VIRTUAL MEMORY

Summer Project Stage 1 Report

Submitted in partial fulfillment of the requirements

of

Summer Project

by

Akshay Verma

(Roll No. 200070005)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering
Indian Institute of Technology Bombay
August 2022

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

XX.

Contents

List of Figures

Chapter 1

Introduction

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. Since we use cache for increasing the performance and concept virtual memory is also for the same we derived some cache structures using both concepts *wiz.*,

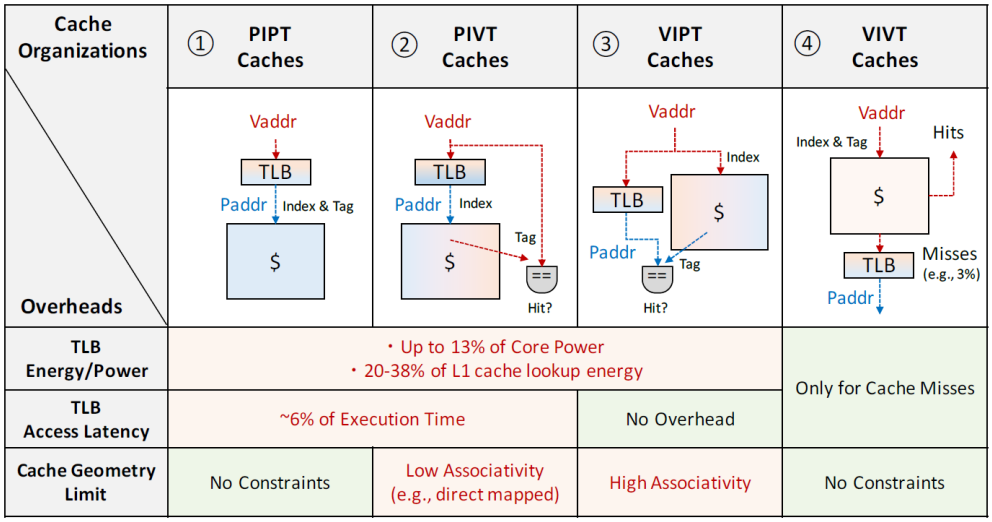

Figure 1.1: [1]Types of Cache Organisations

although the best one among them is VIVT but it has issues with Programmability like Synonyms and Homonyms. The basic aim of this project is to implement something to overcome this issue.

1.1 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.

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 paper provided a proposal to overcome these issues too.

2.1.1 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 with is not too expensive as there are very few VMAs as compared to number of pages in real-world workloads. The three pillars on which this proposal is based are:

- 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.

2.1.2 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:

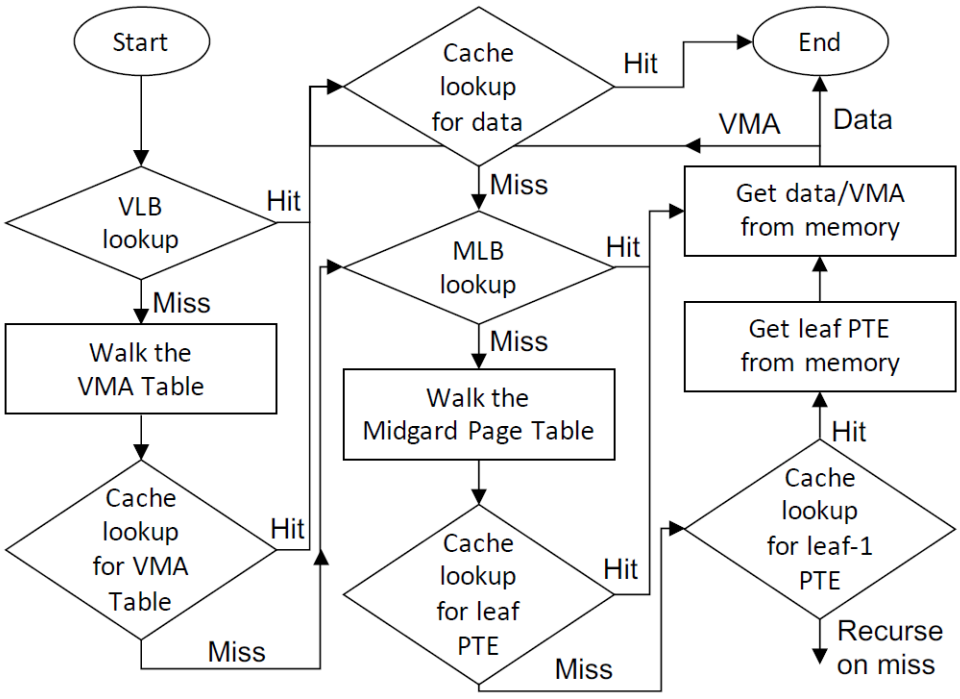

Figure 2.1: Two Step Translation

2.1.3 System Parameters

Core	16 × ARM Cortex-A76
Traditional TLB	L1(I,D): 48 entries, fully associative, 1 cycle Shared L2: 1024 entries, 4-way, 3 cycles
L1 Caches	64KB 4-way L1(I,D), 64-byte blocks, 4 cycles (tag+data)
LLC	1MB/tile, 16-way, 30 cycles, non-inclusive
Memory	256GB capacity (16GB per core) 4 memory controllers at mesh corners
Midgard	VLB: L1(I,D): 48 entries, fully associative, 1 cycle L2 (VMA-based VLB): 16 VMA entries, 3 cycles

2.1.4 Results

	Dataset Size (GB)				Thread Count					
	0.2	0.5	1	2	4	8	12	16	24	32
BFS SSSP	51	51	52	52	52	60	68	76	84	108

Table 2.1: VMA Usage Characterization

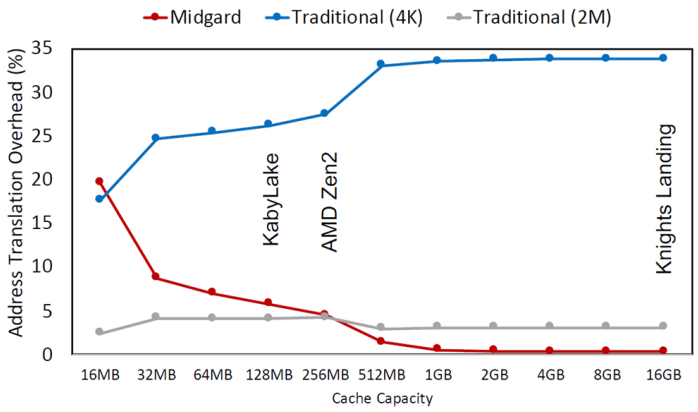

Figure 2.2: Percentage AMAT Spent in Translation

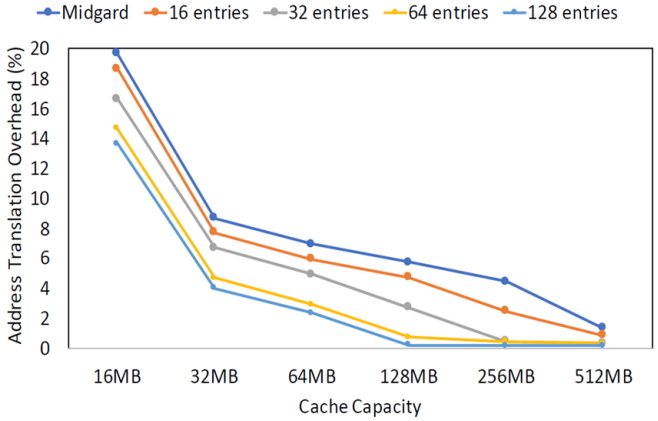

Figure 2.3: Required MLB size as a function of LLC Capacity

Chapter 3

Future goals

3.1 Next month's target

- Simulate some standard workload benchmarks in full system simulation mode by creating a Linux OS and also some of my simple programs and observe the statistics of the simulation.
- 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.
- Start implementation of Midgard Address Space in Gem5 Simulator.

3.2 Final goal

Implement the abstraction of Midgard into Gem5 and compare the performance with Traditional and Virtual Cache Hierarchy for modern-day workloads.

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.