Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 362–371

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

GPU-Assisted Buffer Management

Jianlong Zhong Bingsheng He[1]

Nanyang Technological University, Singapore

Abstract

Cloud computing has become an emerging virtualization-based computing paradigm for various applications such as scientific computing and databases. Buffer management is an important factor for the I/O performance of the virtualized platform. In this study, we propose to leverage the memory and the computation power of the graphics processors (GPUs) to improve the effectiveness of buffer management. GPUs have recently been modeled as many-core processors for general-purpose computation. Designed as co-processors, they have an order of magnitude higher computation power than CPUs, and have a large amount of GPU memory, connected to the main memory with the PCI-e bus. In particular, we present two approaches of GPU-assisted buffer management, namely *GRAM* and *DEDU*. GRAM utilizes the GPU memory as additional buffer space and models the main memory and the GPU memory as a holistic buffer, whereas DEDU performs GPU-accelerated de-duplication to increase the effective amount of data pages that can fit into the extended buffer. We optimize both approaches according to the hardware feature of the GPU. We evaluate our algorithms on a workstation with an NVIDIA Tesla C1060 GPU using both synthetic and real world traces. Our experimental results show that the GPU-assisted buffer management reduces up to 68% of I/O cost for the traces generated from Xen.

Keywords: GPGPU, Memory Management, Virtualization, Deduplication, Cloud Computing

1. Introduction

Cloud computing (also known as utility computing) has emerged as a new computing paradigm for various applications, including scientific computing, databases and service-oriented computing. One of the key techniques for cloud computing is virtualization. Through virtualization, multiple virtual machines concurrently run on a physical machine, sharing the computation, memory and network resources. Memory management (or buffer management) has been a hot research topic for virtualization [1, 2, 3]. Previous studies have shown that the content redundancy becomes a new dimension for memory management. In the virtualized environment, there exists considerate data redundancy among the operating system images (especially for those virtual machines with the same operating system) and from the applications. In this study, we investigate how the existing hardware in the commodity machine improves the efficiency and the effectiveness of memory management, especially for the virtualized environment.

Memory management is a core and important component in the systems when the data cannot fit into the main memory. Clearly, effective buffer replacement policies [4], large buffer sizes and main memory compressions [5] are important issues in improving the buffer effectiveness. Given a fixed-sized buffer, researchers basically focus on two

[1]Email: bshe@ntu.edu.sg

Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.038

aspects of buffer management: predicting the set of hot pages, and packing more pages into the buffer. Dozens of replacement policies have been proposed to capture the set of hot pages by their recency and frequency in the history, e.g., LRU (Least Recently Used) and LIRS [4]. Recently, there have been proposals on replacement polices for multi-level memory hierarchy [6] and for new devices such as solid state drives [7]. To pack more pages into the buffer, main memory compressions [5, 8] are the common methods to reduce the data size. Due to the computational overhead, main memory compressions often require special and often costly hardware [9], or with a coarse granularity. For example, in VMware ESX Server, the default memory scan for compressions is once an hour, with a maximum of six times per hour. Encouraged by the success of GPGPU (General-Purpose Computation on Graphics Processing Unit), we consider leveraging the GPU to alleviate the computational overhead and to extend the buffer size.

GPUs, originally designed for graphics rendering and gaming applications, have recently evolved into many-core processors for general-purpose computation, and GPGPU has become an active research area [10]. The GPU has over an order of magnitude higher computation capability than the CPU in terms of GFLOPS (Giga Floating Point Operations per Second). This is because the GPU devotes the majority of the die area for execution logics, whereas the major die area of the CPU is contributed to the cache. Moreover, due to the recent introduction of 64-bit addressing on the GPU, the GPU has a large piece of video memory. For instance, the NVIDIA Tesla M2070 has 6GB RAM, which can be larger than the main memory in some machines. While the data in the GPU memory is accessed via the PCI-e bus, the access latency is still over two orders of magnitude lower than the disk access (Section 2.1). When there are few rendering tasks for the GPU, the GPU is underutilized. The free GPU memory can be used as extra buffer space for storing the buffer pages, and the GPU computation power can be leveraged to perform compressions.

In order to realize the GPU-assisted buffer management, we need to manage two memory areas, connected with the PCI-e bus. Since the PCI-e bus does induce latency to the page access, hotter pages should be kept in the main memory buffer other than the GPU buffer. Moreover, the PCI-e bus is usually a significant factor in the performance of GPGPU applications [11]. In particular, we are facing the following challenging design issues:

1. What data should be stored in the GPU memory? As an extended buffer, we should consider the recency and the frequency of data pages to maximize the utility of the GPU buffer.
2. How to handle the page accesses in the GPU buffer, such that the bandwidth of the PCI-e bus is fully utilized? A data transfer on the PCI-e bus involves a fixed cost of initializing the bus. The page accesses to the GPU buffer should be designed with this overhead in mind.
3. How to efficiently compress the data pages at runtime? Compressions are not without drawbacks. While compressions can reduce the memory space required for the same amount of data, they introduce the runtime computational overhead. Moreover, some data types, such as those have been compressed, cannot be effectively reduced by compressions. Thus, we must develop a compression algorithm to balance between the gain of reduction on the data size and the runtime overhead.

Attempting to address these issues, we have developed two GPU-assisted buffer management algorithms, called *GRAM* and *DEDU*. GRAM manages all the pages within the GPU buffer and the main memory buffer in a uniform buffer management policy. For example, LRU manages all the pages in a single LRU queue, consisting of pages from the main memory buffer and the GPU buffer. Due to the data transfer cost of the PCI-e bus, the GPU buffer is designed to be an eviction cache. Moreover, the page accesses to the GPU buffer are performed in a batched way, so as to amortize the cost of the PCI-e overhead. The batch size is a tuning parameter for GRAM to balance the response time and the throughput. Since all the pages in the extended buffer are managed by the same replacement policy, the fruitful results in the buffer management research are relevant to GRAM.

Enhancing GRAM with data compressions, DEDU further packs more pages into the extended buffer with deduplications [12]. Deduplication has demonstrated to be useful in reducing the effective data size for archives [13] and memory management of virtual machines [1]. In our implementation, a page is divided into multiple blocks of the same size. Deduplication is applied to remove the redundancy among data blocks. Since the blocks are with the same size, deduplication well fits the GPU computation model. Moreover, deduplication is highly parallelizable, with intra- and inter-block parallelism. Due to the massive computation power, the GPU supports a high throughput of deduplications, and the most time consuming part of the deduplication is off-loaded from the CPU to the GPU.

We conduct experiments on a NVIDIA GPU of 240 cores and 4GB device memory. We evaluate the effectiveness of GRAM and DEDU with synthetic workloads in order to fully control the parameters to assess the design of GPU-assisted buffer management. The results show that: a) batched processing significantly improves the bandwidth