Exploration of the Feasibility of the VIVT Caches

R&D Monthly Report - January

Submitted in partial fulfillment of the requirements

of

Research and Development

by

Akshay Verma

(Roll No. 200070005)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering
Indian Institute of Technology Bombay

January 2023

Acknowledgement

I express my gratitude to my guide Prof. Virendra Singh for providing me the opportunity to work on this topic.

Akshay Verma
Electrical Engineering
IIT Bombay

Abstract

To ease the programmers while writing the code without caring about the size of program code and data exceeding the size of the main memory and data privacy, the concept of virtual memory was developed. This automated the movement of program code and data between the main memory and secondary memory giving an appearance of single large memory.

Implementation of this concept alongside the usage of cache memory lead to different cache structures with different efficiencies and protections. This also increases many coherence problems for the cache hierarchy hence creating issue with programmability.

Contents

List of Figures

Chapter 1

Introduction

1.1 Virtual Memory

There are several limitations to the usage of Physical Memory like fragmentation issue causing a great limit to number of processes running simultaneously, privacy and security of data of one process from other is compromised and crashing of processes due to insufficient memory size. To overcome these we use the concept of virtual memory and Virtual Address Space.
Virtual memory and caches are two of the most common elements of computers today. Most modern computer systems take advantage of fast caches to increase the performance of a processor, and virtual memory provides access protection, an illusion of large physical address space, efficient memory management, and support of data sharing.
A processor generates virtual addresses which should be then translated to the corresponding physical addresses to access the appropriate data in caches or memory. Operating system (OS) page tables maintain virtual-to-physical address translations. However, accessing the OS page table on every memory access is not latency and energy efficient because it requires multiple memory accesses. Hence, an efficient address translation is necessary to support virtual memory.
These overheads are overcome by using the Translation Lookaside Buffer, or TLB. The Memory Management Unit, or MMU, keeps track of the recently used translation from the OS Page Table and stores in a small cache known as TLB. The OS Page Tables are accessed only in the case of TLB misses. Modern computers use multi-level TLBs to improve the translation overhead.

1.2 Cache Structures

Since use of cache is also for increasing the performance and concept virtual memory is also for the same, so some cache structures[1] were derived using both concepts. The different possible interactions of TLB with Cache and the location for Virtual to Physical address translation lead to these cache structures.

- **Physically Indexed Physically Tagged:** Also known as PIPT caches, uses only the physical address for both the tag and index to access the cache. Hence the Virtual to Physical address translation is to be done, via TLB, before the cache access. This adds TLB access latency to the cache access hence reducing the efficiency. This is just a basic serial model using both the concepts and is very inefficient.

- **Physically Indexed Virtually Tagged:** Also known as PIVT caches, uses the index of the Physical address and tag of the Virtual address to access the cache. Here the tag comparison of the accessed data using index is to be done hence Virtual to Physical address translation is to be done before the cache access. In case of tag match its a hit else its a miss. Since TLB is accessed prior to cache hence the overheads in this case is similar to that in the case of PIPT, hence very inefficient.

- **Virtually Indexed Physically Tagged:** Also known as VIPT, uses the index of the Virtual address and the tag of the Physical address to access the cache. Since it uses Virtual tags so TLB access is not needed prior to cache access, instead both are done parallelly, hence the latency overhead by TLB access is not seen as compared to physically indexed caches. However these still require TLB lookup for every cache access and hence is very energy inefficient. Due to indexing with virtual addresses, it is possible for the same data be cached with different virtual addresses in different sets in the cache. This data duplication issue is known as *synonyms* which is a problem for every Virtually indexed cache.

- **Virtually Indexed Virtually Tagged:** Also known as VIVT caches, uses only the Virtual address to access the cache. No TLB access until a cache miss, hence latency overhead is similar to that of VIPT also making it more energy efficient compared to VIPT due to less TLB accesses. These also have synonyms like VIPT.

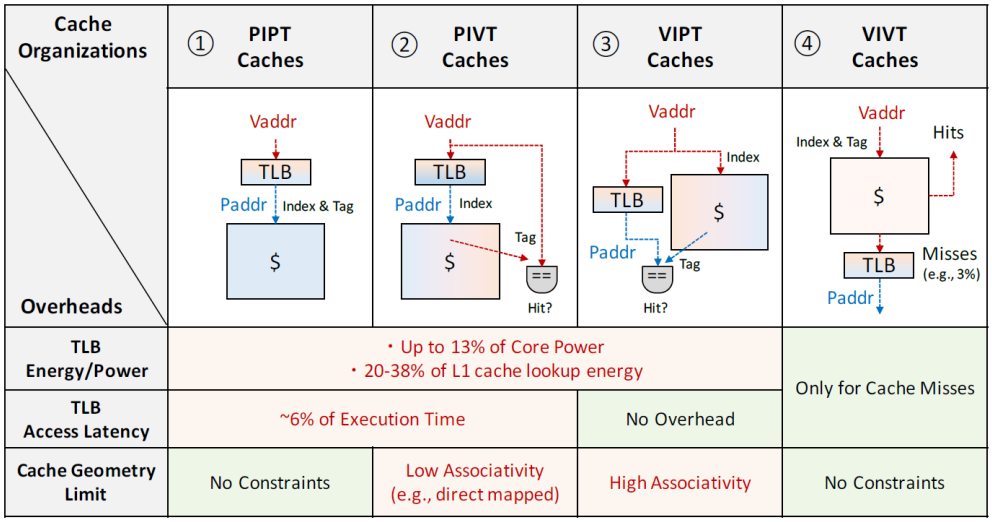

Figure 1.1: Types of Cache Organisations

Although the best one among them is VIVT but it has issues with Programmability like Synonyms issues. There are several methods for minimizing the issue of synonyms like using Address Space ID (ASID), Midgard, etc. But this project will be looking into usefulness of using Midgard to overcome this issue. I will be analyzing this by implementing the Midgard abstraction into *Gem5* simulator and compare standard workloads for both configurations, with and without Midgard.

Chapter 2

Literature Survey

2.1 Rebooting Virtual Memory with Midgard

To work with modern big-data workloads computer architectures are having higher and higher capacity cache hierarchy which obviously improves the system performance but shifts the performance bottleneck to Virtual-to-Physical address translation. Along with this higher capacity cache hierarchies require complex address translation hardware support resulting in larger on-chip area and sophisticated heuristics. More sophisticated coherence protocols in the Operating System (OS) are also required due to increased number of per-core TLBs and MMU which are slow and buggy. One of the method to overcome these issues is using a VIVT-Cache, but it has issues with programmability like synonyms and homonyms. The authors provided a proposal to overcome these issues as well as improve the translation overhead using another address space between the Physical and Virtual address spaces known as Midgard Address Space.

Chapter 3

Proposal

The proposal is to create an intermediate address space in between Virtual Address Space and Physical Address Space known as Midgard Address Space[2]. This address space is created so that VMAs of different processes are uniquely mapped. This address space works as the namespace for all the data in coherence domain and cache hierarchies hence the programmability is also increased. Here all cache access are to be made by Virtual-to-Midgrad address translation which is not too expensive as there are very few VMAs as compared to number of pages in real-world workloads.

3.1 Key Ideas

- Midgard enables the placement of the coherent cache hierarchy in a namespace that offers the programmability of traditional VM.
- Midgard quickly translates from virtual addresses to this namespace, permitting access control checks along the way and requiring significantly fewer hardware resources than modern per-core TLBs and MMU cache hierarchies.
- Translating between Midgard addresses and physical addresses requires only modest augmentation of modern OSes. Midgard filters heavyweight translation to physical addresses to only those memory references that miss in the LLC. Sizing LLCs to capture workload working sets also naturally enables better performance than traditional address translation.

3.2 Hardware and Software Support

- **Software Support (in OS):**
    1. A data structure known as Virtual Memory Area (VMA) Table is added to OS which stores mapping between Virtual and Midgard Address Spaces for V2M translation. This is a B-Tree data structure.
    2. Another data structure know as Midgard Page Table (MPT) is also added to OS which stores mapping between Midgard and Physical Address Spaces for M2P translation.This is a Radix Page Table structure.

- **Hardware Support (via Registers):**

    1. A register is used as a pointer to the root node of VMA and Midgard Page Table.
    2. In case of M2P translation failure to permit the rollback to store buffer (for OoO CPUs) mappings before translation attempt they use some registers which they haven't mentioned and will do in their future work.
    3. The Midgard Page Table entries (like conventional page tables) track access and dirty bits to identify recently used or modified pages. While access bits in TLB-based systems can be updated on a memory access, modern platforms opt for setting the access bit only upon a page walk and a TLB entry fill. Midgard updates the access bit upon an LLC cache block fill and the corresponding page walk.
    4. Using Midgard Lookaside Buffer (MLB) to accelerate M2P translation. This buffer caches frequently used entries from the Midgard Page Table with a mapping, access control information and access/dirty bits. This is optional and better for only small LLC size (<32MB).

The two step translation can be understood by the following flowchart:

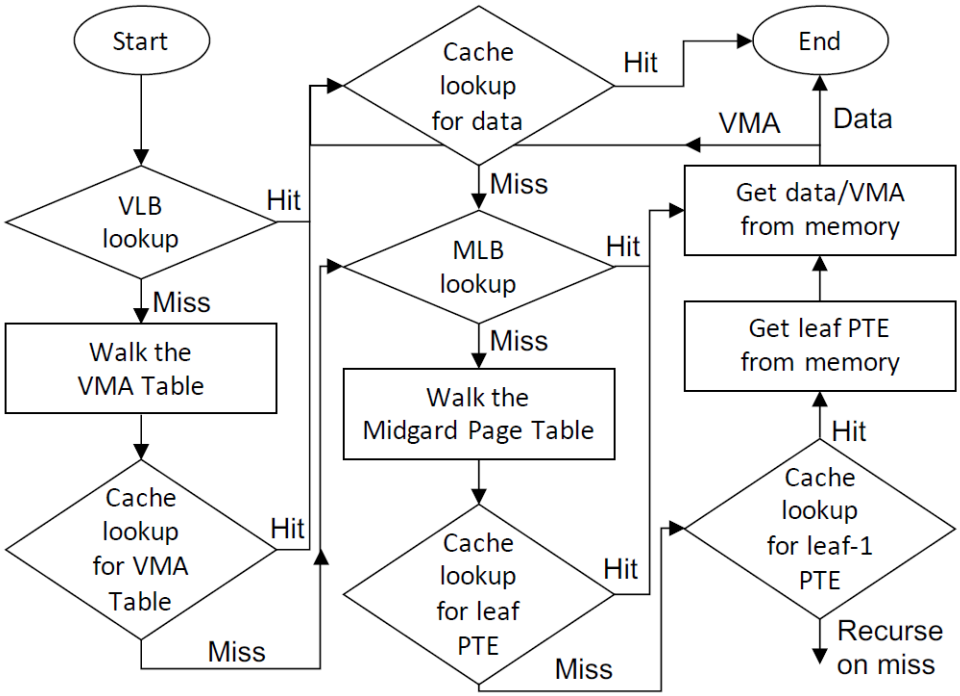

Figure 3.1: Two Step Translation

Chapter 4

Conclusion

4.1 Work done so far

- Read about the concept of Virtual Address Space and Virtual Memory and its benefits over the Physical Memory.
- Learnt how to use Gem5 simulator and for the final goal went through its OOP Model to search for the function responsible for Virtual to Physical address translation. Read the function and understood how the address translation is done by Gem5 and how are the miss and hit values updated for a simulation.
- Ran a few simulations of Gem5 using System Call Mode for Atomic and Derive O3 (Detailed) CPUs for both some standard workload benchmarks and my own simple programs.
- Read and presented the paper on Midgard from 48th Annual International Symposium on Computer Architecture.

4.2 Next month's target

- Modify the translation function to print the Virtual Address and the Physical Address during the translation and hence find the details about the Page Table Structure like no. of PTE per cache, level of each PTE, etc.
- Simulated standard workload benchmarks in full system simulation mode by creating a Linux OS and a few simple programs and observed the statistics of the simulation.
- Start implementation of Midgard Address Space in Gem5 Simulator.

4.3 Final goal

To be able to use VIVT Caches.

References

[1] H. Yoon, *Reducing address translation overheads with virtual caching*. The University of Wisconsin-Madison, 2017.

[2] S. Gupta, A. Bhattacharyya, Y. Oh, A. Bhattacharjee, B. Falsafi, and M. Payer, "Rebooting virtual memory with midgard," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 512–525, IEEE, 2021.