable kernel module is created, which is a modified allocator program for the native linux kernel.

C. Loadable Kernel Module

LKM is a way to design and implement linux kernel module or drivers. The traditional method of including a kernel module involves making a source file, adding this source file to the kernel source tree and recompiling the kernel. This method is not so flexible in the sense that this method does not allow to add a module ad-hoc way. That is, an extension cannot be attached while the kernel is running. For this reason use of LKM is preferred greatly, as it facilitates the flexibility of loading and unloading a new module on the run. Rebuilding the kernel as often would not be required.

This saves a lot of time as kernel building is a time consuming process. Another vital advantage of LKM is that it makes the diagnosis of problems a lot easier as compared to the kernel bound modules. If a kernel bound module has a bug it could halt the entire system and isolating the problem causing module can prove to be a real task. Typically all the LKM object files are kept together in one directory. The **insmod** command can be used, with the name of the

LKM object file that is to be inserted. For the purpose of this paper a source code is written for a modified memory allocator and an object file is made for it. By inserting this object file using insmod, this module becomes a part of kernel and runs its functionalities.

D. Using the prefetch function

As mentioned before, to ensure that the RTOS does not have to be cache hungry very often, a function is developed for prefetching the data from the RTOS VM and placing it in the LLC space reducing cache misses. To implement this, a function is written that uses a built-in function namely builtin_prefetch. The process id (pid) of the RTOS VM is passed as an argument to the built-in function. In the introduction to KVM it was mentioned that the virtual machines run like a process on the native linux kernel. To get the process id of the VMs running on a KVM there are two prescribed methods. If libvirt manager is being used then a command sudo grep pid /var/run/libvirt/qemu/GUEST_X.xml can help. Notice here, GUEST_X is the name of the guest OS running as the VM on the KVM. So in our case it is RTOS.xml. Another method to determine the pid of the VM is by using ps. The complete command can look like this ps aux | grep GUEST_X . This built-in function is used to pre-fetch the page table entries of the RTOS processes. Our module makes use of this function by calling it in function of our own definition which is branched out only if the memory space in LLC allotted for RTOS during the page coloring falls short.

IV. Experiment and Analysis

E. Experimental platform

The experiment is run on a Laptop 2.60GHz Intel i3 CPU (Dual-core) with 128KB of L1 cache, 512 KB of L2 cache and 3 MB of L3 cache and 4GB of memory. The version of Qemu-KVM is 2.0.0.The host linux kernel version is 3.12.37-rt 51. One core is dedicated for GPOS and another for RTOS with 1 GB of virtual memory allocated for each. Cyclictest is a program that measures the performance of the RT operating system. Repeated testing of the product to a great number of times is done to ensure the reliability. The cyclictest program puts a timestamp initially and goes to sleep. As the clock interrupt occurs the program is woken up and another time stamp value is taken. The difference of the two is calculated to give the latency value. The minimum, average and maximum values of latency for a certain process is calculated and printed out in the desired format. Another program called ‘stress’ was used to add a uniform load to the processor. This the erratic output values that may appear due to fluctuating load on the processor.

F. Comparison with native

The latency values were observed in the native linux kernel by running the cyclictest with appropriate arguments. Then the cyclictest command was run on the guest RTOS which was made fully pre-emptible using *make config* features. The average value of latency in the guest RTOS was almost three times that of the latency value in native linux, which too was made fully pre-emptible beforehand. The maximum latency values showed many times hike in the guest real time operating system. This clearly indicates that the performance of a real-time OS is significantly reduced when running as a virtual machine. This latency difference obviously cannot be eliminated, though to reduce it several attempts have been made. In related paper published earlier the CAP and PTP methods were implemented and the results were observed. The results, as discussed in earlier sections were significant but biased. Hence this paper has tried to overcome this problem of biasness.

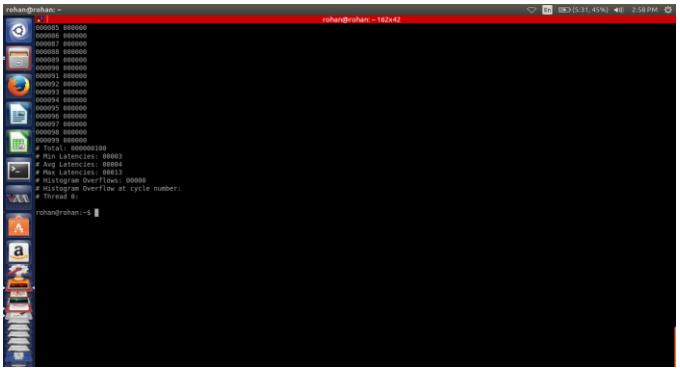

Figure 1. Cyclictest on native linux.

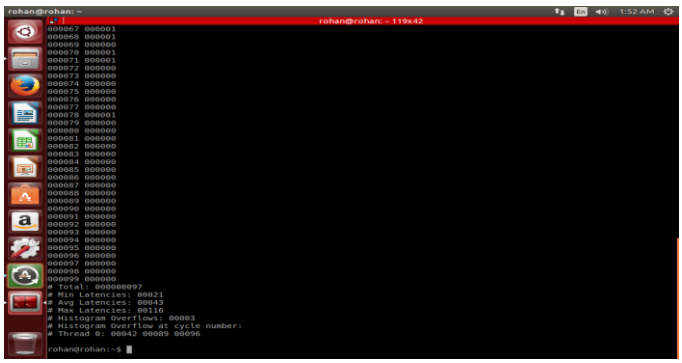

Figure 2. Cyclictest output on RTOS VM.

G. Results after nesting

As usual the cyclictest was performed after the LKM object file was included using insmod in the native kernel. The point to be kept in mind is that the cyclictests are now being performed on the VM which is a real-time OS and not the native real-time OS, even though the LKM was included in the native linux kernel.

The snapshot in figure 3 depicts the latency report of the RTOS VM. As can be seen the average, minimum and the maximum latency values are all lesser than those observed before the inclusion of the module into the native linux

kernel. The maximum value in the output may fluctuate at times. This happens as any one erratic process might have accessed data through the long route during the cyclictest run.

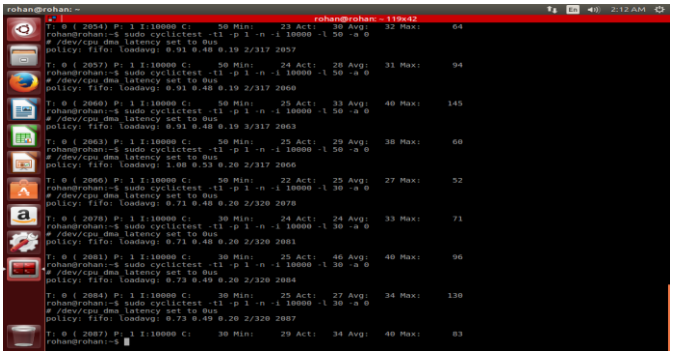

Figure 3. Cyclictest on the RTOS VM after code implementation.

V. Related Works

In the work that has previously been published, simulation models have been used to test many proposed methods for cache optimization. Some cache space is allocated generally for every thread that is running on the system. Based on the requirement of cache size of partition is adjusted. But it has so been proved that cache optimization strategies running on physical machine give better results over simulation mode.

As a result of which static as well as dynamic division of LLC has become a common topic for research work. Page coloring technique has been used for the cache partitioning on operating systems. Cyclictest [3] is used to determine the latency value of real-time OS.

VI. Conclusion and Future Works

With the implementation of Page-table prefetching method nested in Cache partitioning method, a better performance is achieved for the real-time OS running in virtualized environment which has a GPOS as another VM running on the same hypervisor. The cyclictest results conform to the expectation.This shows that in virtualization the sharing of the hardware resources which cannot be isolated for each VM can be done effectively. This encourages the development of one GPOS one RTOS virtualized systems. In future this method can be implemented on Xen hypervisor and to enhance its scalability SPEC2006 benchmark can be used to put more load on GPOS systems and test for heavy applications.

References

1. Ruhui Ma, Wei Ye, Alei Liang ,Haibing Guan, Jian Li ,Cache isolation for virtualization of mixed general-purpose and real-time systems ,2013.

2. A. Kivity, Y. Kamay, D. Laor, KVM: the Linux virtual machine monitor, in: Proceedings of the Linux, Symposium, 2007.

3. Cyclictest. Available at: <http://rt.wiki.kernel.org/index.php/> (Cyclictest 29.12.10).

4. J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, P. Sadayappan, Gaining insights into multi-core cache partitioning: bridging the gap between simulation and real systems, in: 14th Intl. Symp. on High-Performance Comp. Arch. (HPCA), 2008.

5. X.N. Ding, K.B. Wang, X.D. Zhang, ULCC: a user-level facility for optimizing shared cache performance on multicores, in: Proc. of 16th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP ,2011), San Antonio, TX, February 12–16, 2011.

6. Bach D. Bui, Marco Caccamo, Lui Sha, Joseph Martinez, Impact of cache partitioning on multi-tasking real time embedded systems, in: Proceedings of the 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, August 25–27, 2008, pp. 101–110.

Corresponding Author:

R.Ezhilarasie*,

Email: *ezhil@cse.sastra.edu*