УДК 004.25
DOI https://doi.org/10.24144/2616-7700.2025.46(1).285-293

**П. В. Тарнавський[1], В. М. Білецький[2]**

[1] Львівський національний університет ім. І. Франка,
аспірант кафедри прикладної математики
petro.tarnavskyi@lnu.edu.ua
ORCID: https://orcid.org/0000-0003-3265-8168

[2] Львівський національний університет ім. І. Франка,
доцент кафедри прикладної математики,
кандидат фізико-математичних наук
vasyl.biletskyy@lnu.edu.ua
ORCID: https://orcid.org/0009-0009-3246-9894

# ВИКОРИСТАННЯ ВЕЛИКИХ СТОРІНОК ПАМ'ЯТІ ДЛЯ ПОЛІПШЕННЯ РОБОТИ СУЧАСНИХ ОПЕРАЦІЙНИХ СИСТЕМ

Здійснено огляд сучасних технологій і викликів, пов'язаних із використанням великих сторінок пам'яті в обчислювальних системах. Розглянуто вплив великих сторінок на роботу *буферів асоціативної трансляції* (БАТ) та ефективність керування пам'яттю. Висвітлено основні проблеми, пов'язані з фрагментацією пам'яті, зростанням ймовірності БАТ промахів і зниженням продуктивності за високого рівня багатозадачності. Згадано сумісність великих сторінок із програмним забезпеченням та у сучасних дистрибутивах Linux.

**Ключові слова:** оперативна пам'ять, керування пам'яттю, буфер асоціативної трансляції, великі сторінки пам'яті, фрагментація пам'яті.

**1. Вступ.** З моменту появи *електронно-обчислювальних машин* (ЕОМ) розпочались перегони щодо збільшення їхньої продуктивності. Однією з ключових подій цього процесу стало відкриття, зроблене Гордоном Муром 1965 року, яке згодом отримало назву "закон Мура". Він передбачив, що кількість транзисторів на інтегральних схемах подвоюватиметься кожні два роки, забезпечуючи експоненціальне зростання обчислювальних можливостей. Це спостереження стало рушійною силою швидкого розвитку напівпровідникових технологій.

Однак, поряд із значними успіхами у сфері розробки процесорів, прогрес у розвитку оперативної пам'яті не демонструє такої ж динаміки. Адже, відсутність експоненціального зростання у цій галузі створюватиме суттєвий розрив між швидкодією процесорів і можливостями роботи оперативної пам'яті. Якщо така тенденція зберігатиметься, то оперативна пам'ять у майбутньому може стати головним обмежувальним фактором продуктивності обчислювальних систем.

Сучасні системи використовують механізм сегментації пам'яті, де вона поділена на серію безперервних блоків — сторінок пам'яті. Здебільшого *операційні системи* (ОС) застосовують стандартний розмір сторінок у 4 КБ під час виконання процесів.

Зміна розміру сторінок може стати одним із потенційних рішень для зменшення розриву між швидкодією процесорів і можливостями оперативної пам'яті. Використання великих сторінок пам'яті (наприклад, 2 МБ або більше) даватиме змогу зменшити кількість операцій зі співставлення адрес і знижуватиме

витрати на керування пам'яттю, що, своєю чергою, поліпшить загальну продуктивність системи. Такий підхід може значно підвищити ефективність використання пам'яті в умовах стрімкого зростання обчислювальних потужностей процесорів.

**2. Віртуальна пам'ять.** Віртуальна пам'ять — це підхід до керування пам'яттю, який створює абстракцію між фізичною пам'яттю та процесами, що виконуються у системі. Він дає змогу кожному процесу сприймати фізичну пам'ять як власний суцільний адресний простір (рис. 1), ізольований від адресних просторів інших процесів. Для роботи цього підходу неохідно вміти співставляти віртуальні адреси, згенеровані процесами, у фізичні адреси комп'ютерної пам'яті.

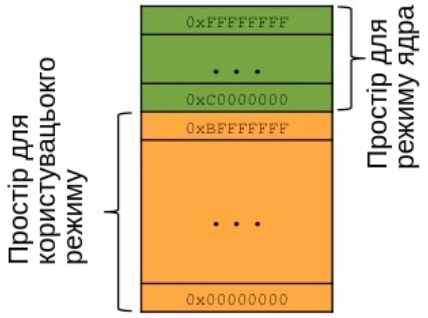


Рис. 1. Приклад адресного простору для процесу

Для реалізації віртуальної пам'яті зазвичай використовують механізм сторінкової організації, який поділяє віртуальний адресний простір та фізичну пам'ять на блоки фіксованого розміру, що називають сторінками, стандартного розміру 4 КБ. При роботі із пам'яттю ОС виконує доступ до цілих сторінок пам'яті, а не послідовно байт за байтом.

Трансляція віртуальних адрес у фізичні здійснено за допомогою модуля керування пам'яттю (МКП), апаратного блоку процесора, що реалізує це перетворення. МКП використовує спеціальні таблиці відповідностей — таблиці сторінок, які зберігають співвідношення між віртуальними сторінками і фізичними фреймами. Операційна система створює та підтримує ці таблиці, надавши кожному процесу окремий віртуальний адресний простір і, таким чином, ізолює його пам'ять від пам'яті інших процесів.

Основними перевагами віртуальної пам'яті є:

- Ізоляція та безпека пам'яті — кожен процес працює у власному адресному просторі, що унеможливлює випадковий або навмисний доступ до пам'яті інших процесів.
- Ефективне використання пам'яті — оскільки у пам'яті збережено лише активні сторінки, система може одночасно запускати більше програм або працювати з більшими обсягами даних.
- Спрощене управління пам'яттю — віртуальна пам'ять забирає необхідність для програм контролювати розподіл пам'яті, ці програми працюють у своїх ізольованих просторах.

Сучасні МКП додатково до таблиць сторінок використовують спеціальний кеш — *буфер асоціативної трансляції* (БАТ). Цей буфер зберігає інформацію

про останні співставлення сторінок, що дає змогу значно прискорити процес доступу до пам'яті.

Операційна система перевіряє, чи в БАТ є відповідний запис щодо фізичної адреси, яка відповідає віртуальній адресі, до якої певний процес пробує достукпитись. Якщо такий запис існує, то це називають БАТ влучанням: процесор отримує доступ до фізичної пам'яті без затримок. За відсутності запису, відбувається БАТ промах. У цьому випадку ОС використовує таблиці сторінок (див. рис. 2), щоб знайти відповідну фізичну адресу. Цей процес повільніший, порівняно з БАТ влучанням, оскільки потребує додаткового часу на пошук відповідності у таблиці сторінок.

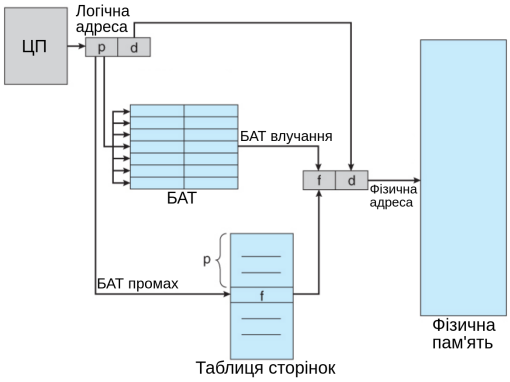


Рис. 2. Схема обробки трансляції адрес

**3. Великі сторінки пам'яті.** Як зазначалось вище, стандартні сторінки мають розмір 4 КБ, ідея великих сторінок пам'яті полягає у використанні сторінок більшого розміру, зазвичай 2 МБ або 1 ГБ. Великі сторінки використовують поряд зі стандартними сторінками. Такий підхід дозволить ефективніше використовувати обмежений розмір БАТ (див. рис. 3), так як один запис для великої сторінки у БАТ буде містити інформацію про більший об'єм фізичної пам'яті.

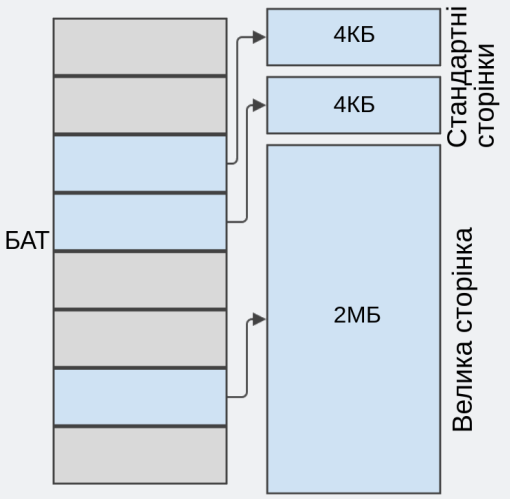


Рис. 3. Трансляція віртуальних адрес

Згідно з дослідженнями, у системах із стандартним розміром сторінки, часті БАТ промахи можуть спричинити значне погіршення продуктивності, що сягатиме до 50 % часу виконання процесу [1–3]. Адже звернення до таблиці сторінок для отримання фізичної адреси може бути у десятки разів повільнішим порівняно з роботою з кешованими даними у БАТ. Зі зростанням обсягу пам'яті в сучасних системах, зазнає і збільшення кількість сторінок, що, своєю чергою, підвищує ймовірність БАТ промахів та сповільнює пошук відповідного запису у таблиці сторінок. Зрештою, операції з обробки стандартних сторінок стануть ключовим місцем, що займає значний обсяг часу та ресурсів у сучасних системах.

Одним із рішень цієї проблеми є використання великих сторінок пам'яті. Великі сторінки, (розміром 2 МБ або більше) даватимуть змогу зменшити загальну кількість сторінок у системі, що знизить навантаження на БАТ і зменшить ймовірність промахів у більшості тестів продуктивності (наприклад, REDIS та CANNEAL). Детальнішу статистику для різних тестів можна переглянути на рис. 4. Також завдяки меншій кількості записів у таблицях сторінок, система може швидше виконувати трансляцію адрес, що доволі важливо у випадках роботи з великими обсягами даних, наприклад, у базах даних або при обробці даних для моделей глибокого навчання.

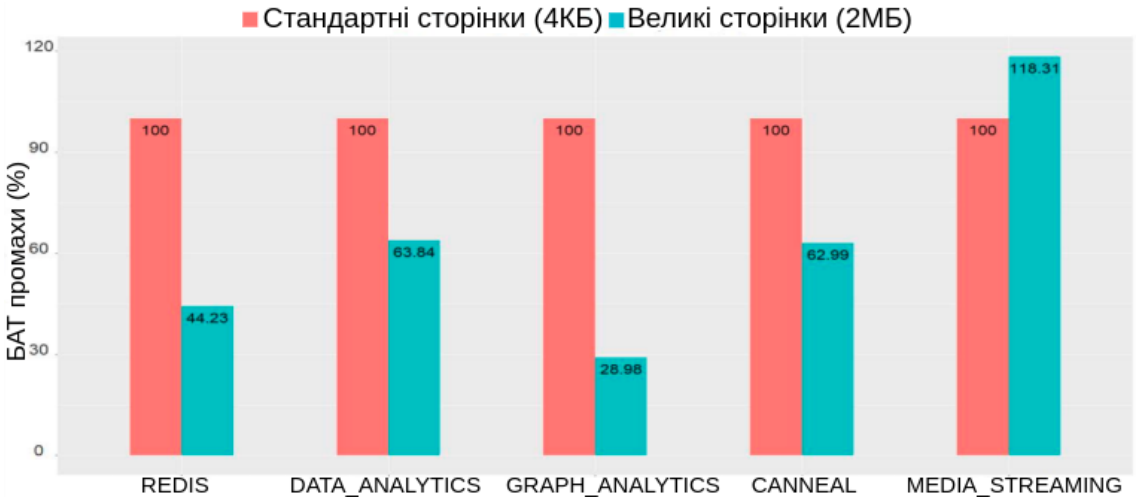


Рис. 4. Вплив великих сторінок на БАТ промахи в порівнянні зі стандартними сторінками [11]

Проте використання великих сторінок також має свої обмеження. Наприклад, великим сторінкам потрібно більше безперервного простору у фізичній пам'яті, що може спричиняти фрагментацію пам'яті. Це може стати проблемою для довготривалої роботи систем, передусім в умовах активної багатозадачності, коли часто звільняють і перерозподіляють блоки пам'яті. Крім того, не всі програмні продукти і драйвери оптимізовані для роботи з великими сторінками, що може спричинити погіршення їхньої продуктивності або проблеми із сумісністю.

**4. Виклики та обмеження використання великих сторінок пам'яті.** Попри те, що великі сторінки пам'яті можуть значно поліпшити ефективність

роботи буферів асоціативної трансляції та зменшити ймовірність промахів, їхнє використання також спричиняє низку викликів та обмежень. Сучасні методи та алгоритми керування віртуальною пам'яттю розробляли переважно для роботи зі сторінками стандартного розміру і не враховували специфіку великих сторінок, що вимагає створення нових методів керування, таких як удосконалені механізми зміни контексту, ефективні стратегії розподілу та звільнення пам'яті.

**Вплив на продуктивність під час зміни контексту**

При перемиканні процесів ОС повинна повністю або частково очищувати вміст БАТ, оскільки записи, актуальні для попереднього процесу, стають неактуальними для нового. Використання застарілих записів у БАТ може спричинити серйозні помилки доступу до пам'яті, отож для вирішення цієї проблеми треба або повністю очищати буфер, або ретельно керувати його записами.

Оскільки часте повне очищення записів БАТ може суттєво вплинути на продуктивність системи, передусім у багатозадачних системах із високою частотою зміни контексту, то часто використовують ідентифікатор адресного простору процесу, що дає змогу розрізняти записи, які належать різним процесам. Це сприяє зменшенню частоти повного очищення БАТ при перемиканні контексту [4].

Використання БАТ з великими сторінками пам'яті робить управління цими записами складнішим. Оскільки запис для великих сторінок охоплює більший обсяг пам'яті та система повинна ретельніше стежити за актуальністю записів під час зміни контексту. Це може суттєво зменшувати продуктивність, насамперед у випадках, коли кількість перемикань контекстів є значною. Отже, ефективність роботи великих сторінок значною мірою залежить від частоти зміни контекстів у системі та ефективності керування БАТ при цьому.

**Фрагментація пам'яті**

У багатозадачних середовищах, де завдання постійно зазнають зміни, звільняють і перерозподіляють різні блоки пам'яті, виникає проблема фрагментації — ситуації коли для великої сторінки необхідно мати в наявності великий безперервний блок фізичної пам'яті, що не завжди можливо. Якщо система не може знайти такий блок пам'яті, то вона може виділити звичайні сторінки замість великих, що знизить ефективність роботи і спровокує зростання ймовірності промахів БАТ.

**Зменшення продуктивності для певних робочих навантажень**

Кожен запис у БАТ для великих сторінок охоплює більший діапазон віртуальних адрес, що дає змогу зменшити кількість записів у БАТ. Якщо ж робоче навантаження процесу пробує доступитись до різних ділянок, розкиданих по всьому адресному просторі, то кількість унікальних сторінок, які можуть бути закешовані, значно зменшиться. У наслідок цього може відбутись зменшення продуктивності, оскільки доступ до кеш-пам'яті є швидшим ніж до оперативної пам'яті.

**Різноманіття розмірів великих сторінок**

Більшість сучасних систем, що використовують великі сторінки, комбінують стандартні сторінки з великими сторінками розміром 2 МБ і 1 ГБ. Це дозволяє гнучко керувати ресурсами пам'яті, але водночас потребує складних адаптивних алгоритмів, які здатні автоматично підбирати оптимальний розмір сторін-

ки для кожного завдання. Такі алгоритми повинні не лише перемикатися між розмірами сторінок відповідно до поточного навантаження, але й враховувати доступні ресурси та специфіку запитів пам'яті.

Реалізація таких алгоритмів є непростим завданням, насамперед у використанні сторінок різного розміру, що значно ускладнює керування пам'яттю та підвищує ризик фрагментації. Передусім це важливо у багатозадачних середовищах, де часто змінюються умови розподілу пам'яті, і адаптивні алгоритми мають швидко реагувати на зміни. Проте розробка таких алгоритмів і їх впровадження на практиці є досить повільним процесом через високу складність і значні вимоги до обсягу пам'яті та обчислювальної потужності для їхньої підтримки.

Через усі ці виклики деякі результати досліджень [5, 6] засвідчують, що використання великих сторінок пам'яті не завжди є виправданим і може спричинити зниження загальної продуктивності. У таких випадках рекомендовано відключення цього механізму.

## 5. Можливості використання великих сторінок пам'яті.

### Підходи до використання великих сторінок пам'яті

Підтримка великих сторінок у сучасних дистрибутивах Linux реалізована за допомогою механізму *прозорих великих сторінок* (ПВС), що дає змогу значно спростити їхнє використання для програмного забезпечення [7, 8]. Прозора підтримка великих сторінок є основним способом виділення великих сторінок у Linux і забезпечує автоматичне керування цими сторінками на рівні ядра. Такий підхід не потребує жодних змін у програмному коді застосунків.

Під час роботи ОС спочатку виділяє пам'ять для процесів у вигляді стандартних сторінок. За оптимальних умов ядро автоматично підвищує послідовність із 512 таких сторінок до однієї великої сторінки, або ж, за потреби, виконує зворотний процес і розбиває велику сторінку на менші. Такий підхід дає змогу значно підвищити продуктивність процесів, використовуючи можливості великих сторінок без потреби змін з боку користувача.

### Динамічні розподільники пам'яті

Програми можуть працювати з великими сторінками і без механізму ПВС, а за допомогою низькорівневих функцій, які дають змогу системі виділити сторінки вказаного у програмі розміру. Наприклад, використання функцій `mmap` та `madvise` з прапорцями `MAP_HUGETLB` і `MADV_HUGETLB`, дає змогу отримати великі сторінки пам'яті. Проте, такий підхід є доволі складним і потребує написання додаткового коду з використанням цих низькорівневих функцій.

Більш зручним варіантом використання великих сторінок без ПВС є динамічні розподільники пам'яті, або алокатори. Наприклад, стандартна бібліотека C містить розподільник malloc, який забезпечує базове керування пам'яттю. Окрім нього, існують спеціалізовані алокатори, налаштовані на ефективне управління великими блоками пам'яті, серед яких популярні jemalloc [9] і tcmalloc [10]. Ці алокатори часто використовують для роботи з великими сторінками, проте програмний код потрібно буде змінювати так як вони не забезпечують прозорий підхід. Існують також алокатори, що підтримують прозоре використання великих сторінок, однак їх застосування обмежене, і вони ще не набули значної популярності в обчислювальних системах.

Алокатори дають змогу використовувати однакові алгоритми розподілу пам'яті у різних операційних системах. Для впровадження нового алгоритму достатньо замінити лише динамічний розподільник, без необхідності змін у самій системі. Варто також зазначити, що реалізація алгоритмів безпосередньо в ОС є складнішим завданням, ніж їх інтеграція в алокатор, через особливості низькорівневої архітектури ОС.

**Особливості використання великих сторінок пам'яті**

Здебільшого сучасні системи, що використовують великі сторінки працюють зі звичайними та великими сторінками розміру 2 МБ, оскільки вони є найефективнішими в процесах з динамічним використанням пам'яті, адже 1 ГБ — здебільшого занадто великий розмір для користувацьких структур даних і через це програми можуть використовувати велику кількість надлишкової пам'яті.

Як уже зазначено вище, поліпшення від використання великих сторінок залежить від того, як процеси виконують доступ до пам'яті та наскільки часто відбувається зміна контексту (див. рис. 4 та рис. 5). Існують певні види навантажень, де і кількість БАТ промахів менша, і час виконання менший. Існують і такі, де лише кількість промахів менша, але по часу виконання співмірно. Проте є такі завдання, де не вдалось зменшити кількість БАТ промахів, а кількість і час виконання, зросли. З огляду на те, що для більшості навантажень відсоток БАТ промахів суттєво зменшився, майбутнє зростання обсягів пам'яті та швидкодії процесорів може призвести до ще більшого скорочення часу виконання.

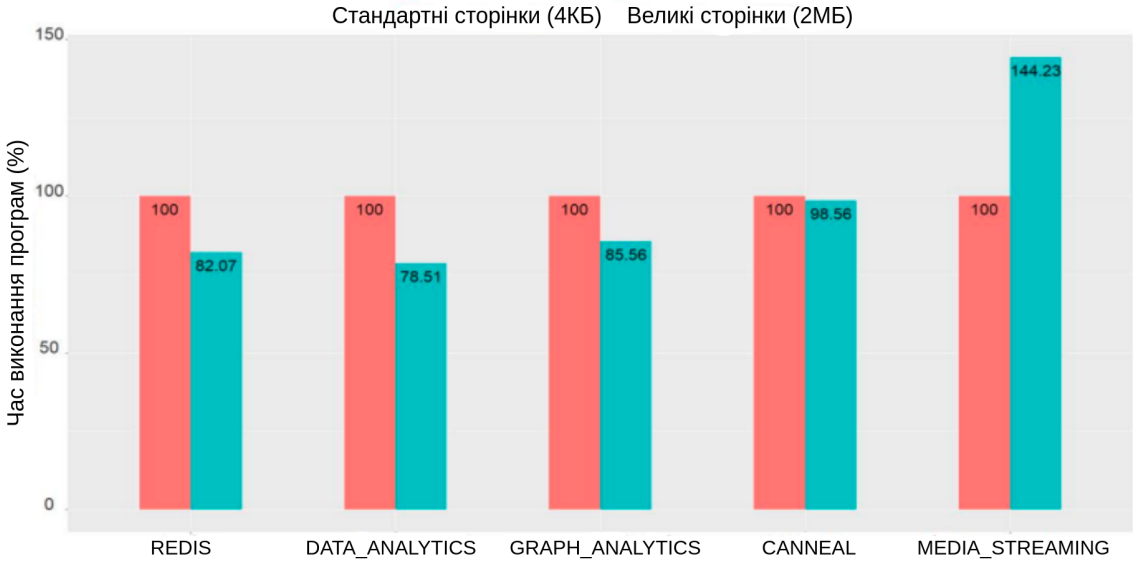


Рис. 5. Вплив великих сторінок на швидкодію в порівнянні зі стандартними сторінками [11].

**6. Висновки.** Використання великих сторінок не завжди ефективне унаслідок впливу фрагментації, недосконалості алгоритмів керування пам'яттю та інші проблемні аспекти цього підходу. Отож подальші дослідження у цій сфері необхідно спрямувати на розробку адаптивних алгоритмів розподілу та керування великими сторінками, які зменшуватимуть вплив фрагментації та полі-

пшуватимуть загальну продуктивність системи.

### Список використаної літератури


1. Basu A. et al. Efficient Virtual Memory for Big Memory Servers. *Proceedings of the 40th Annual International Symposium on Computer Architecture.* 2013. Vol. 41, No. 3. pp. 237–248. DOI: https://doi.org/10.1145/2508148.2485943
2. Bhattacharjee A. Large-reach Memory Management Unit Caches. *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture.* 2013. pp. 383–394. DOI: https://doi.org/10.1145/2540708.2540741
3. Karakostas V. et al. Redundant Memory Mappings for Fast Access to Large Memories. *Proceedings of the 42nd Annual International Symposium on Computer Architecture.* 2015. Vol 43, No. 3S. pp. 66–78. DOI: https://doi.org/10.1145/2872887.2749471
4. Awad A. et al. Avoiding TLB Shootdowns Through Self-Invalidating TLB Entries. *The Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques.* Portland : OR, USA, 09–13 September 2017. pp. 273–287. DOI: https://doi.org/10.1109/PACT.2017.38
5. Kwon Y. et al. Coordinated and Efficient Huge Page Management with Ingens. *The Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation.* 02 November 2016. pp. 705–721. DOI: https://dl.acm.org/doi/10.5555/3026877.3026931
6. Panwar A., Prasad A., Gopinath K. Making Huge Pages Actually Useful. *The Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems.* 2018. pp. 679–692. DOI: https://doi.org/10.1145/3173162.3173203
7. Corbet J. Transparent Hugepages. 2009. URL: https://lwn.net/Articles/359158 (date of access: 15.02.2025).
8. Navarro J. et al. Practical, transparent operating system support for superpages. *USENIX Symposium on Operating Systems Design and Implementation.* 2002. Vol 36, No. SI. pp. 89–104. DOI: https://doi.org/10.1145/844128.844138
9. Evans J. A Scalable Concurrent malloc(3) Implementation for FreeBSD. 2006. URL: https://www.bsdcan.org/2006/papers/jemalloc.pdf (date of access: 15.02.2025).
10. Hunter A. et al. Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator. *The Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation.* 2021. URL: https://www.usenix.org/system/files/osdi21-hunter.pdf (date of access: 15.02.2025)
11. Choi G. et al. HPanal: A Framework for Analyzing Tradeoffs of Huge Pages. *The 34th ACM/SIGAPP Symposium on Applied Computing.* 2019. pp. 1438–1443. DOI: https://doi.org/10.1145/3297280.3297425


**Tarnavskyi P. V., Biletskyy V. M.** Using Large Pages to Improve the Performance of Modern Operating Systems.

A review of current technologies and challenges related to the use of huge memory pages in computing systems. The impact of such pages on the operation of Translation Lookaside Buffers (TLB) and memory management efficiency is discussed. The main issues related to memory fragmentation, increased probability of TLB misses, and decreased performance under high multitasking conditions are outlined. The article also covers the compatibility of huge pages with software and the usage of them in modern Linux distributions.

**Keywords:** random access memory, memory management, translation lookaside buffer, huge memory pages, memory fragmentation.

### References


1. Basu, A., Gandhi, J., Chang, J., Hill, M. D., Swift, M. M., & Venkat, A. (2013). Efficient virtual memory for big memory servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture,* 41(3), 237–248. https://doi.org/10.1145/2508148.2485943
2. Bhattacharjee, A. (2013). Large-reach memory management unit caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture,* 383–394. https://doi.org/10.1145/2540708.2540741

3. Karakostas, V., Alipour, M., Chisnall, D., Jones, T. M., & Watson, R. N. M. (2015). Redundant memory mappings for fast access to large memories. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture,* 43(3S), 66–78. https://doi.org/10.1145/2872887.2749471
4. Awad, A., Kelley, K., & Falsafi, B. (09–13 September, 2017). Avoiding TLB shootdowns through self-invalidating TLB entries. In *Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques.* Portland: OR, USA, 273–287. https://doi.org/10.1109/PACT.2017.29
5. Kwon, Y., Miedl, E., & Swift, M. M. (02 November, 2016). Coordinated and efficient huge page management with Ingens. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation,* 705–721. https://dl.acm.org/doi/10.5555/3026877.3026931
6. Panwar, A., Prasad, A., & Gopinath, K. (2018). Making huge pages actually useful. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems.* 679–692. https://doi.org/10.1145/3173162.3173203
7. Corbet, J. (2009). Transparent hugepages. Retrieved from https://lwn.net/Articles/359158
8. Navarro, J., Iyengar, A., McKinley, K. S., & Burger, D. (2002). Practical, transparent operating system support for superpages. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation.* 36(SI), 89–104. https://doi.org/10.1145/844128.844138
9. Evans, J. (2006). A scalable concurrent malloc(3) implementation for FreeBSD. Retrieved from https://www.bsdcan.org/2006/papers/jemalloc.pdf
10. Hunter, A., Schuh, S., Tarjan, D. R., & McElroy, R. (2021). Beyond malloc efficiency to fleet efficiency: A hugepage-aware memory allocator. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation.* https://www.usenix.org/system/files/osdi21-hunter.pdf
11. Choi, G., Shin, J., Kim, J., & Ryu, J. (2019). HPanal: A framework for analyzing tradeoffs of huge pages. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing,* 1438–1443. https://doi.org/10.1145/3297280.3297425

Одержано 15.02.2025