Diploma Project

OBSERVING THE EFFECTS OF NUMA PAGE MIGRATIONS BETWEEN MEMORY TIERS IN CXL PLATFORMS

Paul Adrian Seica

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

OBSERVING THE EFFECTS OF NUMA PAGE MIGRATIONS BETWEEN MEMORY TIERS IN CXL PLATFORMS

Paul Adrian Seica

Advisor:

Dr Haris Volos

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

Acknowledgements

I would like to sincerely thank my girlfriend Angelina, for putting up with me throughout the course of this thesis and all the hours spent working on it.

I love you to the moon and back, and dedicate the efforts of this thesis to you.

ABSTRACT

As data center memory demands continue to grow, Compute Express Link (CXL) has emerged as a promising technology for expanding memory capacity beyond what traditional DRAM alone can provide. CXL Type 3 devices attach additional byteaddressable memory to a host processor and are exposed to the operating system as slower NUMA nodes, making them natural candidates for the cold tier in a tiered memory system. When multiple such devices are configured within a single interleaved memory region, a question arises that has not been well studied: does the Linux kernel's automatic demotion policy distribute cold pages across all devices uniformly, or does it concentrate them on one device at a time? This thesis investigates that question with a focus on energy proportionality. A virtual machine is configured with two identical CXL Type 3 devices, each contributing 1 GB of capacity, interleaved into a single 2 GB NUMA node. A custom block of code is added in the Linux Kernel's migration function, logging the physical source and destination addresses, and the migration reason for every page demotion and promotion event. Three workloads from the GAP benchmark suite, PageRank, Breadth-First Search, and Betweenness Centrality, are used to generate realistic memory pressure and trigger migrations.

Keywords: CXL, Memory Tiering, NUMA, Page Migration

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Motivation

Modern data centers are under increasing pressure to support larger memory footprints. Workloads like graph analytics, in-memory databases, and machine learning inference routinely require more memory than a single server can provide with DRAM alone. Populating a server with enough DRAM to meet these demands is expensive, and at some point, it becomes physically impractical. Operators are often forced to choose between overprovisioning hardware or accepting performance hits when memory runs out.

Compute Express Link (CXL) has become one of the more promising responses to this problem. It is an interconnect standard built on top of PCIe that allows additional byteaddressable memory devices to be attached to a host CPU and used as regular system memory. CXL memory is slower than local DRAM, adding roughly 50 to 100 nanoseconds of extra latency per access, but it costs less per gigabyte and can be scaled independently of the CPU. Companies like Meta have already started using CXL-based memory in production systems [2], and CXL 3.0 devices are now available from several hardware vendors.

The Linux kernel treats CXL memory devices as additional NUMA nodes without CPUs, fitting them into the existing NUMA balancing infrastructure as a slower memory tier. Pages that have not been accessed recently are demoted from DRAM to CXL memory, and pages that are accessed again while sitting in CXL memory are promoted back up to DRAM. This behavior has been part of the mainline kernel since version 5.18 and requires no changes to applications [2]. However, the details of how the kernel makes these decisions are not easy to observe. There is no standard tool that shows where exactly within a CXL memory region demoted pages end up.

This becomes a more interesting question when multiple CXL devices are involved. A CXL switch can connect two or more Type 3 memory devices and present them to the operating system as a single interleaved memory region under one NUMA node. The kernel allocates pages into this region without knowing which physical device belongs to any given address. Whether demoted pages end up concentrated on one device or spread evenly across both has a direct effect on energy efficiency. Memory devices consume power even when idle, due to refresh cycles and standby circuitry. If cold pages are concentrated on one CXL device, the other can sit at low utilization and potentially enter a reduced-power state. If both devices receive pages from the start, neither gets the chance to idle, and any potential power saving is lost. This is the statement this thesis wants to observe.

1.2 Objectives

This thesis looks at how the Linux NUMA page migration mechanism behaves in a two-tier CXL environment, and whether the page placement patterns it produces are consistent with energy-proportional memory use, more specifically, whether the pages will be concentrated in such a way so as to allow the enabling of power-saving mechanisms.

The first objective is to observe kernel demotion decisions. Knowing which addresses within an interleaved CXL region receive demoted pages, and which device those addresses belong to, is necessary before any conclusions about energy proportionality can be drawn. At the moment, the Linux kernel provides no built-in way to track this.

The second objective is to assess whether the existing Linux demotion policy produces energy-proportional placement. If pages are consistently sent to one device before the other, then the current software behavior already supports potential power savings without any modifications. If both devices fill up in parallel, it points to a concrete gap that future work would need to address, whether at the kernel allocator level or through different hardware interleaving settings.

The third objective is to test this across multiple workloads and repeated runs. Graph processing benchmarks place heavy pressure on memory and are representative of the kinds of applications that motivate tiered memory in the first place. Using several benchmarks and running each multiple times makes it possible to tell consistent patterns apart from random variation and gives a clearer picture of how the kernel behaves in practice.

By completing these objectives, we can get a clearer idea into how interleaved CXL devices are utilized.

1.3 Contributions

This thesis makes the following contributions:

1. A tool developed and integrated into the Linux kernel's page migration function that records the physical source address, physical destination address, and migration reason for every demotion and promotion event. This provides a level of detail about page migration behavior that is not available through any existing kernel interface.
2. An empirical study of Linux demotion behavior in a configuration with two interleaved CXL devices, tested across three graph benchmarks from the GAP benchmark suite.
3. An energy proportionality analysis of the observed placement patterns, showing whether the current Linux policy favors one CXL device over the other or whether it splits demotions across them.

4. A cross-run analysis using Jaccard similarity to measure how much the sets of source and destination pages overlap between runs.

2 Background

2.1 NUMA Architecture

Modern servers are rarely built around a single processor and a single block of memory. Instead, they are organized using a design called Non-Uniform Memory Access (NUMA), where multiple processors, each with their own local memory, are connected through a high-speed interconnect. The term ”non-uniform” captures the core idea: not all memory is equally fast. A processor can access its own local memory significantly faster than it can reach memory belonging to another processor node.

In a NUMA system, the hardware is logically divided into nodes. Each node contains one or more CPU cores along with a portion of the system's physical memory. Accessing memory within the same node, referred to as a local access, is fast and suffers low latency. Accessing memory that belongs to a different node, referred to as a remote access, must travel through an interconnect such as Intel Ultra Path Interconnect or AMD Infinity Fabric. This cross-node traffic introduces additional latency and can create bandwidth contention, especially in systems with many nodes [3].

The performance gap between local and remote accesses is what motivates most NUMAaware software design. Operating systems attempt to allocate memory on the same node that a running process belongs to, a policy known as first-touch allocation [4]. Under this policy, a page is placed on the node that first writes to it. This works well for simple singlethreaded programs but can become suboptimal in multi-threaded scenarios, where different threads may be scheduled across different nodes, causing some of them to always read from remote memory [3].

To address this over time, the Linux kernel introduced the Automatic NUMA Balancing mechanism, which periodically observes which processes are accessing which pages and gradually migrates pages toward the node that accesses them most frequently [5]. This mechanism is discussed in more detail in Section 2.4.

From the kernel's point of view, each NUMA node is represented as a memory zone with its own allocator, free list, and set of CPUs. The kernel attempts to satisfy memory requests from the local zone first. If local memory is insufficient, it falls back to zones on other nodes in order of NUMA distance, which is a value configured in firmware that describes how expensive cross-node accesses are [6].

Understanding NUMA is important for this thesis because CXL memory devices are exposed to the operating system as additional CPU NUMA nodes. This means the same kernel mechanisms used to manage locality and page migration in a traditional multisocket system apply directly to

how pages move between DRAM and CXL memory. In this thesis, the local zone will be the local DRAM, and the fallback zone being the CXL memory region.

2.2 Memory Tiering

As systems grow in size and workloads become more memory-intensive, the traditional model of having a single flat pool of uniform DRAM is increasingly impractical. The emerging alternative is tiered memory, where the system contains multiple types of memory with different performance and cost characteristics.

In a two-tier system, the fast tier is typically local DRAM, which offers low latency and high bandwidth. The slow tier consists of a secondary memory medium that provides more capacity at a lower cost per gigabyte, but at the expense of higher access latency. Examples of slow-tier memory include the discontinued Intel Optane persistent memory [7], high-bandwidth memory attached over CXL [2], and disaggregated memory pools accessible over a fabric [2, 8].

The fundamental goal of memory tiering is to maximize the fraction of memory accesses that land in the fast tier. This is achieved through page migration: pages that are accessed frequently, called hot pages, should reside in DRAM, while pages that have not been accessed recently, called cold pages, can be moved down to the slow tier. When a page is demoted, it is removed from the active LRU list and placed onto the inactive LRU list. When it comes to promotions, the first time that same page is accessed from the cold tier, the kernel moves it from the inactive LRU list back to the active LRU list. Then, if that page is accessed a second time while in the active LRU list but in the cold tier, it gets promoted to the hot tier [8]. The two-list design is necessary because a single LRU list cannot distinguish between pages accessed once and pages accessed repeatedly. Without this separation, a sequential scan could evict genuinely hot pages.

The kernel's tiering support builds on existing NUMA balancing mechanisms. Cold DRAM pages are demoted to the slow tier when the fast tier is under memory pressure. This demotion was introduced in Linux 5.15 via the commit that enables page migration during memory reclaim [8]. Promotion of hot pages from the slow tier is handled by the NUMA hint-faulting mechanism, described in the next section.

A key challenge in tiered memory management is the definition of hot and cold pages. Pages that were frequently accessed yesterday might be irrelevant today if the workload changed. Choosing an appropriate hotness threshold is therefore non-trivial. Research systems such as AutoTiering [7], TPP [2], and HybridTier [9] have each taken different approaches, including hardware-assisted monitoring, NUMA hint-fault latency measurement, and lightweight profiling. These are discussed further in Chapter 3.

For this thesis, the tiering configuration is imposed by the virtual machine setup. DRAM forms

the fast tier, and CXL memory, exposed as a separate CPU-less NUMA node, forms the slow tier. The Linux kernel manages demotion and promotion automatically, and the goal of this thesis is to observe how that management plays out in terms of which physical addresses within the CXL node receive demoted pages.

2.3 CXL Memory Devices

Compute Express Link (CXL) is an open industry interconnect standard built on top of the PCI Express physical layer. It was first released in 2019 and has since gone through multiple generations, with CXL 1.0, 2.0, and 3.0 each adding new capabilities [10]. CXL's primary purpose is to enable high-speed, cache-coherent communication between a host CPU and attached devices such as accelerators, memory expanders, and compute devices.

The specification defines three types of devices. Type 1 devices are accelerators that can cache host memory but have no device-local memory. Type 2 devices are accelerators that both cache host memory and have their own device memory. Type 3 devices, which are the relevant type for this thesis, act purely as memory expanders. They attach additional byteaddressable memory to the host and expose it through the CXL.mem protocol, which allows the CPU to access device memory using standard load and store instructions.

A key property of CXL Type 3 devices is that they appear to the operating system as Hostmanaged Device Memory (HDM). This memory is integrated into the physical address space of the host and can be used as regular system RAM, with the OS deciding how to partition it between applications and kernel use. CXL memory access latency is higher than local DRAM, typically adding 50 to 100 nanoseconds of additional delay due to the PCIe traversal and the external memory controller [2, 9]. This latency overhead is what motivates placing hot pages in DRAM and cold pages in CXL.

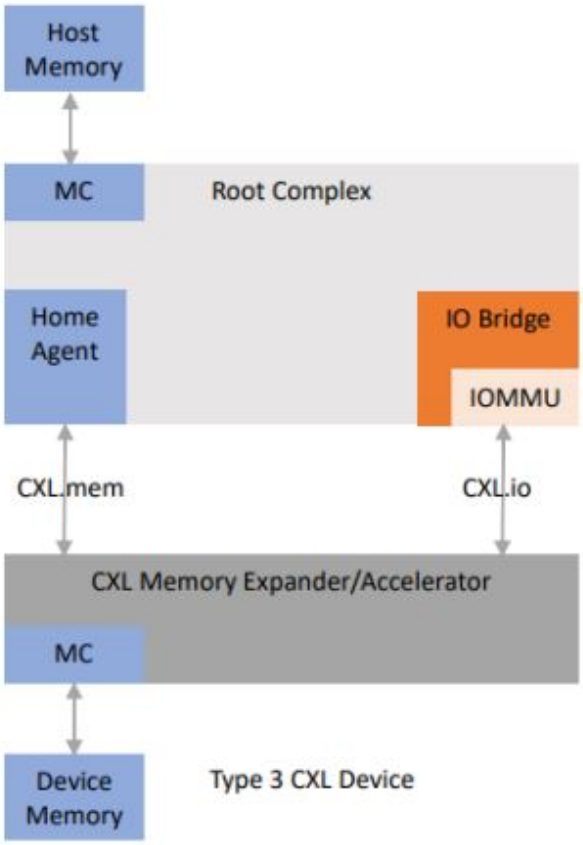

Figure 2.1: Example CXL hierarchy with one Type 3 device, thus not requiring a CXL switch [1]

CXL follows the hierarchy shown in Figure 4.1 [1]:

1. **Root Complex**

   The Root Complex is the silicon block inside the CPU that owns and orchestrates all CXL/PCIe traffic. It contains the CPU cores, the DDR memory controller, and the CXL controller, which includes the routing tables that map Host Physical Addresses (HPAs) to CXL destinations.

2. **Host Bridge**

   The Host Bridge is the logical boundary between the CPU's internal bus and the external CXL fabric. It is where HDM decoders reside. When the CPU issues a load to a CXL address, the HDM decoder determines which downstream Root Port and device own that address range and routes the request accordingly.

3. **Root Port**

   A Root Port is a circuit integrated into the CPU that represents one end of a CXL link. It forms the connection between the Host Bridge and a CXL Switch. Its role is to handle

data transfers over the link to the CXL Switch.

4. **CXL Switch**

   A CXL Switch sits between a Root Port and two or more CXL devices. It has one upstream port (facing the Root Port) and multiple downstream ports (facing the devices). This allows a single Root Port to connect to multiple memory devices and enables memory interleaving across them within the same memory region. A CXL Switch also enables multiple hosts to share a common memory pool. The switch contains a Fabric Manager responsible for dynamically assigning devices to hosts without requiring system reboots.

   A CXL Switch is optional. If the architecture contains only a single CXL device, a switch is not required. In that case, multiple hosts cannot share the device's memory.

5. **CXL Type 3 Device**

   Using CXL.io for device discovery and CXL.mem for memory access, CXL Type 3 devices expose memory regions to the host that are mapped into the host's physical address space. These devices may provide either volatile or persistent memory. In this thesis, only volatile-memory devices are used.

CXL also supports interleaving, which is the configuration used in this thesis. When multiple CXL devices are attached to the same via a CXL switch to the same root port, the Host Bridge can be configured with an HDM decoder that distributes accesses across both devices in a round-robin or granular fashion [1]. This means that a single logical memory region can physically span multiple devices. In this thesis, two CXL Type 3 devices, each contributing 1 GB, are interleaved into a 2 GB region exposed as a single NUMA node. The interleaving granularity determines how page-sized allocations are distributed across the two physical devices.

From the operating system's perspective, this interleaved region behaves as one contiguous range of physical addresses mapped to a single NUMA node. The kernel has no built-in awareness of which physical device backs any given address within that range. This is a central point of the thesis: when the kernel demotes pages to this region, it does not explicitly choose which of the two underlying devices a page lands on. Instead, the physical destination is determined by the HDM decoder's interleaving configuration. Understanding whether this results in one device filling up before the other, which would be energy-efficient, or both devices filling in parallel, which would not, is the core research question.

2.4 Linux Automatic NUMA Balancing

Linux provides a subsystem called Automatic NUMA Balancing, enabled via the kernel parameter `kernel.numa_balancing`, that periodically monitors page access patterns and migrates

pages to improve locality [8, 11]. This mechanism was originally designed for traditional multi-socket NUMA systems, where all nodes contain CPUs and the goal is to keep data close to the CPU that accesses it. More recently, it has been extended to support memory tiering, where CPU-less nodes represent slower memory tiers.

The mechanism works by periodically unmapping pages in each process's page tables. The next time a thread accesses one of these pages, a page fault is triggered. The kernel intercepts this fault, records which NUMA node the faulting thread belongs to, and then restores the page mapping. This event is called a *NUMA hint fault*. By collecting these hints over time, the kernel builds a picture of which node most frequently accesses each page.

For memory-tiering systems, the mode is set to `NUMA_BALANCING_MEMORY_TIERING`. In this mode, the kernel uses hint faults to identify hot pages residing in the slow tier and migrates them to the fast tier (promotion). At the same time, when the fast tier is under memory pressure, the kernel's memory reclaim daemon (`kswapd`) demotes cold pages from DRAM to the slow tier rather than swapping them to disk [12].

The demotion path is particularly relevant to this thesis. When kswapd determines that the DRAM node is above its memory watermark, it begins scanning the Least Recently Used (LRU) page lists for cold candidates. Pages identified as cold are removed from DRAM and copied to the slow-tier NUMA node, specifically the CXL node in this setup [8]. The kernel function responsible for executing this migration is where the custom logging code in this thesis was inserted. Each log entry captures the old physical address in DRAM, the new physical address in CXL memory, the reason code for the migration, and the corresponding virtual address.

One design choice relevant to this thesis is that the kernel chooses the destination address in the CXL region based on its own allocator, without awareness of whether that address maps to device 0 or device 1. Whether the kernel's allocation pattern respects the physical boundaries of the two interleaved devices is not documented behavior, and observing this pattern empirically is the contribution of this thesis.

The promotion path works in reverse: when a NUMA hint fault occurs for a page currently on the slow-tier node and that page appears hot, based on fault frequency or latency, the kernel migrates it to a free slot in DRAM. The promotion rate is throttled to avoid excessive traffic between tiers, using a mechanism that tracks migration throughput [11].

2.5 Energy Proportionality

The concept of energy proportionality was formally introduced by Barroso and Holzle of Google in 2007. They observed that servers in production data centers spend the large majority of their operational time running at low utilization, roughly between 10% and 50% of peak load. Yet

during this time, the servers still consume a significant fraction of their maximum power. A server that uses 60% of its peak power while sitting idle is very far from the deal behavior, where a machine would consume close to zero power when idle and scale its consumption linearly with how much useful work it performs [13].

The core argument is that energy efficiency should be consistent across all utilization levels, not just at peak load. A system that achieves good efficiency only when fully loaded is structurally wasteful in real-world deployments, where peak load is rarely sustained [14].

Memory was specifically identified as one of the most energy-disproportional components of a server. DRAM requires periodic refresh cycles to maintain stored data, and these refresh operations consume power continuously regardless of whether the memory is actively being read or written [15]. In a server with many populated memory channels, the standby and refresh power of all the inactive DRAM ranks adds up substantially. Research has shown that even under peak workloads, the majority of memory resources remain in standby mode, with their power draw accounting for more than half of the total memory subsystem power [15].

This behavior makes memory a poor fit for environments where workloads are bursty or intermittent. The memory subsystem burns a roughly constant amount of power whether it is serving thousands of requests per second or sitting completely idle. The key to improving memory energy proportionality is reducing the number of active devices when the workload does not require them. If all the data that needs to be accessed can be concentrated into fewer DRAM modules or fewer memory devices, the remaining ones can be allowed to enter low-power states such as self-refresh, where power consumption drops significantly [16].

This principle connects directly to this thesis. The two CXL devices in the experimental setup are identical in capacity and performance characteristics. If the Linux kernel's demotion policy consistently fills one device with cold pages before using the other, then the second device can remain in a low-utilization state. Whether a CXL device in a lowutilization state can nter a reduced-power mode depends on the specific hardware implementation, but the general principle that oncentrating usage enables power savings is well-established in the memory literature [13, 17]. There is a tradeoff here though. By utilizing one device fully before touching the other, you would lose peak bandwidth potential, so if your program requires a mass demotion event to happen, following an energy proportional demotion pattern –filling up one device before demoting on to the other- would increase latency since you wouldn't have the ability to use the bandwidth of both devices to mass demote pages. If your workload's memory pressure is uncertain and you might require suddenly more memory than can fit in the active CXL device, then following the energy proportional demotion pattern would be bad for two reasons: Lowering your peak bandwidth and having a wake-up time for the idle CXL device.

Since this thesis runs in a virtual machine environment and does not have direct access to hard-

ware power monitoring interfaces, energy proportionality is assessed indirectly through page placement patterns. If demotions are concentrated on one device first, the configuration is considered energy-proportional in a structural sense. If both devices receive demotions roughly equally from the start, the configuration is not energyproportional, as neither evice has the opportunity to remain idle.

3 Related Work

3.1 Automatic NUMA Balancing and Page Migration

The challenge of placing pages close to the CPU that uses them has been a subject of operating systems research for decades. Early work by Lameter [3] described the basic design of NUMA memory management in Linux and the tradeoffs between local allocation and remote access. As NUMA systems became more common in commodity servers, the community developed a range of approaches to automate page placement.

Automatic NUMA Balancing, referred to as AutoNUMA, was introduced into the Linux kernel in version 3.8 as a mechanism to periodically hint-fault pages and migrate them toward the node that accesses them most frequently [5]. While effective for standard multi-socket systems, AutoNUMA was found to have limitations in tiered-memory scenarios. Specifically, its original design aimed for locality between a CPU and its local DRAM, but it lacked the ability to distinguish between fast and slow memory tiers when all tiers are presented as NUMA nodes.

The Carrefour system explored the relationship between page placement and memory controller load, showing that blind locality optimization can sometimes cause bottlenecks by overloading a single controller. This work introduced the idea that interleaving, distributing pages across controllers, can sometimes outperform strict locality. This is relevant to the interleaved CXL setup used in this thesis, where accesses to the CXL node are automatically spread across two physical devices.

AutoTiering [7] explored the design space of page management for multi-tiered memory systems. It proposed a conservative promotion mechanism that avoids unnecessary migrations by only promoting pages when there is sufficient evidence of hotness, and an opportunistic promotion mechanism that takes advantage of free slots in the fast tier. This work ran on systems with DRAM and persistent memory, but many of its findings generalize to DRAM and CXL setups.

3.2 CXL-Based Memory Management

As CXL hardware became available, research interest shifted toward understanding how to manage CXL memory effectively in production environments.

TPP (Transparent Page Placement) by Al Maruf et al. [2] is one of the most widely cited works in this area. The authors characterized memory usage patterns across a fleet of Meta production servers and showed that a large fraction of allocated pages are cold at any given time. TPP proposed a lightweight OS mechanism that proactively demotes cold pages to CXL memory

and promotes hot pages from CXL to DRAM, without requiring any application modification. Most of TPP's patches were merged into Linux 5.18. TPP represents the closest prior work to the kernel-level behavior being observed in this thesis, as it directly uses the demotion path that this thesis instruments.

Dissecting CXL Memory Performance at Scale (SupMario) by Liu et al. [18] provided a comprehensive empirical characterization of CXL performance across 265 workloads on 4 different CXL devices. The study identified that CXL memory introduces variable tail latencies and that CPU tolerance to those latencies depends heavily on workload access patterns. The paper also evaluated interleaving policies for CXL memory, introducing a "best-shot" interleaving strategy that chooses how to distribute pages across CXL devices based on workload characteristics. This directly relates to the interleaved setup in this thesis, where the question is not which interleaving strategy to use, but what access distribution results when the kernel manages placement without explicit interleaving awareness.

Colloid by Vuppalapati and Agarwal [19] argued that memory access latency is the most actionable signal for tiering decisions. Rather than relying on access frequency counts, which can be slow to converge, Colloid monitors per-node memory latency in real time and uses this to balance hot pages across tiers based on congestion. This is relevant as a more dynamic alternative to the relatively coarse-grained NUMA balancing approach used in the thesis setup.

HybridTier by Song et al. [9] proposed a lightweight user-space tiering daemon that adapts to dynamically shifting hotness distributions. It demonstrated that real-world workloads often exhibit power-law access patterns, where a small fraction of pages account for the large majority of accesses. Understanding such distributions would be useful when interpreting the migration logs in this thesis, as workloads with concentrated hot sets are likely to show more pronounced demotion patterns.

Managing Memory Tiers with CXL in Virtualized Environments by Zhong et al. [20] studied exactly the setting relevant to this thesis: CXL memory inside a virtualized environment. The paper identified two failure modes specific to multi-tenant virtualization: cross-VM memory contention and intra-VM access pattern conflicts. It proposed Memstrata, an allocator that uses page coloring to isolate VM memory. While this thesis operates in a single-VM setup, this work confirms that the virtualized CXL context introduces complexities that are not present on bare metal, making empirical observation more valuable than theoretical predictions.

3.3 Energy Efficiency in Memory Systems

Malladi et al. [17] analyzed DRAM power consumption in data center contexts and showed that standby and background power dominate memory power at typical server utilization levels.

They proposed using low-power mobile DRAM (LPDDR) in servers as a path toward energy-proportional memory. While the hardware approach differs from what this thesis addresses, the core finding, that memory consumes significant power even when not actively used, motivates the energy proportionality angle of this work.

Barroso and Holzle [13] established the conceptual framework of energy-proportional computing that this thesis builds on. Their paper argued that hardware components should consume power in proportion to the work they perform, and they highlighted memory as one of the worst offenders. Nearly two decades later, the problem remains relevant, and new memory technologies like CXL introduce fresh opportunities to address it through data consolidation.

Research on DRAM self-refresh by Jung et al. [16] showed that enhanced self-refresh modes can reduce DRAM idle power by up to 48%. For CXL memory devices, which are powered and managed independently from the host CPU, similar low-power modes are theoretically possible. Concentrating cold pages onto fewer physical devices could therefore create conditions where unused CXL devices spend more time in reduced-power states.

Jin et al. proposed the DRAM Translation Layer (DTL), a CXL-based mechanism for software-transparent DRAM power management in disaggregated memory systems. Their design uses flexible address remapping and host-transparent page migration to consolidate pages into fewer DRAM ranks, allowing unused ranks to enter low-power states such as power-down and self-refresh. Their evaluation showed average DRAM power savings of 31.6% at a 1.6% performance cost, while a hotness-aware self-refresh mechanism further reduced DRAM energy consumption by up to 14.9% with negligible performance loss [21]. This is directly relevant to the present thesis, because it supports the broader idea that concentrating colder data into fewer physical memory regions can improve energy proportionality.

3.4 Positioning of This Thesis

The existing literature on CXL memory management focuses primarily on optimizing the placement of pages from the perspective of performance, specifically minimizing the number of accesses that land in the slow tier. Energy efficiency is discussed as a secondary motivation in many papers but is rarely evaluated directly in terms of devicelevel utilization patterns.

This thesis takes a different angle. Rather than proposing a new tiering policy, it observes the behavior of the existing Linux kernel policy and asks whether the resulting page placement patterns are consistent with energy proportionality. The specific contribution is an empirical analysis of which physical CXL device within an interleaved two-device configuration receives demoted pages during graph workload execution, and whether the pattern suggests that one device could be powered down while the other serves the full workload.

4 Experiment

4.1 Warning

If anyone is reading this hoping to continue the work or do something similar, I'd like to issue a warning. The reader is discouraged to use institutional/organizational servers that have restrictions such as proxy servers, IP tables, and other safety restrictions. If feasible, a semi-capable computer that can spare 50-100GB of storage, can handle the setup and experiments of this work, especially since the VM's CPU and memory configurations are very light.

4.2 Methodology

This experiment was held on the platform CloudLab with a profile consisting of 2 nodes: Any normal compute node, and an emulab-blockstore fsnode so that the data and VM don't get lost between sessions. To start, I installed QEMU (stable-10.0) and Linux kernel (dcd-v6-2025-09-23). I inserted my migration-logging tool into the kernel and compiled it with a custom configuration file that includes CXL functionality as well as other must-have flags for the VM to run. After the compilation, I created the RAM init functions for booting the VM and downloaded Ubuntu-25.04 as the file system. Using QEMU's commands and the Ubuntu file system, I created the disk image that the VM will live in. Lastly, I booted the VM with its ROM to set up Ubuntu. Since this experiment was done in a CloudLab server with no GUI, I installed a TigerVNC on my home computer, which is a client/server application that allows users to launch and interact with graphical applications on remote machines [22] so that I could complete the setup procedure.

The migration-logging tool I developed is a tool that prints the before and after addresses whenever a migration happens as well as the reason (promotion/demotion). It's placed in the migrate.c file in the migrate_folio_move function right after the variable declarations. Since the migrate_folio_move function already has the from and to addresses as well as the reason for the migration, the tool prints them to the kernel log. With those migrations logged, I analyzed them offline with a Jupyter Notebook for the findings.

There were some considerably time-consuming roadblocks along the way, the most notable was trying to work on an organizational server here at the university. Specifically, giving internet access to the VM on this server. This server had iptables blocking requests from IP addresses excluding some docker containers, and not wanting to alter them, I decided to create an internet bridge from the VM's IP to the server's, so that any internet request such as git or curl would

get routed through the server's IP address, which would be allowed. Usually to do this in Linux you create an internet bridge and choose source and destination IP addresses to attach to it (the VM and server respectively), and when this bridge is created it's considered "DOWN", in other words not working, and you're supposed to bring it "UP". So, after creating this bridge and setting it to UP, I tested from within the VM to access the internet and it didn't work, and not wanting to leave the bridge in the server if it's not working, I tried deleting it, but to delete a bridge you need to set it "DOWN" again and so I did. It happened that I forgot an important step to this procedure: to unlink the ends of the bridge. So, as soon as I set this bridge -that had the server's IP connected to it- "DOWN", the server's IP address went down along with it. Making it inaccessible by ssh as the server didn't have an IP address.

Apart from the struggle of getting an internet connection to the VM, it's another problem that made switch from working on it to CloudLab: Kernel-based Virtual Machine (KVM) acceleration. KVM acceleration allows QEMU to use hardware virtualization extensions (Intel VT-x, AMD-V) for executing guest code. Rather than emulating a CPU in software, it executes guest code directly on the host processor, significantly improving performance. The problem is that to enable KVM acceleration, you need to enable CPU virtualization in BIOS, and this server didn't have it enabled. Not having KVM acceleration makes a VM unbearably slow, the VM is already slow when emulating CXL (20ms for one page demotion) and when I experimented with running very simple programs on it, even before adding CXL components, it was unworkably slow. While yes maybe I could've asked for it to be enabled in the BIOS, after the first mess-up of evaporating the server's IP address and the fact that this needs BIOS changes made me think that it could be a better idea to find another way to work on this thesis. Thankfully I did, because at one point I needed 90G of storage for everything (Linux, QEMU, Disk Image, and others).

Before the switch to CloudLab, I tried working on my laptop with 4 threads and 8GB of RAM and solely from the fact that compiling a Linux kernel took 6 hours I knew that there was no way the thesis would finish this year if I continued on my laptop. After switching to CloudLab, there was only 1 giant time-wasting roadblock. To create a CXL memory region you can use the cxl-create region command from the ndctl repository, and you would expect that by having all the CONFIG_CXL_* flags enabled, it would work fully. For a larger-than-I'd-like-to-admit period, this command would do around 70% of the work, and I spent very long seeing how to do the last 30% manually and incrementally: enabling some decoders, attaching devices to memory regions and other things. After a very thorough manual experimentation process, I created a Direct Access Region (DAX region), that you would have to later point a decoder to. So never seeing the term DAX memory before, I looked online to learn about it and discovered that it also has a utility tool "daxctl list" that's also in ndctl that lists you all your DAX memory regions, so me wanting to see what I created and how to link it to CXL, I type that command and get the reply that I don't have some CONFIG_DAX_* flags enabled. So, like many times before

I recompile the kernel with new flags and want to get back to where I was in the manual labor of enabling CXL. I typed my cxl create-region command, and it did absolutely everything. It created the memory regions and enabled the devices in them, and a new NUMA node appeared. So, after learning how to manually do cxl create-regions job for a very long period of time, all it needed was some DAX configuration flags. Even in the official QEMU docs for CXL, they tell you that the required flags are all CONFIG_CXL_* flags and nothing about DAX.

4.3 Results

To start, I booted the VM with 4GB of DRAM and 2 CXL type-3 devices with 1GB of memory each, and once inside, I created a 2GB CXL memory region that interleaves the 2 devices and makes them appear as a single NUMA node and changed the numa_balancing flag to "tiered". I used the GAPbs which is a benchmark suite about graph processing [23], specifically these 3 benchmarks were used and executed 5 times each, with a larger memory footprint than what local DRAM can fit:

Breadth-First Search (BFS): A graph traversal order which traverses all vertices at the current depth before moving onto the next depth.

PageRank (PR): A ranking algorithm for graphs that computes the PageRank score for all vertices in the graph.

Betweenness Centrality (BC): An algorithm that captures how much a given node is between others.

All 3 of these benchmarks generate synthetic input graphs using the Graph500 Kronecker generator, a stochastic Kronecker model in which each edge is placed by recursively selecting one of four adjacency-matrix quadrants according to initiator probabilities 0.57, 0.19, 0.19, 0.05, for N = 2g vertices and M = edgefactor * N edges [23].

In my experiments, I had the following values:

Parameter	**Value**
g (scale)	22
Edgefactor	16
N (vertices)	$2^{22} = 4{,}194{,}304$
M (edges)	$16 \cdot 2^{22} = 67{,}108{,}864$

Table 4.1: Graph generation parameters used in the experiment

After 5 runs of each benchmark we have these migration statistics:

We can see that the number of promotions is worryingly small, that I believe is due to some undiagnosed error in the setup of this thesis that I could not uncover.

Algorithm	Total Migrations	Promotions	Promotion Rate
BC	6845	10	0.14%
BFS	12700	77	0.60%
PageRank	8159	16	0.20%

Table 4.2: Migration statistics across graph algorithms

4.3.1 BC

Over 5 runs, BC had the following migrations per run:

Run	Migrations
Run 1	484
Run 2	303
Run 3	3593
Run 4	1446
Run 5	1019

Table 4.3: BC run migration counts across executions

These repeats of source and destination addresses were observed:

In runs 1, 4 and 5, BC did not have any pages that got demoted or promoted more than once.

Run 2:

In run 2, BC had demoted the page 0x162fc5000 10 times, and for demotion destinations the page 0x691855000 was chosen 11 times. There were no other pages that got demoted more than once, nor any destinations chosen more than once. The fact that the run with the least migrations had repeated page migrations, where runs with more migrations didn't, shows the randomness of the decisions of the allocator.

Run 3:

This run, being our most migration-filled run, has many re-used addresses for both sources and destinations. The most demoted pages were 0x101a2d000, 0x1256f0000 and 0x 17b4c8000 being demoted 10 times each, along with a few other pages being demoted twice each. For demotion destinations, 3 addresses again were the most used as destinations with 10-11 demotions each and many other addresses with 4 demotions each.

Jaccard Similarity:

Jaccard Similarity of 2 sets (A, B) is defined as:

$$\frac{|A \cap B|}{|A \cup B|}$$

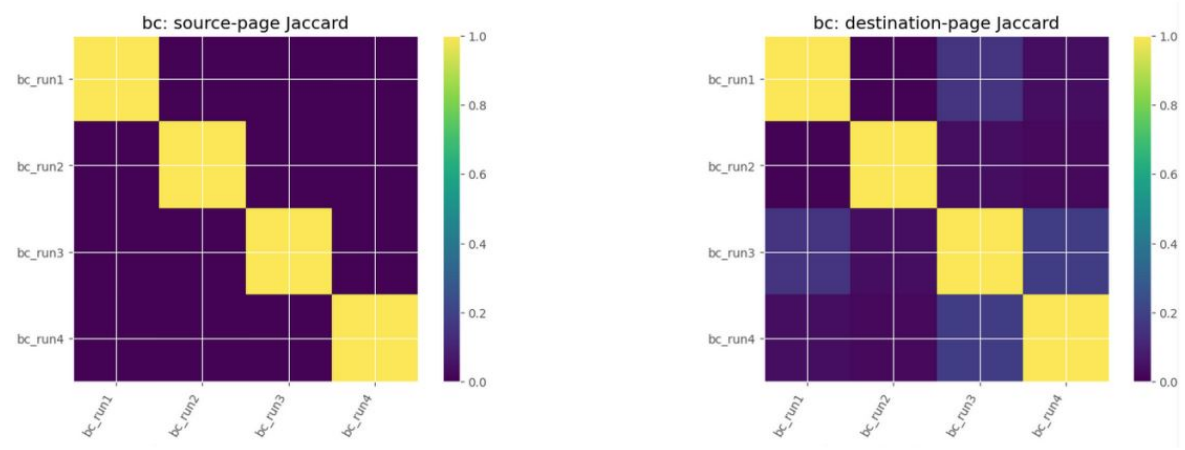

Figure 4.1: Jaccard Similarity of source/destination page addresses

In this case, we are computing the Jaccard Similarity to compare addresses between all the runs, with source pages (addresses in the hot tier) on the left and destination pages (addresses in the cold tier) on the right.

Observations:

Jaccard similarity shows that source-page migrations have very low cross-run overlap, indicating that the exact pages selected for migration vary substantially between executions. In contrast, destination-page overlap is occasionally higher, suggesting that placement targets are more repeatable than source-page selection.

An assumption I made at an early stage in this thesis was that the CXL memory region would fill linearly, so for example: If the CXL region would be 0x100-0x200 and we would have 5 runs with minimum 0x050 memory worth of migrations each, there would be an at least 50% overlap between the destination pages, since all 5 runs would fill up the CXL region at a minimum until 0x150 (up to 0x200).

But the above Jaccard Similarity disproves this. The smallest run (A) had 303 migrations and second smallest (B) 484, if the CXL memory region would fill linearly as I assumed, these 2 runs would have a 0.626 Jaccard Similarity (303/484), because B's destinations would be the 303 destinations of A plus 181 new destinations since both runs would fill the same region linearly.

So, by disproving this, we can make a new assumption that the allocator chooses destinations for demotions non-deterministically.

Migrations to each CXL device throughout all runs:

Observations:

The migrations with interleaved devices show a near 1-1 ratio of demotions to each device. Which is a clear "no" to the question this thesis asks of "Do interleaved CXL devices get utilized

Benchmark	Memory Device	Migrations
bc	mem0	3472
bc	mem1	3373

Table 4.4: Number of migrations to each CXL device throughout all BC runs

in an energy proportional way?".

This finding is very interesting though considering the above assumption that "the allocator chooses destinations for demotions non-deterministically". If the CXL devices would get filled linearly (as disproven above), then a 50-50 split of migrations among the CXL devices would make total sense as they are interleaved, so a linear allocation of 100 pages would land in a pattern of: page 1 → device 0, page 2 → device 1, page 3 → device 0... and so on. However, if it's true that the allocator chooses destinations non-deterministically, the fact that the destinations managed to land in an almost perfect 50-50 split among the devices is a very curious finding.

4.3.2 BFS & PageRank

BFS & PageRank exhibit the same behavior as BC in all aspects.

Benchmark	Memory Device	Migrations
BFS	mem0	6418
BFS	mem1	6282
PageRank	mem0	4137
PageRank	mem1	4022

Table 4.5: Number of migrations per CXL device throughout all 5 runs of BFS and Pagerank

These findings are again so close to a 50-50 split, specifically with this precision: So, with a

Benchmark	mem0	mem1
BC	50.72%	49.28%
BFS	50.54%	49.46%
PageRank	50.70%	49.30%

Table 4.6: Memory device page distribution across benchmarks

supposedly non-deterministic decision of what address to demote to, such a nearly perfect split is an interesting discovery.

5 Conclusion

5.1 Summary

This thesis set out to answer a specific question: when the Linux kernel demotes cold pages to a CXL memory region that spans two interleaved devices, does it fill one device before the other, or does it spread pages across both from the start? The answer has direct implications for energy proportionality. A policy that concentrates pages on one device would allow the second device to remain idle, creating conditions for potential power savings. A policy that fills both devices in parallel denies either device the chance to enter an idle state.

To investigate this, a custom block of code was added to the Linux kernel's page migration function. The patch records the physical source address, physical destination address, virtual address, and migration reason for every demotion and promotion event. A virtual machine was configured with 4 GB of DRAM and two CXL Type 3 devices, each contributing 1 GB, interleaved into a single 2 GB NUMA node. Three graph benchmarks from the GAP benchmark suite, Breadth-First Search, PageRank, and Betweenness Centrality, were each run five times with a memory footprint large enough to trigger sustained demotion activity.

5.2 Findings

The results were consistent across all three workloads and all fifteen runs. Demoted pages were split with a near 1:1 ratio between the two CXL devices in every case. The Linux kernel's demotion policy, as implemented in the version tested, does not favor one device over the other. Both devices receive pages from the very first demotion, and neither is given the opportunity to remain idle. The answer to the central research question is therefore no: the current Linux policy does not produce energy-proportional page placement in this configuration.

A secondary finding came from the Jaccard similarity analysis of destination addresses across runs. An initial assumption was that the CXL region would fill linearly, meaning that two runs with similar migration counts would show high overlap in their destination pages. The Jaccard scores disproved this. Destination pages vary substantially between runs even when the total number of migrations is similar, which suggests that the kernel's allocator chooses demotion destinations non-deterministically rather than filling the region in order. This makes the consistent 50/50 device split more interesting. Even though the individual pages chosen as destinations change between runs, the overall distribution across the two physical devices remains balanced.

The promotion rate was unexpectedly low across all workloads, ranging from 0.14% to 0.60%

of total migrations. This was not the intended behavior and is likely the result of an unresolved issue in the experimental setup. The implications of this are discussed below.

5.3 Limitations

Several limitations should be kept in mind when interpreting these results.

The experiment was conducted entirely inside a virtual machine. As discussed in the related work, virtualized CXL environments introduce complexities that are not present on bare metal, including address translation overhead and potential interference from the hypervisor. The page placement patterns observed here may differ from what would be seen on a physical system with direct CXL hardware access.

Energy proportionality in this thesis is assessed through page placement patterns, rather than through direct power measurements. The experimental environment did not provide access to hardware power monitoring interfaces. Whether a CXL device that receives fewer demotions would actually enter a reduced-power state in practice depends on the specific hardware implementation and is not something this thesis can confirm.

The low promotion rate is the most significant unresolved issue. In a correctly functioning tiered memory setup, pages that are accessed again after demotion should be promoted back to DRAM at a meaningful rate. The near-zero promotion counts seen here suggest that either the promotion mechanism was not functioning correctly in this kernel configuration, or that the workloads, once they caused an initial wave of demotions, did not re-access cold pages frequently enough to trigger promotions. This limits the conclusions that can be drawn about the full migration cycle and is something that would need to be resolved before extending this work.

5.4 Continuing this work

This section is dedicated to anyone thinking about continuing this work. My proposed topics would be:

1. Most critical: Considering the main issue of allocations to the CXL device in QEMU taking 20ms to happen, try figuring out why that happens. Is it really just a “QEMU can’t optimize cache lining” problem?
2. Figure out if there are any issues with the setup and hopefully lead to: Why promotions are so few? Is there a better way to log migrations compared to my code in `migrate.c`?
3. Maybe compare QEMU with other (if there are) emulators that can emulate CXL.
4. Model energy metrics for this VM or create scripts/daemons for it that can do it.

5. Figure out why the VM gets “bogus descriptor or out of resources” crashes under heavy memory pressure and why it crashes with XSAVE instructions.
6. Grand idea: Since QEMU is open source, explore whether you can alter migration/allocation destinations to interleaved CXL devices. If so, design it so one device fills to a threshold before the other receives allocations/migrations, enabling the other to stay in low-power mode. This may introduce complications with multiple hosts, so focus on a single-host setup.

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/, and with full accountability on my part for the accuracy, integrity, and validity of all content.

BIBLIOGRAPHY

[1] Q. Documentation. Compute express link. [Online]. Available: https://www.qemu.org/docs/master/system/devices/cxl.html

[2] H. Al Maruf, H. Wang, A. Dhanotia *et al.*, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[3] C. Lameter, "Numa (non-uniform memory access): An overview," *ACM Queue*, vol. 11, no. 7, 2013.

[4] Intel. Hardware and software approach for using numa systems. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/hardware-and-software-approach-for-using-numa-systems.html

[5] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quéma, and M. Roth, "Challenges of memory management on modern numa systems," *ACM Queue*, 2015. [Online]. Available: https://queue.acm.org/detail.cfm?id=2852078

[6] L. K. Documentation. What is numa? [Online]. Available: https://www.kernel.org/doc/html/v4.18/vm/numa.html

[7] J. Kim *et al.*, "Autotiering: Exploring the design space of page management for multi-tiered memory systems," in *USENIX Annual Technical Conference (ATC)*, 2021.

[8] S. Scargall. (2022) Using linux kernel memory tiering. [Online]. Available: https://stevescargall.com/blog/2022/06/using-linux-kernel-memory-tiering/

[9] K. Song *et al.*, "Hybridtier: An adaptive and lightweight cxl-memory tiering system," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

[10] D. D. Sharma, "Compute express link 3.0," CXL Consortium, White Paper, 2022.

[11] L. K. Documentation. /proc/sys/kernel/numa_balancing. [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/sysctl/kernel.html

[12] LWN.net. (2022) Numa rebalancing on tiered-memory systems. [Online]. Available: https://lwn.net/Articles/893024/

[13] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[14] J. Cebrian, S. D. D. Anton, and F. Bellas, "Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and

performance," *Applied Energy*, vol. 268, p. 114957, June 2020. [Online]. Available: https://doi.org/10.1016/j.apenergy.2020.114957

[15] V. Delaluz, "The power of memory," *Electronic Design*, June 2010. [Online]. Available: https://www.electronicdesign.com/markets/energy/article/21791777/the-power-of-memory

[16] M. Jung *et al.*, "Enhancing dram self-refresh for idle power reduction," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2016.

[17] K. T. Malladi *et al.*, "Towards energy-proportional datacenter memory with mobile dram," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012.

[18] J. Liu, H. Hadian, H. Xu, D. S. Berger, and H. Li, "Dissecting cxl memory performance at scale: Analysis, modeling, and optimization," *arXiv preprint arXiv:2409.14317*, 2024. [Online]. Available: https://arxiv.org/abs/2409.14317

[19] M. Vuppalapati and R. Agarwal, "Tiered memory management: Access latency is the key!" in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2024.

[20] Y. Zhong *et al.*, "Managing memory tiers with cxl in virtualized environments," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

[21] e. a. Jin, "Dram translation layer (dtl): Software-transparent dram power management in disaggregated memory systems," placeholder entry added from thesis text; complete venue, year, and URL if available.

[22] TigerVNC. Tigervnc. [Online]. Available: https://tigervnc.org/

[23] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015. [Online]. Available: https://arxiv.org/abs/1508.03619

APPENDICES

APPENDIX I
Github Tutorial

During this thesis I made a tutorial on github about the setup process.

https://github.com/PaulAdrianDev/Emulating-CXL-devices-with-QEMU-VM-tutorial