Virtual Machine Coloring for Resolving Cache Conflict

Youngjun Jang*, Jaeook Kwon*, Sungwon Ahn#, Cheolho Hong# and Chuck Yoo#
Department of Computer Science and Engineering, Korea University, South Korea*
Samsung Electronics, South Korea*#
[e-mail: yjjang@os.korea.ac.kr]
*Corresponding author: Chuck Yoo

Abstract

Cache conflict is a common problem when a shared cache is used in multi-core processors. In a virtualized environment, the performance of one virtual machine can be degraded because of other virtual machines due to cache conflict. The page coloring method can resolve cache conflict by partitioning a shared cache into several separated caches. However, this method cannot be adapted when a guest operating system (OS) requires physically contiguous memory property like huge page support. This paper proposes a virtual machine coloring technique that minimizes cache conflict while maintaining the physically contiguous memory property. Furthermore, this paper shows performance improvement on a real environment.

Keywords: virtualization, cache partitioning, coloring, virtual machine coloring, physically contiguous memory

1. Introduction

A multi-core processor today has shared cache and it contributes to the enhancement of a chip’s overall performance. However, in some cases, a shared cache may have a negative effect when there are few shared resources between applications running on different cores but that shares the cache. Cache conflicts between cores can reduce a core’s cache hit rate. To deal with this situation, cache partitioning [1] is used to divide the cache among different applications competing for the same cache entries. Page coloring [2] is a representative method for cache partitioning that can be implemented with only software modifications; other techniques require modifications at the hardware level. Page coloring can also be adapted for a virtualization environment [3]. Cache partitioning is established by applying page coloring to different virtual machines. However, page coloring has its disadvantages: pages are fully fragmented so a guest OS, which receives fragmented memory from the hypervisor, cannot utilize the features related to physically contiguous memory. For example, the huge page feature [4], the continuous memory allocator feature [5], and some device drivers that need a physically contiguous buffer on the Linux do not work with page coloring; so if any of these features are used in a guest OS, page coloring cannot be used.

This paper proposes a new technique, “Virtual Machine Coloring,” which minimizes shared cache conflict while maintaining the physically contiguous memory property in a guest OS. It helps improve performance when page coloring cannot be adapted. This paper also shows an improvement in performance in a real virtualized environment under a network packet forwarding
2. Background

There are many reasons why OS tries to maintain physically contiguous memory. The first is that some devices that use DMA (Direct Memory Access) require this feature. The second is when a huge page is required; for example, the Linux typically uses a page size of 4KB, but in some cases a huge page (2MB, 4MB) can be more useful. A huge page size also reduces the number of TLB (Translate Look-aside Buffer) entries to maintain and reduces address access time, thus helping improve performance. To support a huge page, the hypervisor and the OS should support the merging of several small pages into single huge page.

Page coloring cannot be used with the above feature. Figure 1 shows how cache partitioning works and utilizes memory and cache. The hypervisor owns all physical memory pages and it groups pages into several regions that have same color. All cache lines in a physical page are cached in one of those regions. The hypervisor also maps some regions into a virtual machine (VM). No region includes contiguous pages as shown by the black color case on Figure 1, so the VM cannot obtain physically contiguous pages at all; eventually, all memory of each VM becomes fully fragmented.

3. Virtual Machine Coloring

By the principle of locality [6], some codes are executed more frequently than other codes. Similarly, some data values are referenced more frequently than other data values. If we divide a runtime application into cache line granularity, some separated line will be accessed very frequently (hotspot) and other lines will be accessed less frequently. Suppose that two same applications are run at the same time under a similar scenario; then, the address of the code, data and heap areas will be same. Further, the address of the hotspot will be same. However, if they run on different CPUs that share a cache, then a cache line conflict will occur on the hotspot, and eventually performance degradation will occur. VMs running on a multi-core machine will suffer from a similar problem.

![Fig. 1. Cache partitioning by page coloring](image)

In this way, we can use either physically contiguous memory or cache partitioning. Therefore a new solution is required to take advantage of both features.

![Fig. 2. Memory address changes with VM coloring](image)

Our suggested method is illustrated in Figure 2, which shows the effect of VM coloring. Before VM coloring, the locations of hotspots are the same between two VMs, so both are mapped onto the same cache line and conflict occurs. After VM coloring, the hotspot location of VM2 is moved to another address, so the hotspots of each VM are mapped into another cache line; in this case, hotspot conflict does not occur.

The idea of VM coloring is very simple. Our method shifts all the memory addresses of a VM with a specified size by adjusting the start offset of each VM. Although this method cannot provide a fully partitioned cache to each VM, it helps reducing cache conflict because hotspots of each VM will occupy a different cache line. Moreover, the method still supports the physically contiguous memory feature such as huge page because page arrangement of each VM remains unchanged.
3. Evaluation

A network packet forwarding scenario was used for evaluation. It is an appropriate example that will show the effect of VM coloring. Two separated packet forwarding modules ran on each VM and used the same OS image and module. Each VM used a different NIC (Network Interface Card) port, and only their network configuration differed. Each VM represented one packet forwarding module that forwards packets at the full NIC rate. Each VM used two dedicated cores and 2GB RAM. All cores that a VM ran shared the same L3 shared cache.

The experimental environment included an Intel X5650 (two six-core processors, 12MB L3 shared cache), 16GB RAM and an Intel 82599 10Gb/s NIC. Xen [7] and Linux were used to implement and evaluate virtual machine coloring since they are open source platforms and support the huge page feature.

The result showed that using one VM, it can deliver 2.68Mpps (mega packet per second). However, when two VMs ran at the same time, the performance degraded to 2.39Mpps, mainly because one VM interfered with the other.

The result showed that using one VM, it can deliver 2.68Mpps (mega packet per second). However, when two VMs ran at the same time, the performance degraded to 2.39Mpps, mainly because one VM interfered with the other.

![Fig. 3. Packet throughput with various VM coloring offset](image)

Figure 3 shows the performance after applying the proposed VM coloring technique. X-axis means the size of offset shifted. The offset 0 means no VM coloring is adapted. With varying offset from 1MB to 4MB, average packet throughput increased from 2.39Mpps to 2.48 ~ 2.51Mpps.

4. Conclusion and Future Work

In this paper, we demonstrated that VM coloring can improve the performance of a VM by resolving the shared cache conflict while maintaining physically contiguous memory property.

Currently this technique is useful only when two or more similar VMs run at the same time, and there is no method for the determination of the best offset value. Our future research will focus on optimizing offset values. If we can evaluate the memory access pattern of a VM – that is, which address is hotspot – then we can predict best offset value more easily.

References


