Nimble Page Management for Tiered Memory Systems

Zi Yan

Rutgers University & NVIDIA

ziy@nvidia.com

David Nellans

NVIDIA

dnellans@nvidia.com

Abstract

소프트웨어가 제어하는 이기종 메모리 시스템은 컴퓨팅 시스템의 성능과 비용 효율을 증가 시킬 수 있는 잠재력을 가지고 있다. 그러나 OS가 효율적인 페이지 관리 정책과 메커니즘이 지원되는 경우에만 시스템을 이행 할 수 있다. 현재의 OS는 이기종 메모리 간의 효율적인 데이터 계층화를 지원하지 않는다. 대신 메모리 메모리 노드 간에 offlining of memory 와 디스크로의 데이터 스와핑에 의존한다. (메모리 노드 간에 핫 또는 콜드 데이터를 profiling하고 migrating한다.) 또한 다양한 최적화를 테이블에 남긴다. 예를 들면 마이그레이션 처리량을 극대화 하기위해 다중 스레드 하드웨어를 활용하지 않아서 사용가능한 메모리 대역폭의 활용도가 최대 95%까지 떨어지게 된다.

이러한 단점을 보완하기 위해, 현재의 OS 페이지 추적구조를 추가 모니터링 오버헤드 없이 메모리 간에 직접 페이지를 계층화 하는 general purpose한 OS 통합 멀티레벨 메모리 매니지먼트 시스템을 제안한다. 이러한 시스템을 위해 4가지 추가적인 최적화가 필요하다:

native support for transparent huge page migration,

(THP 마이그레이션을 위한 네이티브 지원)

multi-threaded migration of a page,

(페이지 단위의 멀티 스레드 마이그레이션)

concurrent migration of multiple pages, and

(여러 페이지의 동시 마이그레이션)

symmetric exchange of pages.

(대칭적 페이지 교환)

Daniel Lustig

NVIDIA

dlustig@nvidia.com

Abhishek Bhattacharjee

Yale University

abhishek@cs.yale.edu

이러한 최적화를 묶어 사용한다면 극적인 커널 소프트웨어 오버헤드 감소와 raw page 마이그레이션 처리량이 15이상 향상 된다. Linux에서 개발되고 x86, Power, ARM 시스템에서 평가 되었다. 이러한 이기종 메모리를 위한 OS support 를 통해 다중레벨의 세분화된 메모리 시스템을 사용하는 실제 메모리 집약적인 워크로드 제품군에 대해 어플리케이션 성능을 40%향상 시켰다.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASPLOS ’19, April 13–17, 2019, Providence, RI, USA

* 2019 Association for Computing Machinery. ACM ISBN 978-1-4503-6240-5/19/04. . . $15.00 <https://doi.org/10.1145/3297858.3304024>

**CCS Concepts • Computer systems organization → Heterogeneous (hybrid) systems; • Software and its engineering → Virtual memory.**

**Keywords** Page migration; Operating system; Heterogeneous memory management

**ACM Reference Format**:

Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In 2019 Architectural Support for Programming Languages and Oper-ating Systems (ASPLOS ’19), April 13–17, 2019, Providence, RI, USA. ACM, New York, NY, USA, [15](#page15) pages. [https://doi.org/10.1145/3297858.](https://doi.org/10.1145/3297858.3304024) [3304024](https://doi.org/10.1145/3297858.3304024)

* Introduction

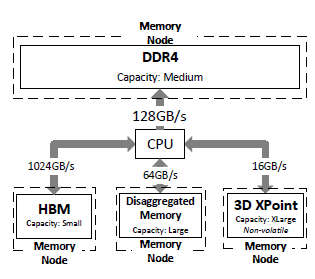
현재 컴퓨터 시스템은 이미 프로세스 및 메모리에서 이기종 시스템을 사용하고 있다. 프로세서는 CPU, GPU 및 가속기를 사용하여 Performance 및 Energy efficiency를 향상시키기 위해 특수화되고 있으며 명령과 데이터 수준에 병렬프로세스를 추진하고 있다. 메모리는 Latency, Bandwidth, Capacity, Cost 문제를 해결하기위한 기술들을 결합하고 있다. 예를 들어 Intel 의 Knight’s Landing은 Multi-channel DRAM (MCDRAM)이라 말하는 높은 대역폭 메모리가 사용되어 높은 대역폭 및 대용량 메모리에 달성되었다. 또한 DDR4 메모리 역시 높은 대역폭과 대용량을 달성했다. Non-volatile(비휘발성) 3D XPoint memory 은 다음 세대 데이터베이스 시스템을 위해 상용화되었으며, 분리된 메모리는 Blade server의 용량 문제를 해결할 솔루션이다. 최근부터IBM, NVIDIA에서 이기종 메모리를 사용하여 고대역폭 GPU 메모리와 대용량 CPU 메모리를 포함한

슈퍼컴퓨터를 공급했다.

Figure 1은 미래에 고려해야할 메모리 시스템 구조와 OS 구조의 추상적인 예를 보여준다. 이러한 시스템은 Varying latency, Bandwidth and/or capacity properties성이 서로 다른 여러 유형의 메모리에 연결된 컴퓨팅 노드(CPU, GPU)로 구성된다.

물론 독립적인 구성은 시스템에 따라 다를 것이다. 프로그램이 잠재적으로 구별되는 작업 세트를 가지고 다른 실행 단계를 탐색할 때 데이터를 효율적으로 이동하는 비전을 가능하게 하는데 필요한 중요한 OS지원은 효율적인 페이지 관리 및 마이그레이션이다. 구성에 상관없이 성능을 최적화하려면 이상적으로는 hottest page를 빠른 메모리 노드(latency or bandwidth)에 가능한 많이 저장해야 한다. Next-hottest page는 second-fastest 노드에 가능한 많이 저장될 것이다. 프로그램을 실행하고 난 후, 이러한 성능을 유지하기 위해서는 계속해서 페이지를 re-organized 해야 한다.

불행하게도 오늘날의 시스템에서 페이지 마이그레이션은 높은 오버헤드를 가지고 있고 놀랄 만큼 비효율적이다. 과거의 작업과 최근의 제안들은 주로 하드웨어 측면에서 오버헤드를 줄이는데 초점을 맞추고 있다. 일부는 마이그레이션된 데이터에 대한 TLB miss 페널티를 줄이기 위해 TLB 커버리지(용량)을 증가시켰다.



|  |  |
| --- | --- |
| Figure 1. A hypothetical future multi-memory system with | |
|
| 4 technology nodes, all exposed as non-uniform memory | |
| nodes to the operating system. |  |

우리는 두개의 메모리 노드 사이에 페이지를 이동하고 각 로컬 노드가 소켓 간 상호 연결보다 더 많은 메모리 대역폭을 사용하는 실험을 수행했다. 별개의 메모리 노드에서 메모리를 할당한 후, 여러 유형의 크로스 소켓 페이지 마이그레이션의 cost 분석과 Throughput을 모두 측정했다. Figure 2 은 오늘날의 Linux에서 각기 다른 크기를 가진 페이지를 마이그레이션을 시도하였을 때 Cost 와 Throughput을 보여주고 있다. Single Base Page이동에서 시간의 대부분을 커널 메모리 관리 및 동기화에 사용한다 실제 페이지 복사에는 적은 시간만이 소비된다. 따라서 19.2GB/s 대역폭을 지원하는 하드웨어를 사용하더라도 효과적인 마이그레이션 처리량은 40MB/s에 불과하다. 또한 우리는 여러 마이그레이션 소프트웨어 오버헤드를 알기 위해 거대한 페이지 크기(2MB)와 일치하는 512개의 페이지로 마이그레이션 되는 페이지수를 조정하였다. 이경우 처리량은 750MB/s이지만 이것 또한 하드웨어 대역폭의 5%에 불과하다. 또한2MB를 포함한 다양한 스레드 수와 데이터 크기로 시스템의 데이터 복사 처리량을 프로파일링 한다. Figure 3은 기존 페이지 마이그레이션 처리량이 2MB 데이터 크기로 달성할 수 있는 것보다 10배느리고 전송된 데이터 크기가 클수록 격차가 크다는 것을 보여준다.

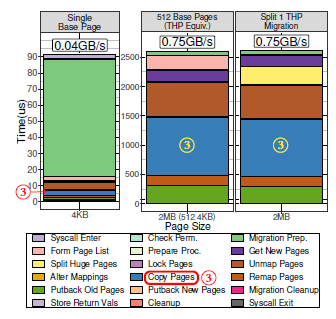


Figure 2. Page migration cost breakdown for migrating a single 4KB page, 512 consecutive base pages, and both splitting and migrating a 2MB THP. (Figure best viewed in color.)

처리량이 2MB 데이터 크기로 달성할 수 있는 것보다 10배느리고 전송된 데이터 크기가 클수록 격차가 크다는 것을 보여준다.

페이지 마이그레이션 병목 현상을 제거하기 위해 우리는 Transparent 대규모 페이지 마이그레이션, 병렬 데이터 복사, 동시적 다중 페이지 마이그레이션, 페이지의 대칭적 교환 등 네가지 최적화를 제안한다. 이러한 메커니즘 위에 기존의 OS active/inactive page list를 사용하여 이기종 메모리 사이에 데이터를 직접 이동시켜 다른 소프트웨어적 오버헤드를 줄여 전체적인 Multi-Level Memory Solution을 build한다. 해당 작업의 기여는 다음과 같다.

1. 우리는 현재 움직일 수 없는 Transparent Huge Page(THP)를 이동가능한 Base pages로 분할하는 것이 THP의 효과를 감소시킨다는 것을 보여준다. 또한 기존 페이지 마이그레이션은 하드웨어 라인 속도(대역폭)보다 훨씬 낮은 처리량만 달성한다는 것을 입증한다. 우리는 기존 리눅스에 비해 마이그레이션 처리량을 2.8배 향상 시키는 것과 동시에 TLB Coverage를 개선하는 부작용을 가지는 Native Huge Page(THP) 마이그레이션을 구현함으로 이 문제를 해결한다.
2. 추가 페이지 마이그레이션 최적화를 통해 기본 THP 마이그레이션 구현에 비해 처리량이 5.2배 향상된다. 이 둘을 합치면 페이지 마이그레이션 처리량이 현재 기술보다 15배 증가한다. 이러한 최적화는 기존 OS 인터페이스를 재사용함으로써 표준 Linux 메모리 관리 API를 사용하는 메모리 관리 Policy에 의해 자동으로 상속된다. 우리의 광범위한 적용가능성에 대한 주장은 우리의 기술들 중 일부가 주류 리눅스로 채택되었다는 사실에 의해 뒷받침된다. 우리는 또한 우리의 모든 최적화를 구현하고 평가에 사용한 실험용 커널을 공개하였다.
3. 간단한 End-to-End 이기종 메모리의 프로파일링 및 Placement Policy를 탐구한다. 기존의 구현 및 기타 최근의 제안과 달리, 우리의 시스템은 데이터를 디스크로 Swap 하지 않으며, 또한 메모리의 한 부분을 page faults를 일으켜 프로파일 액세스에 사용할 수 없게 하지 않는다. 대신 기존의 OS active/inactive page list를 용도 변경하기만 하면 된다. 그러므로 우리의 접근 방식은 이 기능이 필요하지 않을 수도 있는 시스템에 프로파일링 오버헤드를 주지 않는다.
4. 우리는 세분화된 multi-tier 메모리 시스템에서, 우리의 최적화된 OS지원은 멀티 레벨 메모리에 대한 현재의 지원보다 평균적으로 40% 이상의 어플리케이션 성능을 향상시킨다는 것을 보여준다.

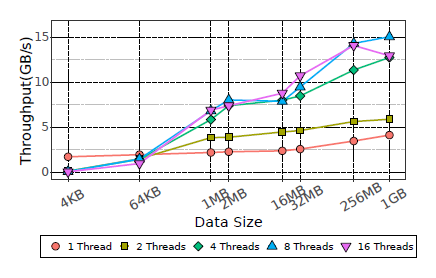


Figure 3. Impact of thread count and transfer size on raw data copy throughput (higher is better)

* Background

현재의 이기종 메모리 시스템은 일반적으로 저용량 고대역폭 메모리에 대용량 저대역폭 메모리로 구성된다. 대용량 메모리의 지연은 더길이지고 잠재적으로 Multi-hop의 물리적 연결 또는 기본 메모리 기술의 차이로 더 높을 것으로 예상된다. 결과적으로 이기종 메모리 시스템은 종종 latency와 bandwidth 모두에 대해 (Non-uniform memory access)NUMA 속성을 가질 것이다. 최적의 성능을 보장하기 위해서 어플리케이션 개발자는 초기 페이지 placement policies 와 후속 페이지 마이그레이션 polices를

* <https://github.com/ysarch-lab/nimble_page_management_asplos_2019>

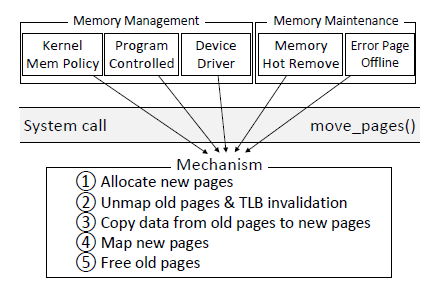


Figure 4. Separation of page migration policy and page migration mechanism in a multi-level memory system.

사용하여 hot date pages가 어플리케이션 런타임 동안 가장 높은 대역폭 또는 가장 낮은 Latency 메모리 노드 내에 유지되도록 할 것이다.

2.1 Page Management Policies and Mechanisms

최근OS는 NUMA를 지원한다. 실제로 페이지 위치 결정을 최적화하여 Throughput을 최대화할 수 있도록 이기종 메모리 속성(Latency, Bandwidth, Durability)을 표준화 하여 운영체제에 전송해 줄 수 있는 시도가 이미 도입되고 있다. 멀티 레벨 메모리에 대한 전체적인 OS 지원을 설명하기 위해서는 메모리 관리 개념적으로 두개의 다른 구성요소로 분리해야 한다. Figure 4는 메모리 관리 정책 결정은 커널, 장치 드라이버, 프로그래머 혹은 시스템 관리자가 선택할 수 있음을 보여준다. 이러한 정책은 페이지 이동 작업을 수행하는 공통 메커니즘인 페이지 마이그레이션 위에 구축된다. 따라서 정책 결정은 Linux의 move\_page()와 같은 system call(혹은 유사한 커널 인터페이스)을 통해 페이지 마이그레이션 메커니즘에서 분리된다.

Figure 4 은 일반적인 페이지 마이그레이션 메커니즘을 보여준다. 1번 새 페이지 할당, 2번 기존 가상, 물리주소 Mapping 해제(많은 아키텍처에서 TLB shootdown 문제 발생), 3번 이전 물리 페이지 데이터를 새 물리 페이지로 복사, 4번 가상주소를 새 물리 페이지에 Mapping, 5번 이전 물리 페이지 Free 이전 페이지와 새 페이지 사이의 실제 복사는 3번 단계에서만 발생한다. 1-2단계, 4-5단계는 페이지 마이그레이션 프로세스 중에 이전 페이지와 새 페이지가 액세스 되지 않도록 정확성과 무결성을 보장하는데 필요한 오버헤드이다. 이 문서는 현대 하드웨어를 활용하고 마이그레이션 처리량을 개선하기 위해 이 다단계 프로세스의 진보에 대해 살펴볼 것이다.

오늘날 리눅스는 autoNUMA를 사용하여 NUMA 노드 간에 메모리와 컴퓨팅 요구사항의 균형을 맞추고 있다. 이를 위해 프로세스 마이그레이션과 페이지 마이그레이션 기술에 의존한다. 안타깝게도 프로세스 마이그레이션은 여러 메모리 노드가 단일 프로세서에 연결된 이기종 메모리 시스템에는 적용되지 않는다. 또한 사용 가능한 공간이 있는 메모리 노드로만 페이지를 마이그레이션하고, 그렇지 않으면 페이지는 디스크로 스왑되기 때문에 페이지 마이그레이션도 현재 다중 레벨 메모리 시스템에서 제한된다. 이것은 가능한 한 많은 페이지를 빠른 메모리에 넣겠다는 시스템의 목표와 상충된다. 또한 페이지 액세스 정보를 얻기 위해 autoNUMA는 프로파일링을 위해 페이지를 오프라인으로 전환하며, 이러한 오프라인화는 예측 불가능한 결과의 원인이 된다. 이 두가지 문제는 Two-level memory에 대한 광범위한 학문적 자극제가 되었다. 그러나 이전 연구는 일반적으로 페이지 마이그레이션 메커니즘이 충분해야 한다고 가정하면서 페이지 마이그레이션 정책에 초점을 맞추고 있다.

2.2 Recent Developments

이기종 메모리는 현제 상업적으로만 채택되고 있기 때문에. 다단계 메모리 페이징은 연구의 활동적인 영역으로 남아있다. 연구자들은 hot/code pages를 식별하고 메모리 장치 간의 이동을 용이하게 하기 위한 하드웨어 기술을 연구하기 시작했다. 이 논문은 복잡한 방식으로 이기종 메모리를 더 잘 관리할 수 있는 소프트웨어 기반 접근법에 대한 추가 연구를 불러 일으킬 것이다. 이러한 연구는 OS 관리형 다중 수준 메모리의 장점을 확립하였지만, 대부분 추적기반 시뮬레이션 (OS효과 제외), 비표준 OS Plumbing 사용(예약 페이지 테이블 엔트리 비트 사용 등),OS에서 Hoist 페이지 마이그레이션 에 의존(소스코드나 금지사항으로 인해) 및/또는 페이지 마이그레이션 중에 OS가 전체 하드웨어 대역폭을 제공할 수 있다고 가정할 뿐이다. 페이지 마이그레이션 메커니즘은 다단계 메모리 시스템의 성능에 중요함에도 불구하고 Policy만큼 광범위하게 연구되지 않았다. 이러한 미묘해 보이는 문제들이 결합되면 정해진 결론이 궁극적으로 불안전하게 보일 수 있기 대문에 많은 이전 제안들의 현실적 채택을 방해할 수 있다. 해결책은 실무에서 채택될 수 있어야 한다. 그러나, 이렇게 증가하는 관련 업무는 OS가 현재 우리가 다루고있는 다단계 메모리 시스템에 어떤 영향을 미칠지에 대한 시스템 수준 평가 외에 더 나은 페이지 마이그레이션 메커니즘과 정책 둘 다를 필요로 한다.

* Native OS Support for Multi-Level Memories

Multi-level Memory 시스템에 대한 전체적 지원에는 Intelligent한 메모리 관리 정책과 효율적인 Page Migration Mechanism이 모두 필요하다. 이 섹션에서는 메모리 관리 시스템의 end-to-end benefit을 입증하기 위해 섹션 4에서 사용하는 특정 하위 오버헤드 정책뿐만 아니라 하나의 특정 정책과는 무관한 Page Migration Mechanism 개선 사항을 제시한다.

3.1 Optimizing Page Migration Mechanisms

효율적인 Page Migration Mechanism을 구현하기 위해 OS 내에서 4가지 중요한 문제를 해결해야 한다 :

**Larger data size**(데이터 크기가 크면 페이지 마이그레이션 소프트웨어 오버헤드를 상각할 수 있다.),

**Multiple threads**(오늘날 페이지 마이그레이션은 주로 단순성을 위해 단일 스레드 방식으로 이루어 지지만, 여러 스레드를 사용하면 복사 시간 자체를 단축할 수 있다),

**Concurrent Migrations**(여러 마이그레이션을 동시에 수행하면 Figure 2에서 볼 수 있는 Amdahl 법칙 병목 현상을 피할 수 있다. 아키텍처는 매우 제한된 페이지 크기 집합만 지원하며 일반적으로 가장 큰 페이지 크기(예 x86의 1GB)는 transparently하게 지원되지 않기 때문에 이 문제는 더 큰 페이지를 단순히 사용한다고 해서 해결될 수 없다.

**Efficient two-sided migration**(페이지가 빠른 메모리로 마이그레이션 될 때, Victim 페이지를 느린 메모리로 마이그레이션 하여 공간을 만들어야 한다.) allocation / deallocation 을 완료하고 단순히 페이지를 교환함으로써 Two-sided operation은 두 개의 단방향 마이그레이션을 합한 것보다 더 빠를 수 있다.

우리는 아래의 각 문제를 차례대로 다룬다.

3.1.1 Native THP Migration

첫 번째 최적화는 기본 THP Migration을 구현하는 것이다. THP Migration을 수행하면 페이지를 분할하지 않고 필요한 TLB invalidation 및 shootdown횟수를 줄임으로써 THP 마이그레이션의 하드웨어 및 소프트웨어 오버헤드가 모두 512배 감소한다. 또한 단일 마이그레이션 작업 내에서 마이그레이션되는 데이터의 양을 증가시킨다. 명백하게 보일 수 있지만,Linux(및 많은 OS)의 THP에 대한 Page Migration 지원은 아직 완료수준 이거나, 일반적이거나, 고성능이 아니다. Page Migration은 원래 THP가 도입되기도 전에 NUMA 시스템 성능을 향상 시키고 Memory Hotplug를 기능적으로 달성하기 위해 제안되었다.

예를 들어 오늘날 리눅스는 다시 응답하는 THP를 프로그래머의 요청에 따라 데이터를 특정 메모리 노드로 이동하도록 설계된 mbind() 및 move\_page()와 같은 프로그래밍된 Resource Management Requests으로 마이그레이션 할 수 없다. 프로그래머가 주도하는 데이터 배치가 우수한 성능을 발휘할 수 있는 중요한 상황이 있기 때문에 이것은 이기종 메모리의 심각한 단점이다. 마찬가지로 리눅스는 Hot removal, soft off-lining 또는 cpuset/cgroup에 대응하여 직접 THP를 마이그레이션할 수 없다. 이 모든 경우에 리눅스는 THP가 포함된 가상메모리 범위를 마이그레이션할 때 THP를 분할하고 대신 Base page페이지를 마이그레이션 하므로 페이지 마이그레이션 성능이 저하되고 TLB적용 범위가 감소한다.

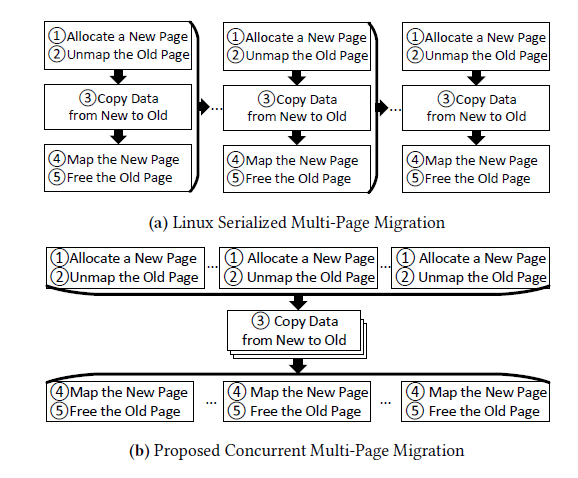


Figure 5. Improvements to Linux multi-page migration to enable large transfers for improved copy bandwidth.

THP 마이그레이션을 구현하기 위해 THP를 인식하도록 Figure 4에 표시된 5 단계를 보강합니다. THP 구현은 모든 자원 관리 요청 (예 : mbind, cpuset 등)을 지원합니다. 또한 THP가 마이그레이션 될 때 THP가 마이그레이션 될 수 있다는 사실을 설명하기 위해 커널의 다른 THP 특정 코드 경로를 조정합니다. 요구. 마이그레이션이 끝날 때까지 기다리거나 기본 페이지에서 와 같이 THP를 건너 뛰어 이를 수행합니다. 다른 OS는 동일한 원칙에 따라 THP 마이그레이션을 실현하여 이기종 메모리 시스템을 보다 잘 지원하기 위해 처리량이 많은 페이지 마이그레이션을 가능하게 합니다.

3.1.2 Parallelized THP Migration

현재 리눅스 페이지 마이그레이션 루틴은 단일 스레드이며 단일 마이그레이션 내에서 전송하는 데이터의 양(일반적으로 하나의 기본 페이지 크기)이 제한되어 있다. Figure 3에서 영감을 받아 페이지 마이그레이션 작업 내에서 가변 스레드 수 기반 복사 서브 루틴을 구현했다.

Linux의 move\_page() system call 호출 내에서 다중 스레드 페이지 복사를 하기위해, Kernel work Queues를 사용하여 임의의 물리적 범위 간에 데이터를 복사하기 위해 helper thread를 생성한다. 우리의 구현에서는 페이지 크기(또는 동시에 마이그레이션되는 페이지 )를 작업자 스레드 수로 나누어 각 병렬 스레드를 통해 복사할 데이터의 양을 계산한다.

처리량을 최대화하는데 필요한 스레드 위치 및 개수의 정확한 선택은 시스템마다 다를 수 있으므로 sysfs 인터페이스를 통해 Parameter configuration을 제공하므로 시스템 관리자는 멀티 스레드 CPU 복사를 활성화 또는 비활성화 하거나 관련 CPU수를 변경할 수 있습니다. 또한 마이그레이션 정책 엔진이 마이그레이션마다 병렬 수준을 동적으로 선택할 수 있도록 Optional parameter flag(MPOL\_MF\_MT)를 사용하여 move\_page() system call을 보강한다.

3.1.3 Concurrent Multi-page Migration

Multi-page Migration 은 spatial locality과 prefetching 효과로 인해 Multi-level Memory system에서 공통적으로 이루어질 것으로 예상된다. Linux의 move\_page() 인터페이스는 메모리 노드 간에 마이그레이션할 페이지에 대한 포인터 목록을 전달하여 단일 system call로 여러 페이지를 이미 마이그레이션 할 수 있도록 지원하고 있다. 단 Figure 5 (a)와 같이, 현재의 구현은 사본을 직렬화 하여 한 번에 한 페이지씩 실시한다.

새로운 Page Migration의 구현은 Figure5 (b)와 같이 모든 데이터복사 프로시저를 하나의 더 큰 논리적 단계로 집계하여 move\_pages()에 제공된 목록의 모든 페이지를 동시에 Migration합니다. 예를 들어 16개의 THP를 2MB 를 마이그레이션 하는 경우를 고려하겠습니다. 현재 Linux구현에서 병렬복사 최적화를 사용하더라도 Linux는 각 병렬 2MB복사본 사이에 암시적 장벽을 가지고 16개의 THP를 2MB로 전송합니다. Concurrent migration 최적화에서 목록의 각 페이지에 크기가 일치하는 새 페이지가 할당되고 할당된 다음 매핑이 해제됩니다. 그런 다음 목록의 모든 페이지가 sysfs 구성에 따라 CPU별 Workqueue에 할당됩니다.

Concurrent migration보다 많은 병렬 전송 스레드를 사용하여 마이그레이션 할 수 있는 경우 구현시 여러 스레드를 사용하여 단일 페이지의 다른 부분을 복사하여 처리량을 최대화합니다. Concurrent page copy단계가 완료되면 새 페이지가 올바른 페이지 테이블 항목에 맵핑되고 이전 페이지가 해제됩니다. 다른 페이지 마이그레이션 단계를 병렬화 할 수도 있습니다. 그러나 아키텍처 종속적인 페이지 테이블 조작을 포함하여 동기화에 대한 정확한 요구사항이 있기 때문에 최적화가 너무 공격적이면 failure recovery가 복잡해집니다.

3.1.4 Symmetric Exchange of Pages

Multi level memory system에서 single-end page migration 은 일반적인 경우가 아닙니다. 고 대역폭 메모리는 일반적으로 큰 저 대역폭 메모리에 비해 용량이 제한됩니다. 따라서 정상 상태에서 페이지를 상위 레벨 메모리 노드로 마이그레이션 할 때 페이지 마이그레이션 정책은 실제 메모리 용량을 초과하지 않도록 해당 노도에서 페이지를 마이그레이션 해야합니다. 따라서 소프트웨어 제어 캐시로 고성능 이기종 메모리 노드를 관리 할 때 각 Hot page 삽입에는 대칭적인 cold page 제거가 필요합니다.

Naïve two-step, 단방향 마이그레이션은 오늘날 OS에서처럼 두 사본이 잠금으로 보호되고 직렬로 실행되는 경우 시스템 하드웨어를 비효율적으로 사용합니다. Figure6 (a)는 이 일반적인 2단계 페이지 마이그레이션 작업을 보여줍니다. 첫째, 잠금 직렬화는 앞에서 설명한 최적화의 이점을 제한합니다. 둘째, 각 마이그레이션 작업은 독립적인 물리적 페이지 할당 및 할당 해제를 수행해야 하며 둘 다 Cost가 높은 software overhead입니다.

이러한 오버헤드를 제거하기 위해 두 가지 단방향 마이그레이션 작업을 새로운 단일 대칭 교환작업으로 결합 할 것을 제안합니다. 우리의 구현은 페이지를 교환함으로써 단방향 페이지 마이그레이션에 필요한 많은 커널 작업을 제거하고 새로운 페이지를 할당하는 대신 기존 물리적 페이지를 재사용합니다.(Figure6 (b)참조)

우리는 동일한 크기의 두 페이지를 수용하는 새로운 exchange\_pages() system call을 사용하여 Linux에서 대칭 페이지 마이그레이션을 구현합니다. 페이지 목록이 요구 사항을 충족하지 않으면 호출자는 전통적인 2단계 마이그레이션 프로세스로 되돌려야 합니다. 두 개의 대칭 페이지 목록과 함께

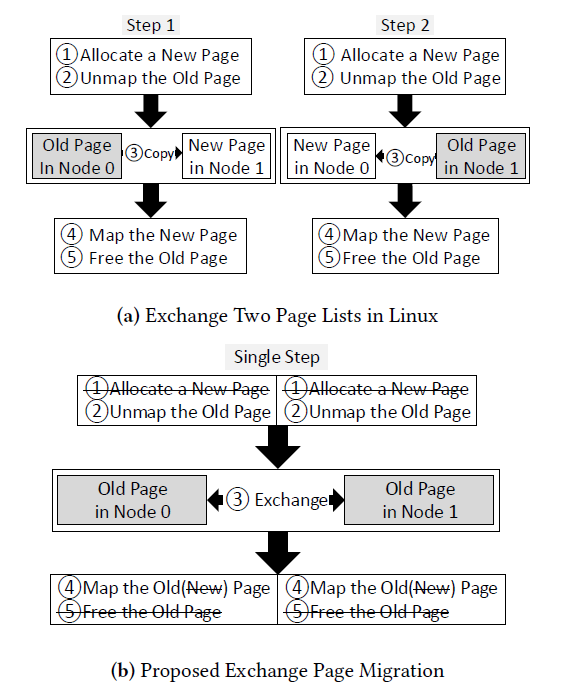


Figure 6. Exchanging pages improves efficiency by eliminating memory allocation and release when migrating symmetric page lists between memory nodes.

호출되면 교환의 구현은 단방향 페이지 마이그레이션과 유사한 경로를 따릅니다. 차이점은 새로운 페이지가 할당되지않고 새로운 페이지로 데이터를 복사하는 대신 CPU 레지스터를 반복적 데이터 교환을 위한 임시 저장소로 사용하는 복사 스레드를 사용하여 각 페이지 쌍간에 데이터를 전송한다는 것입니다. 이러한 레지스터를 사용하면 메커니즘이 완전한 임시 페이지를 할당하지 않아도 됩니다.

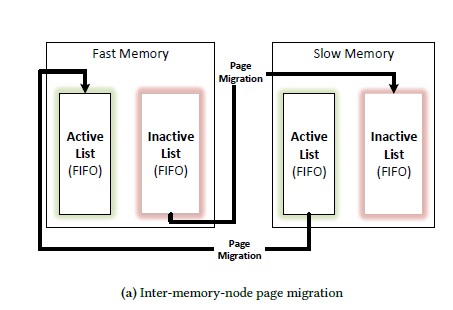
Parallel page copy, concurrent page migration 최적화는 데이터 복사 자체를 개선하는데 중점을 두는 반면 대칭 페이지 교환은 페이지 마이그레이션 중에 발생하는 Cost가 높은 두가지 software overhead(page allocation 및 release)를 제거합니다. 이러한 커널 작업은 페이지 크기에 관계없이 일정한 시간을 소비하므로 페이지 교환은 기본 페이지와 THP 의 마이그레이션 처리량을 향상시킵니다. 또한 각 Parallel exchange thread가 isolation된 상태로 작동하는 한 별도의 locking없이 Parallel exchange (Section 3.1.2)와 Concurrent exchange(Section 3.1.3), 또는 둘 다를 사용하여 페이지 쌍을 교환할 수도 있다.

3.2 Optimizing Page Tracking and Policy Decisions

Multi level memory paging policy and system은 실제 시작되는 다양한 종류의 이기종 메모리 시스템에 광범위하게 유용하기 위해서는 실제 시나리오를 충분히 일반적이고 대표해야 합니다. 이를 위해 우리는 간단하고 광범위한 환경에서 잘 작동하는 것으로 보이는 페이지 마이그레이션 정책을 탐색하고 기준에 무시할 수 있는 오버헤드를 추가합니다. 따라서 우리의 구현은 기존 Linux page replacement 알고리즘을 기반으로 합니다. 우리는 구현이 Upstream Kernel과 최대환 호환되도록 하기 위해 가능한 한 적은 변경을 합니다.

Page replacement 알고리즘의 목표는 page migration 메커니즘이 hot page를 (과거에)disk나 (현재의)느린 메모리 및 빠른 메모리로 마이그레이션 할 수 있도록 hot page와 cold page를 식별하는 것입니다. 리눅스는 이미 각 메모리 노드의 Hot page와 Cold page를 active / inactive 목록으로 분리하여 이를 달성합니다. 여기서 페이지는 한 번에 하나의 목록에만 있을 수 있습니다. Hot page가 Cold화 되고 그 반대의 경우도 Kernel은 Figure 7과 같이 목록사이에서 Page를 능동적으로 이동시킵니다.

Linux와 마찬가지로 우리의 Policy는 각 페이지의 상태와 페이지를 가리키는 페이지 테이블 항목과 커널이 유지 관리하는 페이지 메타 데이터의 두가지 액세스 비트를 확인하여 페이지를 한 목록에서 다른 목록으로 옮깁니다. 우리는 전자를 하드웨어 액세스 비트, 후자를 소프트웨어 액세스 비트라고 부릅니다. 페이지 테이블 항목의 액세스 비트는 첫 번째 TLB miss의 하드웨어 페이지 테이블 워커가 해당 페이지에 해당하는 각 가상-물리 변환으로 설정됩니다. 소프트웨어 액세스 비트는 각 물리적 페이지에 대한 기존 Linux Paging 알고리즘에 의해 설정됩니다. 하드웨어 및 소프트웨어 액세스 비트는 Figure 7 에서 “Ignored”으로 표시된 경우를 제외하고 atomic operation test\_and\_clear()를 사용하여 검사합니다. 이 경우 비트가 확인 check 되거나 clear 하지 않습니다.

우리의 페이징 정책과 표준 Linux 접근 방식의 주요 차이점은 Figure7의 회색 상자 그래픽으로 표시되어 있습니다. 오늘날 Linux에서는 최근에 액세스하지 않은 Cold page를 Reclaim(freed or paged to disk) 할 수 있습니다. 그러나 이기종 메모리 시스템은 나중에 디스크에서 페이지를 다시 페이징하는 높은 비용을 피하기위해 이러한 페이지를 느린 메모리에 percolate하는 것을 목표로 합니다. 결과적으로 우리의 정책은 이 페이지를 inactive list 에 유지하여 빠른 메모리 에서 마이그레이션 할 수 있도록합니다. 마찬가지로 빠른 메모리에서 용량을 사용할 수 있는 경우(예를 들어 메모리 할당해제 또는 비활성화 페이지 마이그레이션) 느린 메모리 active list의 페이지가 빠른 메모리로 마이그레이션 됩니다. 빠른 메모리가 

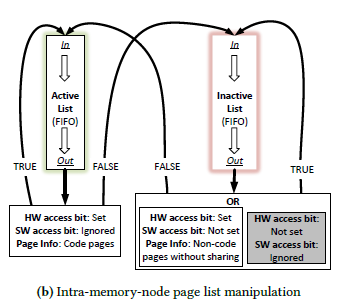


Figure 7. Proposed native multi-level paging policy consisting of (a) inter-memory node page migration and (b) intra-memory node page list manipulation. The former migrates hot pages (in the active list) from slow to fast memory and vice versa for cold pages (in the inactive list). The latter moves pages within a given memory node from one tracking list to the other.

가득 차서 inactive 페이지가 없으면 마이그레이션이 수행되지 않습니다.

기존의 NUMA할당 정책과 유사하게 새 메모리가 할당 될 때 여유 공간이 있으면 빠른 메모리에 할당합니다. 그렇지 않으면 느린 메모리 노드에 할당합니다. 할당 시 페이지를 제거하지 않으므로 메모리 마이그레이션 경로에서 페이지 마이그레이션을 유지할 수 있으므로 성능이 중요합니다. 마지막으로 우리의 시스템은 응용 프로그램 런타임동안 5초마다 페이지 위치를 최적화하여 프로파일링 결과에 따라 프로세스 간섭을 최소화합니다.

Intel Xeon Dual Socket System

|  |  |  |
| --- | --- | --- |
| Processors | 2-socket E5–2650v3 | |
| Memory | DDR4 — 2133MHz | |
| Cross-socket QPI BW | 19.2 | GB/s |
| Memory BW | 34.0 | GB/s (per-socket) |
| Memory Latency | 84.9 ns | |
| OS & Kernel | Debian Buster — v4.14.0 | |
| Disaggr Mem BW (Emulated) | 17.0 | GB/s |
| Disaggr Mem Latency (Emulated) | 199.2 ns | |

Table 1. Overview of experimental system.

4 Experimental Results

구체적인 시나리오에서 최적화된 페이지 마이그레이션 유틸리티를 수량화 하기위해 업계에서 이러한 시스템의 중요성이 커지면서 disaggregated memory system에 대한 접근 방식을 평가한다.

4.1 Methodology

두개의 메모리 노드가 존재하는 환경을 사용하여 분리된 메모리 시스템을 에뮬레이트 합니다. 하나는 Fast Local Memory, 다른 하나는 Slow Remote Memory를 정의합니다. 이전 연구에서 시스템을 로드할 때 사용 된 Artificial Memory Intensive Workload 인 하나 이상의 memhog 인스턴스를 실행하여 느린 메모리를 에뮬레이트 합니다. Idle 상태인 CPU에서 실행되는 memhog 인스턴스는 시스템에 추가 메모리 트래픽을 주입합니다. 이는 Remote Memory 대역폭을 Local Memory 대역폭의 절반으로 줄이고 언로드된 액세스 대기 시간을 증가시켜 Local Memory의 두배로 늘리는 효과가 있으며 이는 Intel Memory Latency Checker에 의해 검증되었습니다. Table 1은 설정에 대한 추가 세부 사항을 보여줍니다.

OS 최적화를 평가하기 위해 제안된 최적화를 Linux v4.14에 통합하는 작업을 했습니다. Git diff가 제공한 커널 수정 통계는 23개 파일 변경, 627개 삽입(+), 114개 삭제(-) 입니다.

평가 자체를 위해 다양한 시험을 수행합니다. 먼저 제안된 최적화 각각의 분리 및 조합 효과를 측정하기 위해 일련의 마이크로 벤치마크를 사용합니다. 둘째, 완벽한 End to End 성능 수치를 얻기 위해 SpecAC-CEL 및 graph500의 워크로드를 실행하고 다양한 고속 메모리 추가에 따른 시나리오에서 성능을 보여줍니다. 셋째, 디자인 공간을 스윕하여 발생하는 흥미로운 동작을 강조하고 가장 성능이 좋은 구성 매개 변수를 식별합니다. 마지막으로, 제안 된 개선 사항의 일반성을 입증하기 위해 x86 이외의 추가 아키텍처에서 이를 평가합니다.

4.2 Page Migration System Call Performance

페이지 마이그레이션 메커니즘을 통해 달성한 전반적인 성능 향상의 원천에 대한 직관을 구축하기 위해 먼저 마이크로

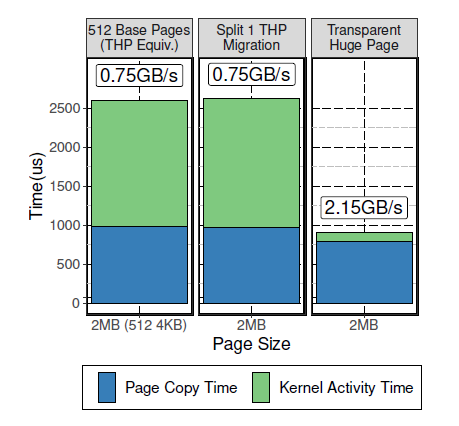


Figure 8. Cost breakdown (lower is better) of 512-basepage migration, THP-split migration, native THP migration.

벤치마크를 사용하여 다양한 최적화의 상대적인 이점을 알아 봅시다. 이 실험에서는 일반 페이지 마이그레이션 인터페이스 move\_pages() 및 새로 제안된 system call인 exchange\_pages()를 사용합니다.

4.2.1 Native THP Migration

Figure 8는 페이지를 세가지 방식으로 마이그레이션하는 커널의 성능을 나타내고 대조하고있습니다. 가장 왼쪽 막대는 수정되지 않은Linux 마이그레이션 512개의 4KB 크기를 가진 페이지입니다. 가운데 막대는 2MB 크기 THP를 4KB 페이지로 분할하여 마이그레이션한 Linux 마이그레이션입니다. 오른쪽 막대는 Native THP Migration 한 결과입니다.

2MB THP를 분할하여 마이그레이션 하면 512개의 4KB 페이지를 마이그레이션하는 것과 거의 동일한 Throughput이 달성됩니다. 512개의 페이지에 대해 동일한 커널 작업과 하나의 추가 THP 분할을 수행하기 때문에 이는 놀라운 일이 아닙니다. Native THP Migration은 512개의 분할 페이지 작업을 단일 페이지 작업으로 통합하여 커널 오버헤드를 줄이므로 Throughput을 2.9배 향상시킵니다. Figure 8은 또한 페이지 복사 시간이 약간만 감소함을 보여줍니다. 즉, 전체 처리량을 향상시키기 위해 추가 최적화를 수행할 여지가 여전히 남아 있습니다.

4.2.2 Multi-threaded Transfers

Figure 9는 두 번째로 최적화된 병렬(Multi Thread) 페이지 복사 결과를 보여줍니다. 2MB THP(위의 그래프)와 4KB 기본 페이지 (아래의 그래프)를 마이그레이션 하는 경우에도 결과를 복사하는데 사용되는 스레드 수가 달라집니다.

멀티 스레드 복사 결과에는 두 가지 주요 요소가 있습니다. 첫째, 페이지 크기가 크면 Parallel Page Copy가 주로 유리합니다. 예를

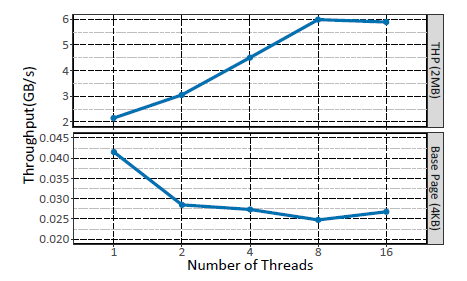


Figure 9. Throughput (higher is better) of multi-threaded single page migration for both base page (4KB) and THP (2MB).

들어 2MB THP 페이지 마이그레이션에서는 속도에서 2.8배 증가가 있었습니다. 하지만 스레드 실행 오버헤드가 충분히 분할 될 수 없기 때문에 4KB 기본 페이지의 처리량은 향상시키지 못합니다. 이 오버헤드는 현재 Linux 페이지 마이그레이션이 기본 페이지 크기에 관계없이 단일 스레드로 유지 된 이유일 수 있습니다. 두 번째, Figure 9의 위에 있는 그래프는 병렬 마이그레이션이 2MB THP에 유용하지만 전체 처리량이 여전히 최대 16GB/s 의 최대 소켓 복사 처리량 보다 훨씬 낮음을 보여줍니다 . (Figure 3 참조) 이는 Concurrent Page Migration의 필요성을 보여줍니다.

4.2.3 Concurrent Page Transfers

Figure 10은 가능할 때 마다 Concurrent Page Migration 사용의 성능적 이점을 보여줍니다. 2MB THP(위의 그래프)와 4KB 기본 페이지에 대해 결과가 다시 분리됩니다. 4KB 페이지에서 사용된 병렬 비 동시 마이그레이션은 단일 스레드 마이그레이션 (상기 언급된 병렬 오버헤드로 인해) 및 병렬 동시 마이그레이션 보다 항상 열등합니다. 병렬 동시 마이그레이션은 충분히 많은 페이지 수에서 기준을 초과합니다. 앞에서 언급했듯이 OS의 오버헤드가 너무 커서 극복 할 수 없습니다. 그러나 병렬 처리와 동시성을 모두 사용하면 2MB THP를 전송할 때 확실한 이점을 얻을 수 있으며 전송되는 페이지 수 에 따라 10~15%범위의 성능 이점(병렬 처리만 가능)이 있습니다.

4.2.4 Symmetric Exchange Pages

Figure 11은 이적 최적화에서 대칭 페이지 교환의 이점을 보여줍니다. THP를 사용하는 경우 교환 페이지 처리량은 동시 페이지 마이그레이션과 유사한 추세를 따르지만 교환되는 페이지 수에 따라 10~50%범위의 성능이 향상됩니다. 흥미롭게도 적은 수의 페이지를 교환 할 때 가장 큰 차이의 결과가 나타납니다. 이러한 경우 소프트웨어 오버 헤드는 총 전송 시간의 상당 부분을 유지하기 때문입니다. 페이지 마이그레이션 프로세스에서 메모리 관리 오버헤드를 제거하면 512개의 2MB페이지의 두 목록( 각 목록의 1GB 데이터)을 교환 할 때 페이지 마이그레이션 처리량을 11.2GB/s 로 향상 시킬 수 있습니다. 이것은 달성가능한 최대 복사 처리량 11.7GB/s (커널 오버헤드 제외)에 매우 가깝습니다.

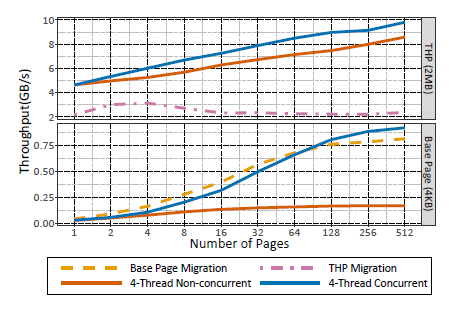


Figure 10. Throughput (higher is better) of concurrent page migration for both base page (4KB) and THP (2MB) with different numbers of pages under migration. 4-Thread Non-concurrent uses 4-thread data copy and 4-thread Concurrent adds concurrent page migration. Single-threaded Base Page Migration and THP Migration are shown for reference.

이전 최적화와 달리 4KB 기본 페이지를 교환 할 때 처리량 개선도 관찰됩니다. 512개의 4KB 페이지 (각 목록에서 2MB데이터)의 두 목록을 교환 할 때 처리량이 1.1.GB/s 이고 Linux 기본 페이지 마이그레이션 처리량보다 37.5% 더 높습니다.

4.2.5 Microbenchmark Summary

기본 페이지와 3개의 비 THP 향상 (멀티 스레드 복사, 동시 복사, 양방향 교환)만 사용하는 경우 시스템은 Linux의 단일 스레드 구현과 비교하여 1.4배의 처리량 향상을 제공합니다. THP 마이그레이션의 경우 기본 THP 마이그레이션 만(즉 병렬/동시/교환 최적화 기능 없이)사용한다면 Linux의 최신 기술보다 마이그레이션 처리량이 2.9배 향상됩니다. 병렬 복사 및 동시 페이지 마이그레이션 최적화 기능을 추가한다면 Native THP 마이그레이션 보다 처리량이 4.6배 향상 됩니다. 양방향 교환을 추가한다면 처리량을 1.1배 더 향상시킵니다. Linux 기준을 통한 THP 마이그레이션에 대한 전체적인 향상은 THP 분할 마이그레이션에 비해 5.2배 기본 페이지 전용 마이그레이션에 대해서는 15배 입니다.

4.3 End-to-End Performance Results

지금까지 마이크로 벤치 마크를 사용하여 페이지 메커니즘 최적화의 성능 이점을 도식화 했습니다. 이제 낮은 오버헤드 페이지 관리 정책과 함께 이러한 최적화가 달성할 수 있는 최종적 개선 사항을 평가하는데 관심을 기울입니다. Disaggregated Memory를 모방한 실험적인 테스트 상황은 4.1 Section에 설명되어 있습니다. Linux의 THP 할당 정책은 변경되지 않았으며 가능하면 운영 체제의 요청에 따라 THP가 제공됩니다.

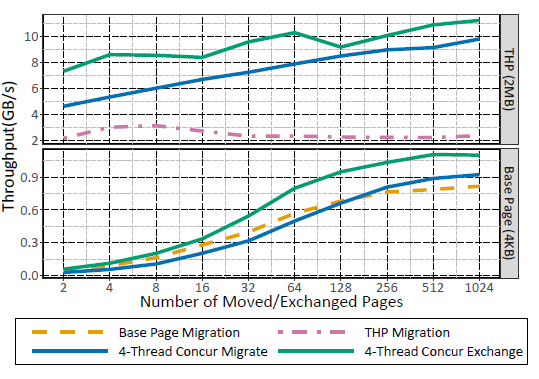


Figure 11. Throughput (higher is better) of page exchange vs. 2 page migrations for both base page (4KB) and THP (2MB) sizes while varying the number of pages exchanged. 4-Thread Concur Migrate and 4-Thread Concur Exchange use both concurrent and 4-thread parallel data copy. Single-threaded Base Page and THP Migration throughput are shown for reference.

우리는 메모리 공간을 32GB로 확장한 SpecACCEL 및 Graph500 으로 평가를 합니다. 실험 설정에서 이러한 워크로드를 실행 하면 각 워크로드에 대한 페이지의 90%이상이 일반적으로 THP에 의해 지원되므로 마이그레이션 전에 THP를 기본페이지로 분할해야 한다는 부정적인 영향이 있음을 나타냅니다.

먼저 16GB 로컬 메모리와 40GB 원격 메모리가 있는 분리된 메모리 시스템 시나리오에서 각 워크로드를 실행합니다. 이 구성에서 로컬 메모리는 워크로드 메모리 풋 프린트 크기의 절반에 불과하며 원격 메모리는 전체 워크로드 풋 프린트를 수용할 수 있습니다. 비교를 위해 상한 및 하한과 함께 네 가지 페이지 마이그레이션 메커니즘을 비교합니다.

1. **All Remote** 원격 메모리에서 워크로드가 완전히 실행하도록 한다. (Lower Bound)
2. **Base Page Migration**, Linux 기본 값(THP 는 분할되어 전송)
3. **Exchange Base Page**, 4-Thread Parallel Copy, 512 Page Concurrent Exchange (THP는 분할되어 전송)
4. **Native THP Migration**(병렬, 동시, 교환 제외)
5. **Exchange Page**, THP migration, 4-Thread Parallel Copy, 8 Page Concurrent Exchange
6. **All Local** 로컬 메모리에서 워크로드가 완전히 실행하도록 한다. (Upper Bound)

Configure 3 및 5에서는 마이그레이션 매개 변수에 대해 512 및 8페이지를 각각 복사하기위해 4개의 스레드를 사용합니다. 마이크로 벤치 마크 결과에서 이것은 기본 및 THP 마이그레이션에 가장 적합한 구성을 나타냅니다. Section 4.5에서 우리는 이러한 선택을 정당화하기 위해 추가 민감도 분석을 제시한다.

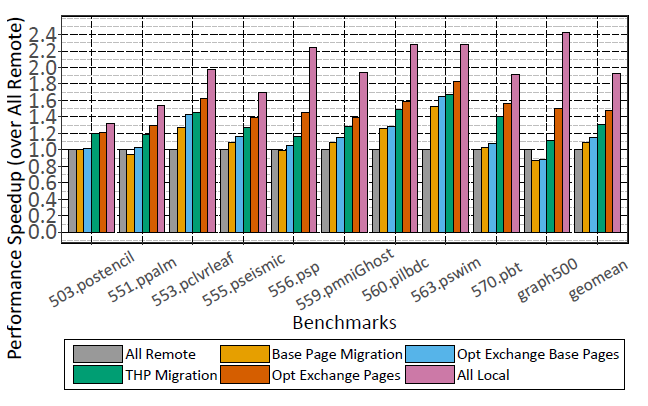


Figure 12. Benchmark runtime speedup (over All Remote, higher is better) with 16GB local memory. Base page migra-tion and THP migration are single-threaded and serialized and shown for comparison, while Opt Exchange Base Pages uses 4-thread parallel and 512-page concurrent migration and Opt Exchange Pages use 4-thread parallel and 8-page concurrent migration.

Figure 12는 All Remote에 비교해서 이 6가지 구성의 상대적 속도를 보여줍니다. 이상적인 경우인 All Local은 All Remote에 비해 평균 2배의 평균 속도를 달성하여 시스템의 로컬 대 원격 메모리 대역폭 및 액세스 대기 시간 차이를 반영합니다. Disaggregated Memory를 관리하기 위한 기준인 Base Page Migration은 평균적으로 9%의 워크로드 성능을 향상시킵니다. 그러나 일부 워크로드(예:551.ppalm, 556.psp 및 graph500)은 All Remote보다 성능이 나빠서 기본 페이지 마이그레이션이 항상 16GB 로컬 메모리보다 빠르게 작동하지는 않습니다. Exchange Base Page 는 워크로드 성능을 평균 16% 향상시킵니다. 이 경우 graph500만 16GB의 로컬 메모리보다 빠르게 활용하지 않습니다. 이 두가지 결과는 더 빠른 메모리 계층을 추가하더라도 열악한 마이그레이션 메커니즘이 실제로 성능을 저하시킬 수 있다는 개념에 대한 증거입니다.

다행이도 Native THP Migration을 사용하면 실제로 빠른 메모리의 이점을 활용할 수 있습니다. THP Migration을 사용하면 All Remote에 비해 Geomean 워크로드 성능이 31%향상되고 Geomean All Local 성능의 68%가 이상적인 사례입니다. Base Page및 THP 모두 마이그레이션 할 수 있는 Exchange Page 는 All Remote에 비해 평균 성능이 48%향상되고 이상적인 All Local 성능의 77%를 달성합니다.

4.4 Sensitivity to Local Memory Size

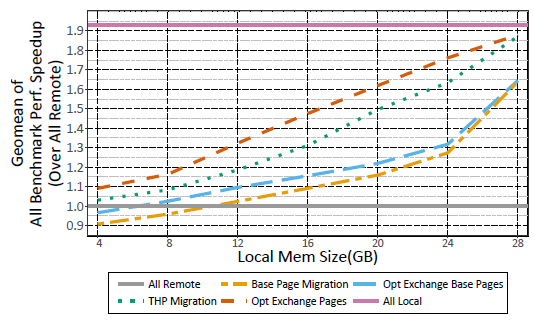
이 접근 방식의 일반적인 적용 가능성을 추가로 보여주기 위해 로컬 메모리 크기를 4GB에서 28GB로 스윕하고 All Remote에 대한 모든 벤치 마크 속도의 기하평균을 보여줍니다. Figure 13 은 성능 추세가 16GB 로컬 메모리 케이스의 성능 추세와 유사하며 4가지 주요 관찰 사항에 주목합니다. 첫째, Linux의 기본 페이지 마이그레이션은 분리된 메모리의 전체 잠재력을 활용할 수 없습니다. 로컬 메모리 크기 (예 4GB 및 8GB)가 워크 로드의 32GB 

Figure 13. Geomean speedup (over All Remote, higher is better) over a sweep of local memory sizes, from 4GB to 28GB. Base page migration and THP migration are single-threaded and serialized, while Opt. Exchange Base Pages uses 4-thread parallel and 512-page concurrent migration, and Opt. Exchange Pages uses 4-thread parallel and 8-page concurrent migration.

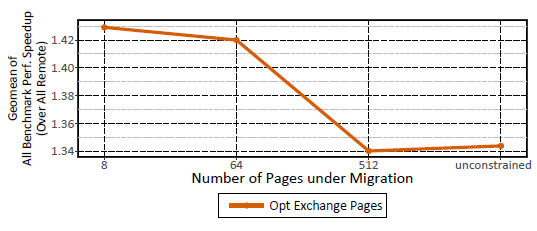


Figure 14. Geomean speedup (over All Remote, higher is better) given different numbers of pages under migration. Opt Exchange Pages use 4-thread parallel data copy. “Uncon-strained” means we do not limit the number of pages under migration; instead we just migrate all pages at once. All use 16GB Local Memory.

메모리 풋 프린트보다 훨씬 작으면 워크로드의 성능이 평균 5~10% 저하됩니다. 둘째, Exchange Base Pages는 성능 향상에 도움이 되지만 여전히 페이지 마이그레이션 처리량에 의해 제한됩니다. 그러나 기본페이지 마이그레이션보다 약간 낫습니다. 셋째, THP Migration은 지속적으로 성능을 향상시킵니다. 따라서 성능 저하없이 Disaggregated Memory를 사용하는 것이 가능합니다. 네번째, Exchange Page는 성능이 크게 향상되어 Disaggregated Memory 시스템의 잠재력을 최대한 활용하고 Linux의 기본 페이지 마이그레이션 성능에 비해 평균 40%를 능가합니다.

4.5 Sensitivity to Tunable Parameters

우리의 페이지 관리 시스템은 다양한 사용자 조정 가능 매개 변수를 제공합니다. 여기 , 우리는 얼마나 민감한 평가 성능은 이러한 매개 변수 중 두가지 입니다. 병렬 복사 스레드 수와 동시에 마이그레이션 된 페이지 수 입니다.

4.5.1 Number of threads for parallel page migration

우리의 시스템은 데이터를 복사하는데 사용할 스레드 수를 변경할 수 있으므로 조정 가능 매개 변수의 중요성을 이해하는 것이 중요합니다. 더 높은 스레드 수를 사용하면 Copy Throughput 이 향상되지만 응용 프로그램 자체에서 컴퓨팅 리소스를 훔치기 때문에 이러한 요소 간의 균형을 유지하는 방법을 이해하는 것이 중요합니다.

병렬 페이지 마이그레이션에서 사용된 스레드 수를 1~16까지 조정할 수 있습니다. 4개의 복사 스레드를 사용하는 것이 가장 성능이 뛰어난 Configure이지만 다른 스레드 수를 사용할 때 성능은 최대 4%에 불과하여 제안된 시스템이 이 매개 변수에 지나치게 민감하지 않습니다. 따라서 합리적인 지점(스레드 수)를 선택하는 것(예 : End-to-End 결과에서 사용한 4개의 스레드 구성 )은 합리적인 결정입니다.

4.5.2 Number of pages being migrated concurrently

복사 스레드의 조정과 유사하게, 동시 마이그레이션하는 페이지 수에 대한 시스템의 민감도를 테스트했습니다. 마이그레이션 할 페이지 수가 증가하면 Throughput도 증가합니다. 그러나 특정 시점을 넘어서 더 많은 수의 페이지를 마이그레이션하면 응용 프로그램이 Stall 되고 또한 성능의 저하로 결과가 나타납니다. 이러한 페이지는 마이그레이션이 진행중이며 사용자 프로세스에서 사용할 수 없기 때문입니다.

Figure 14는 동시 마이그레이션 페이지 수를 변경하여 나타낸 결과를 보여줍니다. 8페이지(페이지 크기에 따라 각각 32KB 또는 16MB의 데이터)가 가장 좋은 결과를 나타내기 때문에 시스템에서 동시에 마이그레이션될 페이지 수를 제한해야 합니다. 64페이지 미만의 값을 사용할 경우 성능 차이가 거의 없지만 수를 제한하지 않으면 성능이 거의 10%감소합니다.

4.6 Generality Across Architectures

Multi Level Memory System과 최적화의 플랫폼 독립성을 탐색하기 위해 다른 하드웨어 플랫폼으로 포팅하고 Section 4.2 에서 사용한 마이크로 벤치마크를 비교합니다. 멀티 레벨 메모리 페이지 Tracking 및 Policy 구현은 Linux의 현재 아키텍처 독립적인 활성 및 비활성 페이지 목록 구현을 기반으로 하기 때문에 설계에 독립적인 아키텍처입니다. 따라서 마이그레이션 최적화를 위한 플랫폼 민감도에 중점을 둡니다. 사용 가능한 메모리 대역폭과 CPU성능에 따라 각 플랫폼에 가장 적합한 구성이 달라 지므로 최적의 조정가능한 매개 변수를 사용하여 Table 2 에 결과를 표시합니다.

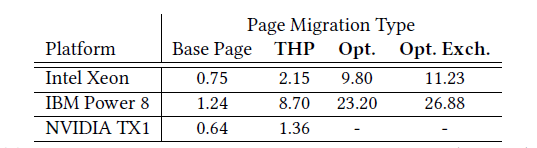
인텔 Xeon 프로세서를 사용하는 모든 플랫폼은 최적화의 이점을 모두 누릴 수 있다. Native THP Migration을 사용하여 마이그레이션 대역폭이 2.9배, Parallel and Concurrent 최적화를 사용하여 추가 4.6배 새로운 Exchage\_page() 를 사용하여 1.1배 더 향상됩니다. 최종적으로 총 15배 향상되었습니다. Power 프로세서를 사용하는 경우 동일한 최적화로 각각 7배 2.7배 1.2 Throughput을 달성합니다. 결과적으로 총 21.7배속 처리량이 향상됩니다. NVIDIA TX1(ARM64) 프로세서의 경우 기본 페이지 마이그레이션과 비교하여 THP 마이그레이션의 2.1배속 처리량을 

Table 2. Maximum achieved huge page migration (in GB/s)

throughput based on architecture independent optimizations

(bolded) shown across three architectures. When on NVIDIA

TX1 (ARM64), due to platform constraints, we are only able

to run THP migration.

관찰할 수 있습니다. 그러나 이 하드웨어 플랫폼에서 NUMA지원이 없기 때문에 단일 소켓 내 에서 결과를 측정하면 결과가 왜곡 될 수 있으므로 Concurrent and Exchange THP Migration 은 포함하지 않고 확인했습니다.

Xeon 프로세서 플랫폼에서 사용된 x86\_64 데이터 복사 명령 (Integer VS Float)에 따라 복사 처리량에 차이가 있습니다. 그러나 테스트 결과 SSE 및 AVX와 같은 SIMD 부동 소수점 명령어는 페이지 마이그레이션과 같이 쉽게 식별되는 메모리 패턴을 위해 캐시 내에서 공격적인 Linear Prefetch 로 인해 x86-64에서 mov보다 더 높은 복사 처리량을 제공하지 않습니다.

Power 프로세서 시스템에서 단일 스레드 데이터 전송은 전송 크기에 관계없이 거의10GB/s의 복사 대역폭을 달성하지만 여러 스레드를 사용할 때 달성 가능한 최대 복사 대역폭의 50%미만 입니다. Power에서 발생하는 향상된 단일 스레드 처리량은 효율적인 16바이트 단위로 데이터를 이동하는 정수 벡터 이동 명령에서 비롯됩니다. 단일 스레드 복사 대역폭을 향상시킬 수 있는 CPU 명령어 및 아키텍처는 궁극적으로 Base Page및 THP Migration 에 모두 도움이 되지만 벡터 명령어 조차도 소프트웨어 전용 기술을 통해 작업이 달성하는 총 성능과 일치하는데 필요한 15배의 향상을 달성 할 수는 없습니다.

4.7 Discussion of Experimental Results

요약하자면. 우리가 사용하는 분리 된 메모리 시스템과 같은 이기종 메모리 시스템에서 처리량이 적은 페이지 마이그레이션 메커니즘(예 : Linux)으로 인해 워크로드가 빠른 메모리의 이점을 활용 할 수 없습니다. 실제로 기존 메커니즘은 종종 빠른 메모리없이 시스템에서 실행하는 것보다 런타임이 더 나빠질 수 있을 정도로 성능을 저하시킵니다.

Multitier Memory의 잠재력을 실현하려면 THP Migration, Mutli Thread Copy, Concurrent migration of multiple pages, Two-way Exchange 등 4가지 주요 최적화 기능을 사용해야합니다. 우리의 평가에 따르면 우리가 제안한 이기종 메모리 페이지 관리 시스템은 여러 벤치마크에서 상당한 속도를 제공 할 수 있으며 디자인 공간을 살펴보면 우리의 기술이 다양한 아키텍처와 메모리 시스템 구성에 적용할 수 있을 만큼 유연하고 일반적임을 알 수 있습니다.

또한 빠른 메모리와 측정된 성능 변화에 비해 느린 메모리의 대역폭과 액세스 대기 시간을 변경했습니다. 우리는 최적화에 의해 제공되는 성능 향상이 4.3 및 4.4 Section에서 볼 수 있는것과 유사한 경향을 가지고 있음을 발견했습니다. 이 관찰은 계층형 메모리 시스템 관리에서 페이지 마이그레이션 메커니즘을 개선하는 것의 중요성을 더욱 강조합니다.

References

1. Neha Agarwal, David Nellans, Mark Stephenson, Mike O’Connor, and Stephen W. Keckler. 2015. Page Placement Strategies for GPUs within Heterogeneous Memory Systems. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 607–618.
2. Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-transparent Page Management for Two-tiered Main Memory. In Pro-ceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’17). ACM, New York, NY, USA, 631–644. [https://doi.org/10.1145/](https://doi.org/10.1145/3037697.3037706) [3037697.3037706](https://doi.org/10.1145/3037697.3037706)
3. Nadav Amit. 2017. Optimizing the TLB Shootdown Algorithm with Page Access Tracking. In 2017 USENIX Annual Technical Con-ference (USENIX ATC 17). USENIX Association, Santa Clara, CA, 27–39. [https://www.usenix.org/conference/atc17/technical-sessions/](https://www.usenix.org/conference/atc17/technical-sessions/presentation/amit) [presentation/amit](https://www.usenix.org/conference/atc17/technical-sessions/presentation/amit)
4. Andrea Arcangeli. [n. d.]. RFC: Transparent Hugepage support. [https:](https://lwn.net/Articles/358904/) [//lwn.net/Articles/358904/](https://lwn.net/Articles/358904/). [Online; accessed 31-Jul-2018].
5. Amro Awad, Arkaprava Basu, Sergey Blagodurov, Yan Solihin, and Gabriel H. Loh. 2017. Avoiding TLB Shootdowns Through Self-Invalidating TLB Entries. In 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). 273–287. <https://doi.org/10.1109/PACT.2017.38>
6. Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. 2013. Efficient Virtual Memory for Big Memory Servers. In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA ’13). ACM, New York, NY, USA, 237–248. <https://doi.org/10.1145/2485922.2485943>
7. Santiago Bock, Bruce R. Childers, Rami Melhem, and Daniel Mossé. 2014. Concurrent Page Migration for Mobile Systems with OS-managed Hybrid Memory. In Proceedings of the 11th ACM Conference on Computing Frontiers (CF ’14). ACM, New York, NY, USA, Article 31, 10 pages. <https://doi.org/10.1145/2597917.2597924>
8. Rohit Chandra, Scott Devine, Ben Verghese, Anoop Gupta, and Mendel Rosenblum. 1994. Scheduling and Page Migration for Multiprocessor Compute Servers. In Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI). ACM, New York, NY, USA, 12–24. [https://doi.](https://doi.org/10.1145/195473.195485) [org/10.1145/195473.195485](https://doi.org/10.1145/195473.195485)
9. Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2017. BAT-MAN: Techniques for Maximizing System Bandwidth of Memory Sys-tems with stacked-DRAM. In Proceedings of the International Sympo-sium on Memory Systems (MEMSYS ’17). ACM, New York, NY, USA, 268–280. <https://doi.org/10.1145/3132402.3132404>
10. Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2014. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchi-tecture (MICRO-47). IEEE Computer Society, Washington, DC, USA, 1–12. <https://doi.org/10.1109/MICRO.2014.63>
11. Julita Corbalan, Xavier Martorell, and Jesus Labarta. 2003. Evaluation of the Memory Page Migration Influence in the System Performance: The Case of the SGI O2000. In Proceedings of the 17th Annual Interna-tional Conference on Supercomputing (ICS ’03). ACM, New York, NY, USA, 121–129. <https://doi.org/10.1145/782814.782833>
12. Jonathan Corbet. 2012. AutoNUMA: the other approach to NUMA scheduling. <http://lwn.net/Articles/488709/>. [Online; accessed 31-Jul-2018].
13. Guilherme Cox and Abhishek Bhattacharjee. 2017. Efficient Address Translation for Architectures with Multiple Page Sizes. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’17). ACM, New York, NY, USA, 435–448. <https://doi.org/10.1145/3037697.3037704>
14. Kathy Davies. 2016.What’s new in Hyper-V on Win-

dows Server 2016 Technical Preview. [https://technet.](https://technet.microsoft.com/en-us/windows-server-docs/compute/hyper-v/what-s-new-in-hyper-v-on-windows) [microsoft.com/en-us/windows-server-docs/compute/hyper-v/](https://technet.microsoft.com/en-us/windows-server-docs/compute/hyper-v/what-s-new-in-hyper-v-on-windows) [what-s-new-in-hyper-v-on-windows](https://technet.microsoft.com/en-us/windows-server-docs/compute/hyper-v/what-s-new-in-hyper-v-on-windows). [Online; accessed: 31-Jul-2018].

1. Peter J. Denning. 1967. The Working Set Model for Program Be-havior. In Proceedings of the First ACM Symposium on Operating System Principles (SOSP ’67). ACM, New York, NY, USA, 15.1–15.12. <https://doi.org/10.1145/800001.811670>
2. Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. 2010. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’10). IEEE Computer Society, Washington, DC, USA, 1–11. <https://doi.org/10.1109/SC.2010.50>
3. Y. Du, M. Zhou, B. R. Childers, D. MossÃľ, and R. Melhem. 2015. Sup-porting superpages in non-contiguous physical memory. In 2015 IEEE 21st International Symposium on High Performance Computer Architec-ture (HPCA). 223–234. <https://doi.org/10.1109/HPCA.2015.7056035>
4. Jayneel Gandhi, Arkaprava Basu, Mark D. Hill, and Michael M. Swift. 2014. BadgerTrap: A Tool to Instrument x86-64 TLB Misses. SIGARCH Comput. Archit. News 42, 2 (Sept. 2014), 20–23. [https://doi.org/10.1145/](https://doi.org/10.1145/2669594.2669599) [2669594.2669599](https://doi.org/10.1145/2669594.2669599)
5. Jayneel Gandhi, Vasileios Karakostas, Furkan Ayar, AdriÃąn Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift, and Osman S. ÃĲnsal. 2016. Range Translations for Fast Virtual Memory. IEEE Micro 36, 3 (May 2016), 118–126. [https://doi.org/10.](https://doi.org/10.1109/MM.2016.10) [1109/MM.2016.10](https://doi.org/10.1109/MM.2016.10)
6. Fabien Gaud, Baptiste Lepers, Jeremie Decouchant, Justin Funston, Alexandra Fedorova, and Vivien Quema. 2014. Large Pages May Be Harmful on NUMA Systems. In 2014 USENIX Annual Technical Con-ference (USENIX ATC 14). USENIX Association, Philadelphia, PA, 231–
   1. [https://www.usenix.org/conference/atc14/technical-sessions/](https://www.usenix.org/conference/atc14/technical-sessions/presentation/gaud) [presentation/gaud](https://www.usenix.org/conference/atc14/technical-sessions/presentation/gaud)
7. Mel Gorman. 2004. Understanding the Linux Virtual Memory Manager. Prentice Hall. <https://books.google.com/books?id=ce1QAAAAMAAJ>
8. Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with In-finiswap. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). USENIX Association, Boston, MA, 649–
   1. [https://www.usenix.org/conference/nsdi17/technical-sessions/](https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu) [presentation/gu](https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu)
9. Nagendra Gulur, Mahesh Mehendale, R. Manikantan, and R. Govindara-jan. 2014. Bi-Modal DRAM Cache: A Scalable and Effective Die-Stacked DRAM Cache. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47). IEEE Computer Society, Washington, DC, USA, 38–50. <https://doi.org/10.1109/MICRO.2014.36>
10. Vishal Gupta, Min Lee, and Karsten Schwan. 2015. HeteroVisor: Ex-ploiting Resource Heterogeneity to Enhance the Elasticity of Cloud Platforms. In Proceedings of the 11th ACM SIGPLAN/SIGOPS Interna-tional Conference on Virtual Execution Environments (VEE ’15). ACM, New York, NY, USA, 79–92. <https://doi.org/10.1145/2731186.2731191>
11. Swapnil Haria, Mark D. Hill, and Michael M. Swift. 2018. Devirtualizing Memory in Heterogeneous Systems. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’18). ACM, New York, NY, USA, 637–650. <https://doi.org/10.1145/3173162.3173194>
12. Intel. [n. d.]. Intel Memory Latency Checker. [https://software.intel.](https://software.intel.com/en-us/articles/intelr-memory-latency-checker) [com/en-us/articles/intelr-memory-latency-checker](https://software.intel.com/en-us/articles/intelr-memory-latency-checker). [Online; accessed 31-Jul-2018].
13. Intel. 2016. Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. [http://www.hotchips.org/wp-content/uploads/hc\_archives/](http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf) [hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.](http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf)

[25.710-Knights-Landing-Sodani-Intel.pdf](http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf). [Online; accessed 31-Jul-2018].

1. JEDEC. 2014. JESD79-4A: DDR4 SDRAM Standard. [https://www.](https://www.jedec.org/sites/default/files/docs/JESD79-4A.pdf) [jedec.org/sites/default/files/docs/JESD79-4A.pdf](https://www.jedec.org/sites/default/files/docs/JESD79-4A.pdf). [Online; accessed 31-Jul-2018].
2. JEDEC. 2015. High Bandwidth Memory(HBM) DRAM - JESD235A. <http://www.jedec.org/standards-documents/docs/jesd235a>. [Online; accessed 31-Jul-2018].
3. Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47). IEEE Computer Society, Washington, DC, USA, 25–37. <https://doi.org/10.1109/MICRO.2014.51>
4. Guido Juckeland, William Brantley, Sunita Chandrasekaran, Barbara Chapman, Shuai Che, Mathew Colgrove, Huiyu Feng, Alexander Grund, Robert Henschel, Wen-Mei W. Hwu, Huian Li, Matthias S. Müller, Wolfgang E. Nagel, Maxim Perminov, Pavel Shelepugin, Kevin Skadron, John Stratton, Alexey Titov, Ke Wang, Matthijs van Waveren, Brian Whitney, Sandra Wienke, Rengan Xu, and Kalyan Kumaran. 2015. SPEC ACCEL: A Standard Application Suite for Measuring Hardware Accelerator Performance. Springer International Publishing, Cham, 46–67. <https://doi.org/10.1007/978-3-319-17248-4_3>
5. Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter. In Proceedings of the 44th Annual International Sympo-sium on Computer Architecture (ISCA ’17). ACM, New York, NY, USA, 521–534. <https://doi.org/10.1145/3079856.3080245>
6. Vasileios Karakostas, Jayneel Gandhi, Furkan Ayar, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift, and Osman Ünsal. 2015. Redundant Memory Mappings for Fast Access to Large Memories. In Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA ’15). ACM, New York, NY, USA, 66–78. <https://doi.org/10.1145/2749469.2749471>
7. Mohan Kumar, Steffen Maass, Sanidhya Kashyap, Ján Veselý, Zi Yan, Taesoo Kim, Abhishek Bhattacharjee, and Tushar Krishna. 2018. LATR: Lazy Translation Coherence. In Proceedings of the Twenty-Third In-ternational Conference on Architectural Support for Programming Lan-guages and Operating Systems (ASPLOS ’18). ACM, New York, NY, USA, 651–664. <https://doi.org/10.1145/3173162.3173198>
8. Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. 2016. Coordinated and Efficient Huge Page Management with Ingens. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI’16). USENIX Association, Berkeley, CA, USA, 705–721. [http://dl.acm.org/citation.](http://dl.acm.org/citation.cfm?id=3026877.3026931) [cfm?id=3026877.3026931](http://dl.acm.org/citation.cfm?id=3026877.3026931)
9. Christoph Lameter. [n. d.]. Swap migration V3: Overview. [https:](https://lwn.net/Articles/156603/) [//lwn.net/Articles/156603/](https://lwn.net/Articles/156603/). [Online; accessed 31-Jul-2018].
10. Christoph Lameter. 2013. NUMA (Non-Uniform Memory Access): An Overview. Queue 11, 7, Article 40 (July 2013), 12 pages. [https:](https://doi.org/10.1145/2508834.2513149) [//doi.org/10.1145/2508834.2513149](https://doi.org/10.1145/2508834.2513149)
11. Lawerence Livermore National Laboratory. 2016. CORAL/Sierra. [https:](https://asc.llnl.gov/coral-info) [//asc.llnl.gov/coral-info](https://asc.llnl.gov/coral-info). [Online; accessed 31-Jul-2018].
12. Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting Phase Change Memory As a Scalable Dram Alternative. In Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA ’09). ACM, New York, NY, USA, 2–13. [https://doi.](https://doi.org/10.1145/1555754.1555758) [org/10.1145/1555754.1555758](https://doi.org/10.1145/1555754.1555758)
13. Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. 2015. Thread and memory placement on NUMA systems: asymmetry matters. In 2015 USENIX Annual Technical Conference (USENIX ATC 15). 277–289.
14. Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. In Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA ’09). ACM, New York, NY, USA, 267–278. [https://doi.org/10.](https://doi.org/10.1145/1555754.1555789) [1145/1555754.1555789](https://doi.org/10.1145/1555754.1555789)
15. Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2012. System-level Implications of Disaggregated Memory. In International Symposium on High-Performance Computer Architecture (HPCA). 1–12.
16. Felix Xiaozhu Lin and Xu Liu. 2016. Memif: Towards programming heterogeneous memory asynchronously. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 369–383.
17. Gabriel H. Loh and Mark D. Hill. 2011. Efficiently Enabling Con-ventional Block Sizes for Very Large Die-stacked DRAM Caches. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44). ACM, New York, NY, USA, 454–464. <https://doi.org/10.1145/2155620.2155673>
18. Jasmina Malicevic, Subramanya Dulloor, Narayanan Sundaram, Na-dathur Satish, Jeff Jackson, and Willy Zwaenepoel. 2015. Exploiting nvm in large-scale graph analytics. In Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. ACM, 2.
19. Sally A. McKee. 2004. Reflections on the Memory Wall. In Proceedings of the 1st Conference on Computing Frontiers (CF ’04). ACM, New York, NY, USA, 162–. <https://doi.org/10.1145/977091.977115>
20. Marshall Kirk McKusick and George V. Neville-Neil. 2004. The De-sign and Implementation of the FreeBSD Operating System. Pearson Education.
21. Mitesh R. Meswani, Sergey Blagodurov, David Roberts, J ohn Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. Heterogeneous Memory Architectures: A HW/SW Approach For Mixing Die-stacked And Off-package Memories. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). 126–136.
22. Micron 2015. Hybrid Memory Cube Specification 2.1. [https://www.](https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf) [nuvation.com/sites/default/files/Nuvation-Engineering-Images/](https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf) [Articles/FPGAs-and-HMC/HMC-30G-VSR\_HMCC\_Specification.](https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf) [pdf](https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf). [Online; accessed 31-Jul-2018].
23. Micron. 2016. 3D XPoint Technology. [https://www.micron.com/](https://www.micron.com/products/advanced-solutions/3d-xpoint-technology) [products/advanced-solutions/3d-xpoint-technology](https://www.micron.com/products/advanced-solutions/3d-xpoint-technology). [Online; ac-cessed 31-Jul-2018].
24. Jeffery Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi. 2009. Operating System Support for NVM+DRAM Hybrid Main Mem-ory. In Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS’09). USENIX Association, Berkeley, CA, USA, 14–18. <http://dl.acm.org/citation.cfm?id=1855568.1855582>
25. Richard C. Murphy, Kyle B. Wheeler, Brian W. Barrett, and James A. Ang. 2010. Introducing the Graph 500. In Cray User’s Group.
26. Linux Newbies. 2017. Linux 4.14 Release Note. [https://kernelnewbies.](https://kernelnewbies.org/Linux_4.14#Memory_management) [org/Linux\_4.14#Memory\_management](https://kernelnewbies.org/Linux_4.14#Memory_management)
27. Dimitrios S. Nikolopoulos, Theodore S. Papatheodorou, Constantine D. Polychronopoulos, Jesús Labarta, and Eduard Ayguadé. 2000. A Case for User-level Dynamic Page Migration. In Proceedings of the 14th International Conference on Supercomputing (ICS ’00). ACM, New York, NY, USA, 119–130. <https://doi.org/10.1145/335231.335243>
28. Dimitrios S. Nikolopoulos, Theodore S. Papatheodorou, Constantine D. Polychronopoulos, Jesús Labarta, and Eduard Ayguadé. 2000. User-Level Dynamic Page Migration for Multiprogrammed Shared-Memory Multiprocessors. In Proceedings of the Proceedings of the 2000 Inter-national Conference on Parallel Processing (ICPP ’00). IEEE Computer Society, Washington, DC, USA, 95–. [http://dl.acm.org/citation.cfm?](http://dl.acm.org/citation.cfm?id=850941.852887) [id=850941.852887](http://dl.acm.org/citation.cfm?id=850941.852887)
29. NVIDIA Corporation. 2013. Unified Memory in CUDA 6. [http:](http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/) [//devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/](http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/). [On-line; accessed 31-Jul-2018].
30. NVIDIA Corporation. 2014. NVLink, Pascal and Stacked Memory: Feed-ing the Appetite for Big Data. [http://devblogs.nvidia.com/parallelforall/](http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/) [nvlink-pascal-stacked-memory-feeding-appetite-big-data/](http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/). [Online; accessed 14-Aug-2016].

1. Oak Ridge National Laboratory. 2018. Summit. [https://www.olcf.ornl.](https://www.olcf.ornl.gov/summit/) [gov/summit/](https://www.olcf.ornl.gov/summit/). [Online; accessed 31-Jul-2018].
2. Mark Oskin and Gabriel H. Loh. 2015. A Software-Managed Ap-proach to Die-Stacked DRAM. In Proceedings of the 2015 Interna-tional Conference on Parallel Architecture and Compilation (PACT) (PACT ’15). IEEE Computer Society, Washington, DC, USA, 188–200. <https://doi.org/10.1109/PACT.2015.30>
3. Ashish Panwar, Aravinda Prasad, and K. Gopinath. 2018. Making Huge Pages Actually Useful. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’18). ACM, New York, NY, USA, 679–692. <https://doi.org/10.1145/3173162.3173203>
4. Misel-Myrto Papadopoulou, Xin Tong, AndrÃľ Seznec, and Andreas Moshovos. 2015. Prediction-based superpage-friendly TLB designs. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). 210–222. [https://doi.org/10.1109/HPCA.2015.](https://doi.org/10.1109/HPCA.2015.7056034) [7056034](https://doi.org/10.1109/HPCA.2015.7056034)
5. Mayank Parasar, Abhishek Bhattacharjee, and Tushar Krishna. 2018. SEESAW: Using Superpages to Improve VIPT Caches. In Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA ’18). IEEE Press, Piscataway, NJ, USA, 193–206. [https://doi.org/](https://doi.org/10.1109/ISCA.2018.00026) [10.1109/ISCA.2018.00026](https://doi.org/10.1109/ISCA.2018.00026)
6. J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In 2011 IEEE Hot Chips 23 Symposium (HCS). 1–24. [https://doi.org/10.1109/](https://doi.org/10.1109/HOTCHIPS.2011.7477494) [HOTCHIPS.2011.7477494](https://doi.org/10.1109/HOTCHIPS.2011.7477494)
7. Binh Pham, Abhishek Bhattacharjee, Yasuko Eckert, and Gabriel H. Loh. 2014. Increasing TLB reach by exploiting clustering in page translations. In 2014 IEEE 20th International Symposium on High Per-formance Computer Architecture (HPCA). 558–567. [https://doi.org/10.](https://doi.org/10.1109/HPCA.2014.6835964) [1109/HPCA.2014.6835964](https://doi.org/10.1109/HPCA.2014.6835964)
8. Binh Pham, Ján Veselý, Gabriel H. Loh, and Abhishek Bhattacharjee. 2015. Large Pages and Lightweight Memory Management in Virtual-ized Environments: Can You Have It Both Ways?. In Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48). ACM, New York, NY, USA, 1–12. <https://doi.org/10.1145/2830772.2830773>
9. Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. 2014. Architec-tural Support for Address Translation on GPUs: Designing Memory Management Units for CPU/GPUs with Unified Address Spaces. In Pro-ceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’14). ACM, New York, NY, USA, 743–758. <https://doi.org/10.1145/2541940.2541942>
10. Jason Power, Mark D. Hill, and David A. Wood. 2014. Supporting x86-64 address translation for 100s of GPU lanes. In 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). 568–578. <https://doi.org/10.1109/HPCA.2014.6835965>
11. Moinuddin K. Qureshi and Gabe H. Loh. 2012. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In Proceedings of the 2012 45th Annual International Symposium on Microarchitecture. 12. <https://doi.org/10.1109/MICRO.2012.30>
12. Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In Proceedings of the 36th Annual Inter-national Symposium on Computer Architecture (ISCA ’09). ACM, New York, NY, USA, 24–33. <https://doi.org/10.1145/1555754.1555760>
13. Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page Placement in Hybrid Memory Systems. In Proceedings of the Interna-tional Conference on Supercomputing (ICS ’11). ACM, New York, NY,

USA, 85–95. <https://doi.org/10.1145/1995896.1995911>

1. Bogdan F. Romanescu, Alvin R. Lebeck, Daniel J. Sorin, and Anne Bracy. 2010. UNified Instruction/Translation/Data (UNITD) coherence: One protocol to rule them all. In HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture. 1–12. [https:](https://doi.org/10.1109/HPCA.2010.5416643) [//doi.org/10.1109/HPCA.2010.5416643](https://doi.org/10.1109/HPCA.2010.5416643)
2. Jee Ho Ryoo, Lizy K. John, and Arkaprava Basu. 2018. A Case for Granularity Aware Page Migration. In Proceedings of the International Conference on Supercomputing (ICS ’18). ACM, New York, NY, USA. <https://doi.org/10.1145/3205289.3208064>
3. Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, , Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, and Michael A Kozuch. 2016. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In2016 IEEE International Symposium on High Performance Computer Archi-tecture (HPCA). IEEE, 481–493.
4. AndrÃľ Seznec. 2004. Concurrent Support of Multiple Page Sizes on a Skewed Associative TLB. IEEE Trans. Comput. 53, 7 (July 2004), 924–927. <https://doi.org/10.1109/TC.2004.21>
5. Jaewoong Sim, Alaa R Alameldeen, Zeshan Chishti, Chris Wilker-son, and Hyesoon Kim. 2014. Transparent hardware management of stacked dram as part of memory. In 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE, 13–24.
6. Jaewoong Sim, Gabriel H. Loh, Hyesoon Kim, Mike O’Connor, and Mithuna Thottethodi. 2012. A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45). IEEE Computer Society, Washington, DC, USA, 247–257. <https://doi.org/10.1109/MICRO.2012.31>
7. Mustafa M. Tikir and Jeffrey K. Hollingsworth. 2008. Hardware Mon-itors for Dynamic Page Migration. J. Parallel Distrib. Comput. 68, 9 (Sept. 2008), 1186–1200. <https://doi.org/10.1016/j.jpdc.2008.05.006>
8. Linus Torvalds. 2014. Performance profiling on core kernel code.[https:](https://plus.google.com/+LinusTorvalds/posts/YDKRFDwHwr6) [//plus.google.com/+LinusTorvalds/posts/YDKRFDwHwr6](https://plus.google.com/+LinusTorvalds/posts/YDKRFDwHwr6). [Online; accessed 31-Jul-2018].
9. UEFI.org. 2017. Advanced Configuration and Power Interface Specifi-cation, Version 6.2. [http://www.uefi.org/sites/default/files/resources/](http://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf) [ACPI\_6\_2.pdf](http://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf). [Online; accessed 31-Jul-2018].
10. Carlos Villavieja, Vasileios Karakostas, Lluis Vilanova, Yoav Etsion, Alex Ramirez, Avi Mendelson, Nacho Navarro, Adrian Cristal, and Osman S. Unsal. 2011. DiDi: Mitigating the Performance Impact of TLB Shootdowns Using a Shared TLB Directory. In 2011 International Conference on Parallel Architectures and Compilation Techniques. 340– 349. <https://doi.org/10.1109/PACT.2011.65>
11. Hao Wang, Jie Zhang, Sharmila Shridhar, Gieseo Park, Myoungsoo Jung, and Nam Sung Kim. 2016. DUANG: Fast and lightweight page migration in asymmetric memory systems. In 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 481–493.
12. Zi Yan, Ján Veselý, Guilherme Cox, and Abhishek Bhattacharjee. 2017. Hardware Translation Coherence for Virtualized Systems. In Proceedings of the 44th Annual International Symposium on Com-puter Architecture (ISCA ’17). ACM, New York, NY, USA, 430–443. <https://doi.org/10.1145/3079856.3080211>
13. Ross Zwisler. 2017. Surface Heterogeneous Memory Performance Information. <https://lwn.net/Articles/727348/>. [Online; accessed 31-Jul-2018].