CS-477 Group Project Proposal
Intelligent Transparent Huge Page Shrinking Utilizing PEBS Monitoring

Tudor-Stefan Pagu EPFL
Mohamed Taha Guelzim EPFL
Mamoun Imghi EPFL
Makar Kuleshov EPFL

Abstract

Given the rising demand for huge RAM capacities, the translation lookaside buffer (TLB) has increasingly become a bottleneck after multiple misses; thus making huge pages, which alleviate some pressure off the TLB, an attractive optimization. However, poor utilization of huge pages can, in turn, increase memory fragmentation, waste DRAM, and affect performance. This project aims to combine several cutting edge approaches to huge page management in order to implement a lightweight and efficient huge page shrinking and merging algorithm.

1 Introduction

Our project tackles the shrinkage and merging of huge pages in memory. Transparent huge pages (THP) is a feature of the Linux kernel which, if enabled, makes the kernel default to using huge pages, without any modification in the user space program. [1]. This reduces the total amount of pages, therefore increasing TLB reach and improving performance. However, as pointed out by [1, 2], THP can cause applications to allocate more memory than needed, and can worsen memory fragmentation. In particular, an application may allocate an entire 2MB huge page but only use a small portion of it. This is known as skewed access, which presents a significant challenge, especially in tiered memory systems where efficient use of more expensive tiers of memory is essential [2].

One solution to these problems is huge page shrinking. This involves an algorithm to identify which pages should be split, split them, and free up when possible some of the subpages that are deemed unused. There have been proposals for such implementations in the Linux kernel [3], but this approach suffers from a lack of insight into the usage pattern for the huge pages. Instead, it relies on simple heuristics such as analyzing the content of the huge page and looking for 0s, which indicates unused memory. Insight can be gained into the usage of huge pages using PEBS counters, as was done in [2]. However, this approach involves constructing a completely new page allocator, centered around tiered memory systems, and is not easy to directly integrate into pre-existing huge page shrinking algorithms.

Our goal for this project is to implement a lightweight version of the PEBS based approach of Memtis [2], which tracks statistics on huge pages which can later be used by a huge page shrinking algorithm. First, by listening to PEBS events such as ”retired store” and ”retired load” instructions and looking at the virtual address of these events, we can maintain some of the access patterns occurring within huge pages. Then, we can use these statistics to draft an implementation of a huge page shrinking algorithm, similar to [4, 3], which will periodically split huge pages. Lastly, we will evaluate the performance of our algorithm against the mainline kernel, measuring for TLB misses, working set size, overall running time, etc, to evaluate its efficiency compared to existing alternatives.

2 Analysis of Prior Work

2.1 Memtis (Lee et al., 2023) [2]

Memtis introduces a hardware-assisted mechanism to improve huge page utilization in tiered memory systems, particularly in environments with heterogeneous memory (e.g., DRAM + NVM). The key idea is to track memory access at a fine-grained subpage level using **Precise Event-Based Sampling (PEBS)** counters available in modern Intel processors. This allows the system to detect skewed huge pages where only a small portion of the page is actively used, which is critical in reducing memory bloat and improving TLB efficiency.

How PEBS works. PEBS is a hardware feature in Intel CPUs that logs performance events directly in a buffer with minimal overhead. Events such as cache misses, or retired loads/stores (i.e. fully executed memory operations) can be recorded along with the memory address, instruction pointer, and execution context. This gives detailed insight into memory access patterns without requiring software to scan entire pages manually. Using PEBS, Memtis can identify which 4 KB subpages within a 2 MB huge page are actively used.

Detecting skewed subpages. Memtis uses PEBS samples to identify *skewed* huge pages: pages where a few subpages are hot while the rest are mostly unused. Actions taken include **splitting** huge pages to reclaim cold subpages and **migrating** hot subpages to faster memory tiers (DRAM) and cold subpages to slower tiers (e.g., NVM) in tiered memory systems.

This enables fine-grained memory optimization while keeping runtime overhead low, since the heavy lifting is done by hardware-assisted sampling rather than intrusive software scanning.

2.2 Thinking about a New Mechanism for Huge Page Management (Li et al., 2019) [4]

Li et al. propose a software-based mechanism to adaptively monitor huge page usage in the Linux kernel. Their system combines **SysMon-H**, a kernel sampling module, with the **H-Policy**, a huge page management policy that promotes and demotes pages based on observed behavior.

SysMon-H monitoring. SysMon-H tracks page activity using **TLB miss counters** as a lightweight indicator of page "temperature." Pages with high TLB misses are classified as *hot*, while those with few misses are considered *cold*. To improve accuracy, limited PTE sampling detects "very hot" pages that remain resident in the TLB. Empirical evaluation shows that SysMon-H adds less than 0.1% runtime overhead on large cloud workloads, compared to more than 2% overhead for conventional access-bit scanning. Monitoring data is organized per-application using an H-Tree, allowing fine-grained decisions for promotion or demotion.

H-Policy promotion and demotion. H-Policy uses SysMon-H data to manage huge pages according to system pressure and workload behavior:
Split cold huge pages into 4 KB base pages when memory utilization exceeds a high-pressure threshold (set at 90%) to reclaim unused memory and **promote** contiguous base pages to huge pages when they show high TLB activity, reducing translation overhead.

The policy also modifies the Linux buddy allocator to preserve order-9 slabs (2 MB blocks) to prevent fragmentation and enable faster huge page allocation without invoking costly compaction.

Significance. Li et al.'s work highlights how dynamic and adaptive policies, combined with per-application monitoring, can significantly improve huge page utilization compared to heuristic-only approaches. It also provides inspiration for implementing lightweight monitoring in our project, even without full hardware-assisted PEBS support. In our project, we can build upon this work by using the more fine grained information PEBS provides.

2.3 Coordinated and Efficient Huge Page Management with Ingens (Kwon et al., 2016) [5]

This paper introduces Ingens – a framework for transparent huge page management.

It is mainly based on two datastructures – a 512-bit utilization bitvector marking which 4 KB subpages are allocated and an 8-bit per-page access history bitvector. A background Scan-kth thread samples and updates the access history from hardware access bits, while Promote-kth consumes the util bitvector and signals from Scan-kth to perform asynchronous promotion or demotion of pages.

On a page fault, Ingens keeps the fast path minimal: the handler only allocates the 4 KB base page and updates the utilization bit. If utilization crosses the promotion threshold it enqueues a request for Promote-kth, but does not perform the operations syncrchronously. Promote-kth later performs the necessary operations for promotion or schedules demotion when utilization drops, using information from Scan-kth to avoid demoting frequently accessed regions.

The experiments that the authors conducted show performance improvements up to 18% and reduction of memory bloat from 69% to less than 1% in real scenarios such as the a web services benchmark and a Redis key-value store.

Relevance to Our Project

Each of these prior works provides a different perspective on huge page management.
First, **Li et al., 2019** introduces a dynamic, TLB-based monitoring system (SysMon-H) combined with a workload-aware H-Policy. This demonstrates the benefits of using hardware counters and per-application monitoring to improve memory management, which inspires our lightweight approach.
Then, **Lee et al., 2023 (Memtis)** shows that PEBS counters can provide fine-grained insight into subpage access patterns, which enables efficient shrinking without scanning entire huge pages. Our project builds a lightweight version of this for integration with the existing Linux huge page shrinker.
Finally, **Kwon et al., 2016** demonstrates that it is possible to build an efficient framework for huge page management by keeping additional statistics for page accesses. With this information it is possible to achieve improve-

ments in performance and significantly reduce the memory bloat in real world scenarios.

Key Takeaway for Our Project: We aim to combine the practical integration strategy of the Linux THP shrinker with the fine-grained access insights from Memtis. Compared to Li et al., we focus on a lightweight implementation that works with the existing Linux kernel infrastructure rather than building a full new allocator. This allows incremental contributions while still improving skewed subpage detection and reducing TLB pressure. As for Ingens, our approach is also based on calculating access statistics, but we will utilize hardware-assisted, sampling-based monitoring that gives more precise information.

3 List of Deliverables

#	Deliverables
1	A kernel component which gathers PEBS statistics to maintain statistics on huge pages and their skewedness.
2	A set of patches to the kernel THP logic to use our component, enabling intelligent decisions, such as what huge pages to split, merge, etc.
3	A set of benchmarks, workloads and experiments which can measure the performance of our system (including setup scripts, test program, and python scripts to create graphs and statistics).
4	Midterm report and Final report documenting implementation, experiments, and results.

4 Research Questions

#	Research Question
1	How accurately can PEBS-based sampling detect skewed subpages compared to heuristic THP shrinker scanning?
2	What is the runtime overhead of a lightweight PEBS-inspired huge page shrinking algorithm in the Linux kernel?
3	How does the proposed approach affect memory utilization and fragmentation for applications with irregular memory access patterns?
4	Can the lightweight algorithm reduce TLB misses compared to the mainline THP shrinker under various memory pressures?
5	How does the algorithm perform across different memory tiers (e.g., DRAM + NVM) in a simulated or actual tiered memory system?

5 Evaluation

Level	**Objectives**
50%	Implement PEBS-based sampling for huge pages; identify hot/cold subpages on small workloads; integrate with Linux THP shrinker.
75%	Validate subpage detection on larger workloads; reduce memory fragmentation; measure runtime overhead and TLB impact; minimize impact on processes accessing pages during shrink/promotion.
100%	Achieve high-accuracy skewed subpage detection comparable to Li et al.; significantly improve memory fragmentation and TLB misses; evaluate tiered memory optimization potential; parallelize PEBS sampling and huge page shrink/promotion in background kernel threads to minimize overhead and adapt in real time across workloads.

6 Importance of Our Project

Efficient huge page management is critical for modern memory-intensive applications and large-memory systems. Poor utilization of huge pages leads to memory fragmentation, wasted DRAM, and increased TLB misses, which directly degrade performance. Our project aims to improve the Linux kernel's huge page shrinking mechanism by leveraging lightweight PEBS-based sampling to detect skewed subpages accurately. By doing so, we expect to:

- Reduce memory fragmentation and improve overall memory efficiency.
- Lower TLB misses and improve system performance for workloads with large or irregular memory footprints.
- Provide a practical, lightweight solution that integrates with existing Linux kernel infrastructure.

This project contributes both a better understanding of huge page access patterns and a tangible improvement to kernel memory management, with potential benefits for data-intensive applications, cloud servers, and tiered memory systems.

7 Distribution of Work

- Tudor and Mohamed: Module to gather statistics about huge pages access patterns and their skewedness.
- Makar and Mamoun: New huge page shrinking algorithm and its integration with the existing THP shrinking logic based on the previously gathered statistics.

References

[1] *Transparent Hugepage Support*. https://www.kernel.org/doc/html/v4.18/admin-guide/mm/transhuge.html. Linux Kernel Documentation (v4.18.0). Accessed 2025-10-16. Linux kernel developers, 2018.

[2] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. "Memtis: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23)*. Koblenz, Germany: Association for Computing Machinery, Oct. 2023. DOI: 10.1145/3600006.3613167. URL: https://dl.acm.org/doi/10.1145/3600006.3613167.

[3] Jonathan Corbet. *The transparent huge page shrinker*. https://lwn.net/Articles/906511/. LWN.net. Accessed 2025-10-16. Sept. 2022.

[4] Xinyu Li, Lei Liu, Shengjie Yang, Lu Peng, and Jiefan Qiu. "Thinking about A New Mechanism for Huge Page Management". In: *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '19)*. Hangzhou, China: Association for Computing Machinery, Aug. 2019, pp. 40–46. DOI: 10.1145/3343737.3343745. URL: https://dl.acm.org/doi/10.1145/3343737.3343745.

[5] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. "Coordinated and Efficient Huge Page Management with Ingens". In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. Savannah, GA, USA: USENIX Association, 2016, pp. 705–723. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-kwon.pdf.