Memory Management Unit and its Performance Enhancement Techniques

Divya Jamakhandi(1), Sowmya K B(2)
1,2Electronics and Communication Dept., R V College of Engineering
1divyaaj.ec16@rvce.edu.in 2sowmyakb@rvce.edu.in

Article Info
Volume 83
Page Number: 8228 - 8232
Publication Issue:
May - June 2020

Article History
***Article Received**: 19 November 2019*
***Revised**: 27 January 2020*
***Accepted**: 24 February 2020*
***Publication**: 18 May 2020*

Abstract:

Memory management unit (MMU) is a computer hardware through which address references are passed for translation of virtual memory addresses to physical addresses for effective access of physical memory. An MMU also performs memory protection, cache control and bus arbitration managing the virtual space. While the processor addresses a larger address space the actual available memory in physical may be variant. Hence it becomes important to have a system that converts the address (virtual address) provided by the processor for accessing a certain location to an address that maps it into the physical memory. While Processors are becoming faster, one major bottleneck to this increasing performance is memory access latency. MMU also performs many accesses into memory slowing down the processor. Therefore, this literature presents a detailed review and analysis of Memory Management Unit applications and various ways to improve its performance.

***Keywords:** Memory Management Unit (MMU), Virtual Address, Physical Address, Address Translation, Page Table, Memory Access.*

I. Introduction

In all systems, every processor or IO device which accesses memory needs a Memory Management Unit of its own. This MMU has a page table residing in memory which has a translation reference from virtual address to physical address. MMUs use a table called page table stored in memory which contains page table entries which are reference from in-memory table of items called a "page table", containing one "page table entry" (PTE) per page, to map virtual page numbers to physical page numbers in main memory. Page Walk is the process of converting virtual addresses to physical addresses by traversing various page tables. The page walk has high latency when compared to the processor speed, as it involves multiple memory access and using them to compute the physical address. The page walk stages, and memory access duration depends on page size. As small page size leads to many accesses to memory, it causes large latency in address translation. But the holes present in memory are usually of small size due to random allocation and deallocation of memory by the operating system. Hence the opting of large sized page sizes may give invalid spaces in memory causing high unreliability with MMU. Optimal page size is very important for better performance of MMU. Virtual address gives a continuous memory space available for the processor while the physical memory may have discontinuous memory allocation.

Memory access is one of the bottlenecks in improvement of speed of any system due to high latency in access of memory bank. A fast and optimized MMU which converts virtual address to physical address with minimum access to memory during page walk will increase the speed of the system. This is the motivation to study MMU and present the review.

II. Literature Review

The understanding of any system performance requires a detailed study of its memory access pattern. Therefore, it is important to study the efficiency and memory management unit performance and various factors that affect it. One such study and improvement technique is proposed by Vasileios Karakostas and team in the paper [1]. This literature states in numbers the MMU overhead for real time scale out applications that use performance counters. The numbers state that about 16% of execution time which are a result of miss rates occurring in TLB and clashes between page walks and data access in the cache. It gives proof that large pages may improve MMU overheads and performance of application by about 13.9%. Due to low memory locality even when large pages are enabled the performance may not improve drastically. About 3.8% performance gap is seen with increased page size. But there is high scope for improvement, as even with large pages the MMU overheads are still very high.

The work by Antoine Faravelon and team in the research [2], gives an Optimization technique in which the translation technique being used is Dynamic Binary Translation which optimizes the memory accesses done by MMU during the Page table walk making it efficient. As processor speed is largely slowed down by memory access, Dynamic Memory Translation proves to be increasing the simulation of SOCs to greater extent as it hits at this bottleneck. A hardware MMU based emulation in software is needed for every load and store issued by the processor. In recent times this software emulation is performed on hardware and used the capabilities of hardware-based virtualization. For running the entire simulation on virtual CPU, firstly a hardware-based shadow page tables are setup are like any other usual hypervisor. The second step is the compression of multiple load and store instruction to a smaller number, to avoid emulation overheads. The work also explains in detail how this technique is implemented to improve the memory access efficiency by MMU implemented by a DBT engine speeding up the performance by about 40% by changing the translation technique.

The Research by Xiantao Zhang and team [3] presents a Hash Translation Lookaside Buffer technique for MMU optimization and virtualization. Virtualization is a technique in which the access to hardware resources which are expensive is shared using virtualization techniques assisted completely by hardware. Virtual machine monitor that uses system virtualization is observed to be a great optimization technique. Firstly, the virtual hash table-based implementation in TLB is studied. The studies show that this technique can improve virtual system performance by a margin of about 5%. Th implementation of this work starts with the design of hash TLB algorithm and its optimization on guest virtual machines. Second, is to study the performance benefits of hash TLB approach and its effect on the walker. This research also extends its study into the scalability of this approach for implementation in various systems. The work implements two techniques, the Single table-based VT and Hash table-based VT each giving a performance improvement of about 0.42 and 3.66% respectively. Figure 1 gives the comparison of Single Table VT and Hash table VT for different systems. For kernel build which have high disk reads and writes a maximum performance improvement of 44.34% is observed.

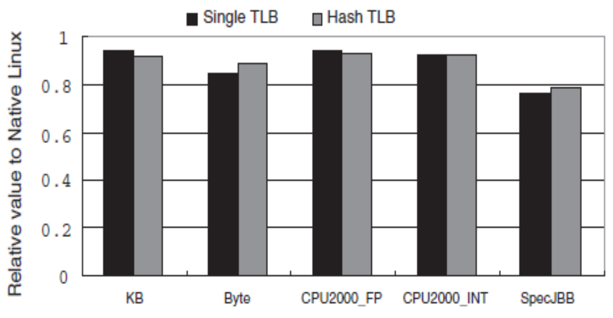

Figure 1: Performance Graphs of Hash TLB Implementation.[3]

Memory is one of the vital components of a Processor. The design of memory system for a multicore processor is of great importance as the high speed achieved by the addition of additional processors should not be bottlenecked by the memory system. The literature by Jianjun Guo and team about the work [4] gives design methodology for an efficient and optimized memory system design as shown in Figure 2. Multi-core processor design has turned into a large research area in recent times. Cache are highly expensive memory storage units, and not fit to be used to store multiple copies of data in any processor. The MMU efficiency is largely affected by memory architecture design and this research concentrates on design of a hybrid and efficient memory hierarchy. This splits the locality of instruction and data storage by use of hierarchical cache. This system includes various levels of storage elements which include L1 instruction cache, local data storage, DMA engine, L2 cache. The MMU is designed accordingly as per the levels of memory and optimized. One more optimization step is the replacement strategies used at L2 Cache Level to reduce miss rate.

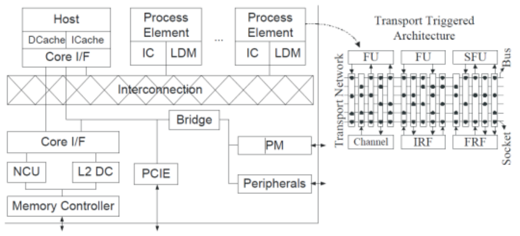

Figure 2: Memory System for Multicore Processor.[4]

The research work by Yonghwan Lee and team [5] proposes a technique in which the MMU and Cache share a common shared tag. Both the area of memory and speed of cache system that used convention tag system are improved using this shared tag based architecture. The proposed architecture shown in Figure 3 is validated by simulations and VLSI circuits are used to estimate its speed and area. Two tag memories and cache memories are concurrently accessed using the virtual tag of TLB. For a hit occurring with the virtual tag in TLB, a physical page is read from memory into the TLB. The cache hit is decided based on the comparison between the physical page numbers and cache tag that have been already read. On cache hit the processor is returned with the requested data. Using this technique, a maximum of 8% reduction of miss rates is achieved.

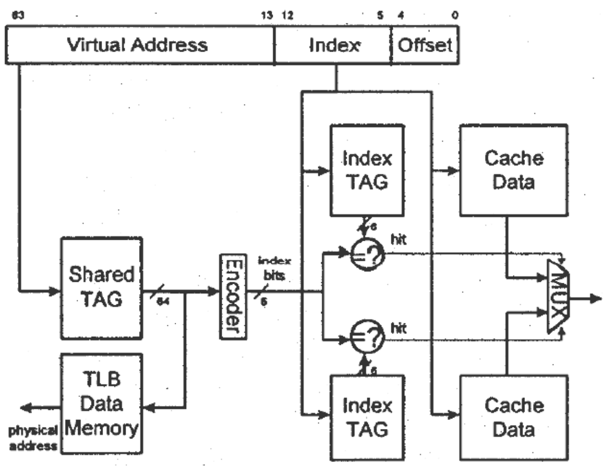

Figure 3: Creation of Tagged address space.[5]

The Literature [6] by Jae Young Hur provides a contiguity-based page table method to improve performance Memory Management Unit. The major overheads in conventional pages is the page table walks which affect the system performance. This scheme introduces a method to represent contiguity in pages that can improves performance by reducing the number of page table walks. Certain physical memory used by the system is allocated continuously without any holes. Hence this continuous page can be resolved with a smaller number of walks. The conventional page method and contiguity-based method are compared to get performance numbers. The Figure 4 shows the implementation of the new page table where new bit is added to represent the contiguity and the continues pages lead to single walk leading the improvement in efficiency. For 100% implementation of Contiguity matrix a performance improvement of 10-50% can be got for different access patterns with minimum of 10% at least.

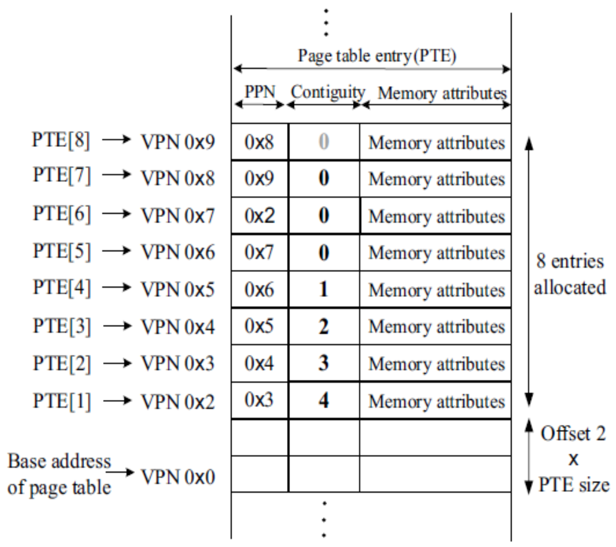
Page table entry(PTE)
PPN
Contiguity
Memory attributes
PTE[8] → VPN 0x9 | 0x8 | 0 | Memory attributes
PTE[7] → VPN 0x8 | 0x9 | 0 | Memory attributes
PTE[6] → VPN 0x7 | 0x2 | 0 | Memory attributes
PTE[5] → VPN 0x6 | 0x7 | 0 | Memory attributes
PTE[4] → VPN 0x5 | 0x6 | 1 | Memory attributes
PTE[3] → VPN 0x4 | 0x5 | 2 | Memory attributes
PTE[2] → VPN 0x3 | 0x4 | 3 | Memory attributes
PTE[1] → VPN 0x2 | 0x3 | 4 | Memory attributes
8 entries allocated
Offset 2 x PTE size
Base address of page table → VPN 0x0

Figure 4: Contiguity Page Tables.[6]

The literature [7] by Robert Witting and team propose an efficient Queue based Memory Management Unit which tries to combine the flexibility of traditional methods with the low latency property of tightly coupled memory. The major idea is to reduce critical path by the conflict detection method without disturbing the flexibility of the dynamic system of memory allocation and data widths that are heterogeneous. To implement this hybrid a queue-based memory controller is used. This design of Q-MMU is such that it pipelines all masters to have same delays in access. Figure 5 shows the architecture of the QMMU system. This connect gives an improvement of over 20% than the AXI-interface in critical path, which can increase up to 60% in other paths.

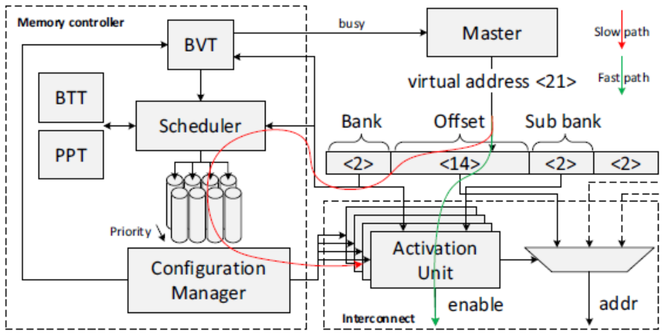

Figure 5: Q-MMU Architecture.[7]

III. Inference

The literatures above show the various roles that MMU plays in the system and its importance for any system performance. The literatures studied above have opted several different methods to improve efficiency of MMU unit. The literature [1] gives the effect of page size on performance of MMU and how ideal design between page size and performance needs to be designed based on analysis of design requirements. Other optimization techniques include Binary Translation Tables[2], Hash-TLB Approach[3], Shared tag based MMU and Cache implementation[5], Coalesced and Shared Memory Management Unit Caches for the purpose of accelerating TLB Miss handling[9] and adaptive memory allocation for better page table allocation[8]. The improvements in performance provided by these implementations are compile in the Table below in Table 1.

Table 1: Performance Comparison of different implementations.

Sl. No.	Method of Implementation	Percentage Performance improvement.
1.	Large Page Size usage.	16
2.	Dynamic Binary Translation	40
3.	Hash TLB	44
4.	Shared Tag based MMU	8
5.	Contiguity Based Page Table.	10
6.	Queue Based MMU	20

IV. Conclusion

Memory Management unit is a computer hardware which converts virtual address into physical address. It functions as interface for both the processor and IO devices to the memory. As memory access is one of the major bottlenecks in system performance enhancement, this review looks at the various ways in which the MMU operation can be optimized. Faster walks, lesser memory access and elimination of memory access during walks are performed in the various literatures reviewed in this survey. The review presents a comprehensive look at the MMU functionality and improvement techniques and their impact on the system performance. It helps to choose among the various implementations based on their performance enhancement.

References

[1] V. Karakostas, O. S. Unsal, M. Nemirovsky, A. Cristal and M. Swift, "Performance analysis of the memory management unit under scale-out workloads," *2014 IEEE International Symposium on Workload Characterization (IISWC)*, Raleigh, NC, 2014, pp. 1-12.

[2] A. Faravelon, O. Gruber and F. Pétrot, "Optimizing Memory Access Performance Using Hardware Assisted Virtualization in Retargetable Dynamic Binary Translation," *2017 Euromicro Conference on Digital System Design (DSD)*, Vienna, 2017, pp. 40-46.

[3] Xiantao Zhang, A. X. F. Xu, Qi Li, D. K. Y. Yau, Sihan Qing and Huanguo Zhang, "A hash-TLB approach for MMU virtualization in xen/IA64," *2008 IEEE International Symposium on Parallel and Distributed Processing*, Miami, FL, 2008, pp. 1-8.

[4] J. Guo *et al.*, "Memory System Design for a Multi-core Processor," *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, Barcelona, 2008, pp. 601-606.

[5] Yonghwan Lee, Wookyung Jeong, Sangjun Ahn and Yongsurk Lee, "Shared tag for MMU and cache memory," *1997 International Semiconductor Conference 20th Edition. CAS*

'97 Proceedings, Sinaia, Romania, 1997, pp. 77-80 vol.1.

[6] J. Y. Hur, "Representing Contiguity in Page Table for Memory Management Units," *2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Seoul, 2017, pp. 21-28.

[7] R. Wittig, M. Hasler, E. Matus and G. Fettweis, "Queue Based Memory Management Unit for Heterogeneous MPSoCs," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 1297-1300.

[8] I. Deligiannis and G. Kornaros, "Adaptive memory management scheme for MMU-less embedded systems," *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, Krakow, 2016, pp. 1-8.

[9] A. Bhattacharjee, "Large-reach memory management unit caches: Coalesced and shared memory management unit caches to accelerate TLB miss handling," *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Davis, CA, 2013, pp. 383-394.

[10] A. Khaled and Q. Zhang, "An Energy Aware Mass Memory Unit for Small Satellites Using Hybrid Architecture," *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Guangzhou, 2017, pp. 210-213.

[11] Y. Hao, Z. Fang, G. Reinman and J. Cong, "Supporting Address Translation for Accelerator-Centric Architectures," *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Austin, TX, 2017, pp. 37-48.

[12] J. Y. Hur, "Representing Contiguity in Page Table for Memory Management Units," *2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Seoul, 2017, pp. 21-28.

[13] N. R. Saxena, C. -. D. Chang, K. Dawallu, J. Kohli and P. Helland, "Fault-tolerant features in the HaL memory management unit," in *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 170-180, Feb. 1995.

[14] K. Vimal and A. Trivedi, "A memory management scheme for enhancing performance of applications on Android," *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, Trivandrum, 2015, pp. 162-166.

[15] F. Shamani, V. F. Sevom, J. Nurmi and T. Ahonen, "Design, implementation and analysis of a run-time configurable Memory Management Unit on FPGA," *2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*, Oslo, 2015, pp. 1-8.