Prefetching with early promotion of huge pages in GPGPU applications under oversubscription scenarios

Nikhil Molugu, Debadatta Mishra, Swarnendu Biswas
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
{nikhilm, deba, swarnendu}@cse.iitk.ac.in

***Abstract*— .**

***Index Terms*— GPGPU, CUDA, Prefetcher**

I. INTRODUCTION

In today's world, intelligent systems (AI/ML applications) are very popular and widely used. These applications train on big data sets and require high computational resources. These application's training behaviour is more suitable for massively multi-threaded processing. Therefore GPUs and accelerators have been the obvious choice for these types of applications. Now with MPS [NVIDIA MPS], GPUs can run kernels from different contexts enabling the GPUs to be shared even more efficiently. Traditional discrete GPU's programming model required copying of the data from the host memory to device memory before launching a kernel (code that runs on GPU) on GPU. The device and the host have different address spaces in this model. After the kernel has finished its execution on the GPU, the data has to be copied back to the host memory. This approach's issue is that a kernel's memory footprint is limited by the GPU's global memory size. The application developer has to take care of data management for the application. This becomes more tedious for the developer when there are multiple GPUs.

NVIDIA introduced Unified Virtual Memory (UVM) back in 2015. UVM enabled the kernels executing on the GPU to share the same virtual address space with the process running on the host. This eliminates the memory management required before/after launching a kernel. Now, with UVM, the oversubscription of applications is also possible. Although the performance is poor as compared to manually memory-managed applications. So, to improve the performance of UVM, prefetching was introduced. When an application is not oversubscribed prefetching improves the performance significantly and makes the performance comparable to non-UVM applications. But when oversubscribed prefetching can degrade the performance. One reason for that is the eviction policy not being aware of the prefetcher [TBNP]. So, to design a prefetcher for UVM the eviction policy also has to be defined carefully. Prefetching also reduces the number of fault batches significantly which results in better performance [In-Depth Analyses of UVM]. In UVM oversubscribed scenarios, a few of the central overheads are address translation and far faults. To improve the address translation performance, huge pages can significantly improve the address translation performance.

To study the benefits of huge paging in UVM oversubscribed scenarios first, we ran an experiment that shows spatial locality behaviour of virtual 2MB (VABlocks) regions of GPGPU applications from the Rodinia benchmark suite. The memory access traces for all these applications are collected using GPGPU-Sim and the application's memory footprint is around 1 GB. The applications were over-subscribed at 110% of the memory. Figure 1(a) shows the number of base pages that are not touched in every 2MB region across applications. From the figure, we can see that almost all the base 4KB pages are touched for every 2MB region in an application's lifetime. Figure 1(b) shows how far apart all the access to a virtual 2 MB region is for some of the 2 MB regions. Usually for regular applications once a 2 MB region is touched for the first time, almost the entire 2 MB region is used briefly in the near future. From these figures, we can see that there is high spatial locality among virtual 2MB regions in GPGPU applications and huge paging can exploit that to improve the address translations performance significantly.

Second, we ran the memory access trace of the benchmarks on a memory simulator with 1 level of TLB. Both the memory and the TLB use the LRU eviction policy. Although GPUs have multi-level TLB hierarchy and the memory access comes from different SMs (NVIDIA's Streaming Multiprocessor), this experiment is to show what benefits can be reaped if huge paging is used in UVM oversubscribed scenarios. Figure (2) shows the TLB hit ratio vs the number of TLB entries with only 4KB paging and with transparent huge paging where the base pages are promoted to a huge page if all of the required 4KB base pages are present in the GPU global memory. The physical memory allocation is similar to MOSAIC, which maximizes the chance of promotion without requiring the migration of base pages in the global memory. As we can see the TLB hit ratio is better with huge paging and converges with 4KB paging when the TLB size is big enough for the

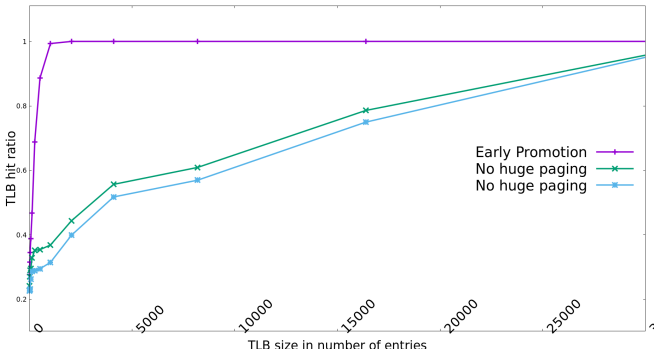

Fig. 1. TLB Hit ratio for different policies

applications footprint.

To further improve the TLB hit ratio we promote a virtual 2MB region to a huge page as soon as the first access to it comes. Not all the pages of the huge page are present in the global memory. If access comes to a region in the huge page that is not present in the global memory, then the hardware raises a far fault and the page is then transferred to the GPU global memory. Figure (3) shows the TLB hit ratio vs TLB size using early promotion. From the figure, we can see that early promotion further improves the TLB hit ratio with a greater TLB reach as compared to only 4KB paging.