BATMAN: Techniques for Maximizing System Bandwidth of Memory Systems with Stacked-DRAM

Chiachen Chou
School of ECE
Georgia Institute of Technology
Atlanta, GA
cchou34@gatech.edu

Aamer Jaleel
NVIDIA Research
NVIDIA
Santa Clara, CA
ajaleel@nvidia.com

Moinuddin Qureshi
School of ECE
Georgia Institute of Technology
Atlanta, GA
moin@gatech.edu

ABSTRACT

Tiered-memory systems consist of high-bandwidth 3D-DRAM and high-capacity commodity-DRAM. Conventional designs attempt to improve system performance by maximizing the number of memory accesses serviced by 3D-DRAM. However, when the commodity-DRAM bandwidth is a significant fraction of overall system bandwidth, the techniques inefficiently utilize the total bandwidth offered by the tiered-memory system and yields sub-optimal performance. In such situations, the performance can be improved by distributing memory accesses that are proportional to the bandwidth of each memory. Ideally, we want a simple and effective runtime mechanism that achieves the desired access distribution without requiring significant hardware or software support.

This paper proposes *Bandwidth-Aware Tiered-Memory Management (BATMAN)*, a runtime mechanism that manages the distribution of memory accesses in a tiered-memory system by explicitly controlling data movement. BATMAN monitors the number of accesses to both memories, and when the number of 3D-DRAM accesses exceeds the desired threshold, BATMAN disallows data movement from the commodity-DRAM to 3D-DRAM and proactively moves data from 3D-DRAM to commodity-DRAM. We demonstrate BATMAN on systems that architect the 3D-DRAM as either a hardware-managed cache (cache mode) or a part of the OS-visible memory space (flat mode). Our evaluations on a system with 4GB 3D-DRAM and 32GB commodity-DRAM show that BATMAN improves performance by an average of 11% and 10% and energy-delay product by 13% and 11% for systems in the cache and flat modes, respectively. BATMAN incurs only an eight-byte hardware overhead and requires negligible software modification.

CCS CONCEPTS

•Computer systems organization → Architectures; •Hardware → Semiconductor memory; *Emerging technologies; Emerging interfaces;* **•Software and its engineering → Memory management;**

An earlier version of this paper was published as a technical report at Georgia Tech in March 2015 [9].

MEMSYS 2017, October 2–5, 2017, Alexandria, VA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5335-9/17/10…$15.00
https://doi.org/10.1145/3132402.3132404

1 INTRODUCTION

3D-DRAM technology, such as hybrid memory cube (HMC) and high-bandwidth memory (HBM) [20; 25; 38], cater to the growing demands for high memory bandwidth. Compared to commodity DRAM (e.g., DDR3 and DDR4 [24; 36]), 3D-DRAM provides 4x-8x high bandwidth at similar DRAM access latency [7; 44]. As 3D-DRAM, compared to commodity DRAM, has limited capacity, both 3D-DRAM and commodity DRAM are likely to co-exist in future memory systems. We refer to such a system as a *hybrid-memory system* or a *tiered-memory system*. Recently announced products such as Intel's Knights Landing (KNL) [43–45] and NVIDIA's Pascal [39] already incorporate a tiered-memory system design.

In tiered-memory systems, 3D-DRAM is referred to as the *near memory (NM)* and commodity DRAM as the *far memory (FM)*. Figure 1(a) shows a typical tiered-memory system available in commercial products today: a 4x-bandwidth NM and a 1x-bandwidth FM. The NM in a tiered-memory system can be architected in two ways; for example, Intel's KNL configures the NM in either a *cache mode* or a *flat mode*. In the cache mode, the NM is configured as a hardware-managed cache (*DRAM cache*) that sits between on-chip caches and the main memory (FM) [11; 17; 26; 27; 31; 41; 42; 45]. In the flat mode, the NM is part of the main memory and is visible to the operating system (OS) [2; 10; 13; 14; 22; 32; 35].

Regardless of how the NM is architected, the conventional wisdom maximizes the fraction of memory requests satisfied by the NM (i.e., the access rate of the NM) [17; 26; 35]. Doing so inefficiently utilizes total available bandwidth in the system. For example, Figure 1(b) presents an application whose frequently accessed working set fits in the NM and shows a system that employs the conventional approach: On an access to data in the FM, the conventional approach always moves the data to the NM, shown in Figure 1(b), so later accesses to the data are serviced by the NM. In a steady state, the entire working set is always serviced by the NM. As a result, the utilized system bandwidth is 4x (only NM), as shown in Figure 1(c). However, such a scheme under-utilizes the FM bandwidth. Given that the FM bandwidth accounts for 20% of overall system bandwidth, the conventional approach does not optimize for total system bandwidth, which leads to sub-optimal performance.

As the NM simply offers higher bandwidth, not lower latency, the performance of tiered-memory systems is determined by the utilization of system bandwidth. We observe that both system bandwidth and performance are maximized when memory accesses are distributed proportional to the bandwidth of each memory. For the example in Figure 1(a), the NM should service 80% of memory

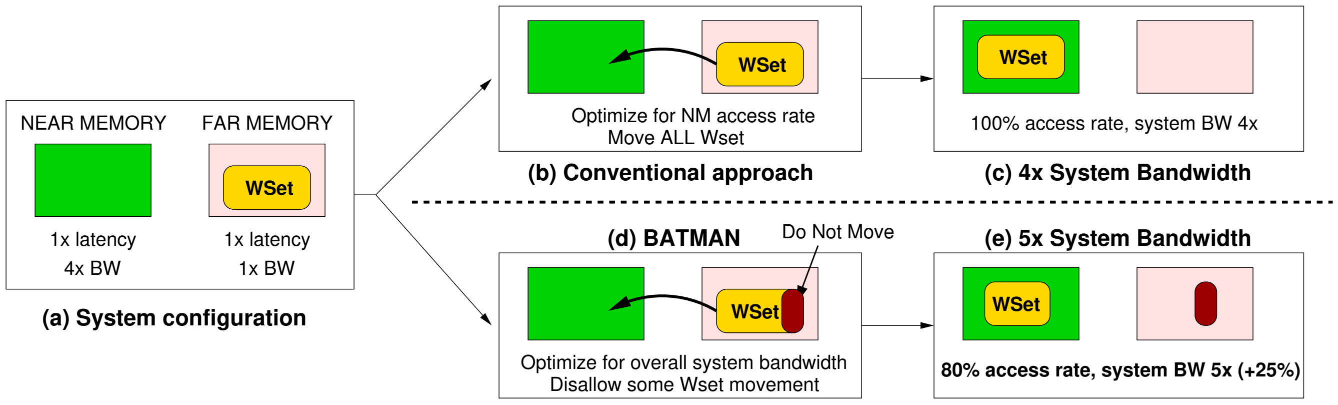
NEAR MEMORY
FAR MEMORY
WSet
1x latency
4x BW
1x latency
1x BW
(a) System configuration
WSet
Optimize for NM access rate
Move ALL Wset
(b) Conventional approach
WSet
100% access rate, system BW 4x
(c) 4x System Bandwidth
(d) BATMAN
Do Not Move
WSet
Optimize for overall system bandwidth
Disallow some Wset movement
(e) 5x System Bandwidth
WSet
80% access rate, system BW 5x (+25%)

Figure 1: Optimizing for the access rate versus for system bandwidth. (a) A system where the NM has 4x the bandwidth as that of the FM. *Wset* denotes the working set of the application. (b) and (c) Conventional approach that optimizes for the NM access rate obtains up to 4x system bandwidth (only NM). (d) and (e) Explicitly controlling the NM access rate achieves 5x system bandwidth (NM+FM), 25% higher than the conventional approach.

accesses and the FM 20% of memory accesses. The observed relationship between access distribution and performance is consistent with a recent study [1] that uses software modification to enforce the access distribution. Specifically, the prior study proposes static page placement strategy, which requires compiler support for profiling and programmer intervention that annotates data structures of programs. However, the proposed static scheme is sensitive to the input set of applications and system parameters, so it cannot distribute memory accesses at runtime. This paper seeks a simple and effective mechanism that achieves the desired distribution at runtime without requiring any software support.

Tiered-memory systems provide functionality to move data between the NM and FM (e.g. OS-supported page migration in the flat mode and cache line transfers in the cache mode). When data move to the NM, the number of memory accesses serviced by the NM increases, and the same principle also applies to the FM. Therefore, if we can control the data movement in the tiered-memory system, we can regulate the memory access distribution. For instance, in Figure 1(d), when the NM already has 80% of memory accesses, if we disallow data movement from the FM to the NM and keep data that account for 20% of memory accesses in the FM, we would achieve the desired split of memory accesses that maximizes the system bandwidth utilization. In such case, the overall system bandwidth has the maximum of 5x (NM + FM), which is 25% higher than that of the conventional approach, shown in Figure 1(e).

We leverage our key insight on controlling data movement and propose *Bandwidth-Aware Tiered-Memory Management (BATMAN)*, which is a runtime mechanism that monitors memory access distribution and explicitly controls the data movement between the NM and the FM. We define the desired access rate of the NM as the *target access rate (TAR)*. TAR is the fraction of memory accesses serviced by the NM when memory accesses to both memories are proportional to the respective bandwidth. When the access rate of the NM exceeds the TAR, BATMAN disallows data movement from the FM to the NM and also proactively migrates data from the NM to the FM, which decreases the NM access rate. When the NM access rate falls below the TAR, BATMAN allows the data movement from the FM to the NM, the same as conventional approaches, which increases the NM access rate.

To the best of our knowledge, BATMAN is the first study that maximizes system bandwidth of tiered-memory systems at runtime. We demonstrate BATMAN in the context of both the cache and flat modes. In the cache mode, BATMAN monitors the NM access rate and distributes memory accesses by partially disabling the cache. When the access rate of the NM exceeds the TAR, BATMAN dynamically turns off a fraction of the cache sets to ensure that memory accesses that could have been satisfied by the disabled cache sets are serviced by the FM. Although BATMAN lowers the cache hit rate by cache disabling, BATMAN improves overall system bandwidth. BATMAN requires no software support and incurs an overhead of only two 16-bit counters and a 32-bit register. In the flat mode, we assume a system that relies on the OS to perform page migration; BATMAN achieves the TAR by monitoring the NM access rate at runtime and dynamically controls the direction of page migration based on the NM access rate. When the access rate of the NM exceeds the TAR, BATMAN disallows page migration from the FM to the NM. BATMAN leverages the existing OS support of page migration and incurs an overhead of two 16-bit counters.

We evaluate BATMAN on a 16-core system with a 4GB NM (4x bandwidth) and a 32GB FM (1x bandwidth), which is similar to the configuration of a *sub-NUMA cluster* (SNC) node of Intel's Knights Landing product. In both the cache and flat modes, BATMAN is effective at maintaining the desired distribution of memory accesses at runtime, while BATMAN incurs only an eight-byte hardware overhead and requires negligible software modification. On average, BATMAN improves performance for a system that uses the NM in cache mode by 11% and for a system that uses the NM in flat mode by 10%. Also, BATMAN improves energy-delay product by 13% and 11% for systems in the cache and flat modes, respectively.

2 BACKGROUND AND MOTIVATION

This section introduces two uses for the 3D-DRAM in a tiered-memory system, analyzes the effectiveness of conventional schemes that maximize the access rate of the 3D-DRAM, and provides a quantitative motivation for optimizing for the overall system bandwidth.

2.1 3D-DRAM in a Tiered-Memory System

The recently introduced 3D-DRAM, such as HBM and HMC [20; 25; 38], is an emerging DRAM technology that, compared to the commodity DDR-based DRAM, offers high bandwidth but has limited capacity and similar latency. Consequently, emerging systems combine the high-bandwidth 3D-DRAM with the low-cost commodity DDR-based DRAM to form a tiered-memory system. One key characteristic of the tiered-memory system is the latency of 3D-DRAM and DDR-based DRAM. Although 3D-DRAM provides higher bandwidth than DDR-based DRAM, recent studies indicate that because the 3D-DRAM and the DDR-based DRAM use the same DRAM technology, the latency of 3D-DRAM is similar to that of DDR-based DRAM [7; 44]. As illustrated in Figure 2, 3D-DRAM technology, such as HMC and HBM, exhibit similar DRAM access latency as the DDR-based DRAM, such as DDR3 and DDR4.

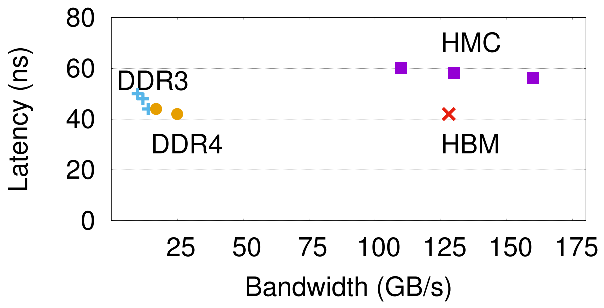

Figure 2: A comparison of DRAM technologies: latency (ns) versus bandwidth (GB/s).

In a tiered-memory system, 3D-DRAM can be architected in two ways: a hardware-managed cache or part of the OS-visible memory space [43–45]. As mentioned, we refer to the 3D-DRAM as the *near memory* (NM) and the DDR-based DRAM as the *far memory* (FM). The first use of the NM is a hardware-managed cache, or a *DRAM cache*, an intermediate level between the last-level cache and the main memory (FM). Although many studies have proposed various organizations for DRAM cache [17; 26; 27; 31; 42], recent academic research and commercial products have converged to a direct-mapped, 64B-line-size, and tags-with-data DRAM cache [11; 41; 45]. This use of the NM is referred to as the *cache mode*. Alternately, the NM can also be architected as a part of the OS-visible memory space, referred to as the *flat mode*. In the flat mode, the NM and the FM are exposed as two nodes to the operating system (OS), which is responsible for managing the data placement in and movement between the NM and the FM [43]. For instance, the OS provides a special malloc() function, such as `hbw_malloc()`, to exploit the high-bandwidth NM.

2.2 Conventional Wisdom: Optimize for the NM Access Rate

In both the cache and flat modes, many studies tend to maximize the access rate of the NM, which is defined as the fraction of memory accesses that the NM receives, shown in Equation 1. The underlyingassumption of prior studies is that the NM has a latency advantage over the FM. Therefore, the conventional wisdom is to retain as much of the frequently accessed working set in the NM as possible, and the effectiveness of the scheme is determined by the fraction of memory accesses that are serviced by the NM, that is, the access rate of NM. For example, in the cache mode, researchers propose mechanisms and organizations that maximize the cache hit rate so that almost all memory accesses are serviced by the NM [17; 26; 31]. Also, in the flat mode, the goal of many studies is to place or move all the *hot* (i.e., frequently accessed) data in the NM [21; 35] and use only the NM to service all memory requests.

$$\text{The NM Access Rate} = \frac{\text{Total NM Accesses}}{\text{Total Memory Accesses}} \tag{1}$$

As 3D-DRAM simply provides higher bandwidth, not lower latency, placing all the frequently accessed data in the NM does not necessarily yield optimal performance. Unlike the conventional designs, which evaluates its effectiveness according to the NM access rate, the key metric for a tiered-memory system is the number of memory requests satisfied within a time period, or, simply put, the overall system bandwidth. Higher overall system bandwidth leads to better system performance. In a conventional approach that optimizes for the access rate of NM, the overall system bandwidth is under-utilized and capped by the bandwidth of the NM. Ideally, to maximize overall system bandwidth and performance, we would like to use all of the bandwidth available from both NM and FM.

2.3 Optimize for the Overall System Bandwidth

The tiered-memory system utilizes the bandwidth of both the NM and the FM by distributing memory accesses to both memories. We corroborate this hypothesis experimentally: As an example, we study a set of memory intensive STREAM benchmarks [34] on a memory system whose configuration is similar to that in Figure 1 (4x bandwidth NM and 1x bandwidth FM, see the experimental methodology in Section 3). As the working set of the STREAM benchmarks fits in the NM, the baseline configuration services all memory accesses entirely from the NM (a 100% NM access rate). We conduct a sensitivity study that distributes memory accesses to both memories by explicitly allocating part of the working set in the FM. Figure 3 shows the speedup compared to the baseline as the the fraction of memory requests serviced by the NM increases from 10% to 100%. For all systems that vary the bandwidth ratio from 2X, 4X, to 8X, the peak performance occurs much earlier than 100%, validating our hypothesis that the system optimizing for overall system bandwidth achieves higher performance.

The maximum performance improvement depends on the overall bandwidth improvement. When the ratio of the bandwidth of the

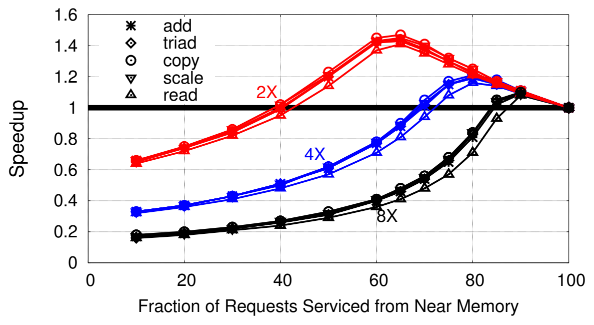

Figure 3: Speedup versus the fraction of accesses serviced from the near memory. Note that peak performance occurs far below 100% (i.e., the baseline system).

NM to that of the FM is 4x, we get a peak performance of 1.2x, close to the ideal speedup of 1.25x, which we expect by improving the system bandwidth by 25% (system bandwidth increases from 4x to 5x, or 25% improvement). A similar observation applies to systems with different bandwidth ratios; systems with 2x and 8X bandwidth ratio have the peak performance of 1.45x and 1.11x, respectively, while the bandwidth improves by 50% and 12.5%, respectively. Therefore, for a tiered-memory system in which the FM accounts for a significant fraction of the overall bandwidth, distributing memory accesses to both memories has great potential to improve performance.

2.4 Goal: Optimum Split at Runtime

Peak performance occurs when memory accesses are distributed in proportion to the bandwidth of each memory. When the NM has 4x as high bandwidth, peak performance occurs when the access rate of the NM is approximately 80%: The NM services four-fifths of the accesses and the FM the remaining one-fifth of the accesses. The same principle applies to the case of 2X and 8X the bandwidth ratio. This observation that maximum system bandwidth and peak performance occur when memory accesses are distributed in proportion to the respective bandwidth is consistent with recent studies [1; 12], one of which is a study by Agarwal et al. on tiered-memory systems for GPU [1]. The authors propose a static page placement strategy that relies on programmers knowledge of the data structures in workloads. However, this study has several drawbacks: First, their scheme requires software modification. Second, the proposed static scheme cannot adjust the memory access distribution at runtime. As the prior work has limited applicability and requires programmer and software intervention, we seek a mechanism that achieves the desired memory access distribution at runtime without any software support.

The key insight that achieves the desired access distribution is the explicit control of data movement in tiered-memory systems. When data moves from the FM to the NM, subsequent memory access to the data will be serviced by the NM, which increases the NM access rate. Examples of such data movement include cache line install in the cache mode and page migration in the flat mode. Therefore, controlling data movement can regulate the NM access rate. To this end, we propose *Bandwidth-Aware Tiered-Memory Management (BATMAN)*, a runtime mechanism that manages the memory accesses distribution in proportion to the bandwidth ratio of the NM and the FM. The desired NM access rate is referred to as the *target access rate* (TAR). BATMAN monitors the access rate of the NM at runtime and controls the data movement to meet the TAR: When the NM access rate exceeds the TAR, BATMAN disallows data movement from the FM to the NM and proactively moves data from the NM to the FM, which lowers the NM access rate. Also, when the NM access rate falls below the TAR, BATMAN does not intervene in data movement that increases the NM access rate. We develop BATMAN in the context of two NM use cases: first, BATMAN in the cache mode (Section 4) and second, BATMAN in the flat mode (Section 5). Before presenting our solutions, we describe our experimental methodology.

3 EXPERIMENTAL METHODOLOGY

3.1 System Configuration

We model a 16-core system similar to one Intel's Knights Landing sub-NUMA cluster (SNC) node [43] by a detailed event-driven x86 simulator. Table 1 shows the core parameters, the cache hierarchy organization, and latency numbers, all of which are similar to the configuration of recent Intel Xeon processors [23]. Each core, running at 3.2GHz, is a four-wide issue out-of-order processor with a 128-entry ROB. The on-chip cache subsystem contains a three-level cache hierarchy with private L1 and L2 caches and an L3 cache shared by the cores. All cache hierarchies use a 64B cache line.

Table 1: Baseline System Configuration

Processors	
Number of Cores	16
Frequency	3.2GHz
Core width	4-wide out-of-order
Prefetcher	Stream prefetcher
Last Level Cache	
Shared L3 cache	16MB, 16-way, 27 cycles
Near Memory (3D-DRAM)	
Capacity	4GB
Bus frequency	800MHz (DDR 1.6GHz)
Channels	8
Bus width	64 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles
Far Memory (DDR-based DRAM)	
Capacity	32GB
Bus frequency	800MHz (DDR 1.6GHz)
Channels	2
Bus width	64 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles

The memory system consists of a 4GB near memory (NM) using HBM2 technology [25] and a 32GB far memory (FM) using DDR3 technology [36]. As recent specification reveals that 3D-DRAM uses the same DRAM technology as DDR-based DRAM and thus has identical no-load latency, we assume the same timing parameters in both DRAM technologies [24; 25]. However, the bandwidth of the NM is higher than that of the FM. In our baseline system, the NM (4x channel) has 4x as high bandwidth as the FM. Note that this configuration is similar to one sub-NUMA cluster (SNC) node in Intel KNL. We present a sensitivity study of the bandwidth ratio in Section 6. Our DRAM simulator is similar to USIMM [8] and contains read and write queues for each memory channel. The DRAM controller prioritizes reads over writes, and writes are issued in batches. The default memory address mapping policy is the minimalist-open page policy [28], which exploits memory channel parallelism and also retains the benefits of DRAM page hits. The policy places a group of four consecutive cache lines in the same DRAM page and interleaves groups of four lines among memory channels.

Cache mode In the first use case of the NM, we configure the NM as a hardware-managed cache that is an intermediate level between on-chip caches and the main memory. We refer to the cache as *DRAM cache*, which is a direct-mapped, 64B-cache-line-size, and tags-with-data cache. The configuration of the DRAM cache is similar to that in a recent academic study [41] and that of commercial products [45]. We equip the DRAM cache with a cache hit-miss predictor. The NM access rate in the cache mode includes all DRAM cache operations, such as miss- and writeback-related operations [11]. To optimize for the NM access rate, the baseline system in the cache mode installs the missed cache lines in the DRAM cache for future memory accesses.

Flat mode In the second use case, we architect the NM as part of the memory space. The system exposes the NM and the FM as two nodes to the OS, which determines the placement of memory page in and migrates memory pages between the NM and the FM. In our evaluation, we model a virtual memory system that translate the virtual memory address to the physical address and uses a 4KB page size. In this mode, we assume that the OS performs aggressive page migration that moves a page from the FM to the NM on an access to a page in the FM (and a page from the NM to the FM if the NM is full) [21; 32; 35].

3.2 Workloads

We use Pin and SimPoints [33; 40] to capture a representative region of one billion instructions from each workload of various benchmark suites, including SPEC CPU2006 [18; 19], STREAM [34], and high performance computing (HPC) workloads. We number the HPC workloads, which represent weather research and forecast (hpc1), high-performance computing cluster (hpc2), computational fluid dynamics (hpc3), in-cylinder flow and combustion (hpc4), and multi-purpose explicit and implicit finite element analysis (hpc5). We evaluate 20 workloads, including ten memory-intensive SPEC workloads, five STREAM and five HPC workloads. Table 2 shows the characteristics of the 20 workloads used in our study. Note that the working set size is the aggregate size of all 16 cores, which is observed during the simulation, unlike studies that report the resident set size (rss) and the virtual size (vsz) for the entire execution of workloads [16; 19]. The L3 MPKI reflects the bandwidth consumption of evaluated workloads, which is consistent with that of prior work [16; 34].

We evaluate our study by executing benchmarks in the rate mode, which executes the same benchmark on all cores. As we shall see in a later section, the effectiveness of our scheme depends on the relative size of the application working set with respect to the NM capacity. Therefore, we divide the SPEC workload into two categories: Applications whose aggregated working set (for 16 cores) is larger than 4GB are categorized as SPEC_BIG; otherwise, they are categorized as SPEC_SML.

Table 2: Workload Characteristics

Category	Name	L3 MPKI	Aggregate Footprint(GB)
STREAM	add	83	3.58
	triad	73	3.58
	copy	71	2.38
	scalar	64	2.38
	read	43	2.38
HPC	hpc1	82	2.09
	hpc2	48	1.27
	hpc3	46	0.72
	hpc4	43	2.06
	hpc5	30	1.88
SPEC_SML	soplex	64	0.84
	omnetpp	50	2.19
	libq	48	0.50
	leslie	43	1.22
	astar	25	0.58
SPEC_BIG	milc	65	6.70
	lbm	64	6.23
	Gems	53	11.4
	mcf	43	18.5
	bwaves	39	6.61

3.3 Figure of Merit

We measure the total execution time as the figure of merit. As we run the workloads in rate mode, the difference in execution time of the individual benchmark within the workload is negligibly small. We normalize the execution time to the baseline system of the respective modes and report the speedup. Also, as the goal of BATMAN is to control the NM access rate, we also report the access rate of the NM, defined in Equation 1.

4 BATMAN IN THE CACHE MODE

The fundamental mechanism of BATMAN is that it explicitly controls the data movement between the NM and the FM and distributes the memory access proportionally to the respective bandwidth. This section examines systems in the cache mode, develops the fundamental mechanism of BATMAN, and demonstrates BATMAN for such systems, which architects the NM as a *DRAM cache* (L4) between the on-chip L3 cache and the memory (the FM).

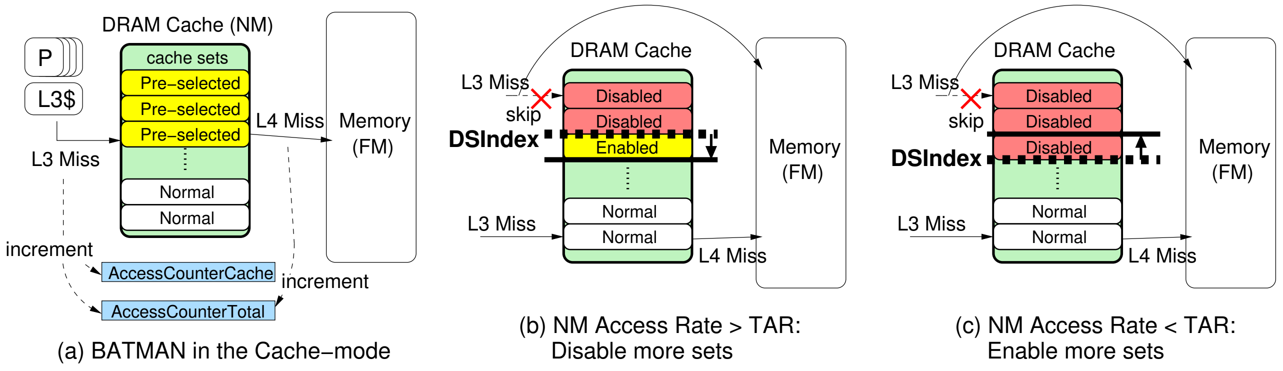

Figure 4: Overview of BATMAN for DRAM caches. (a) BATMAN in the cache mode. BATMAN uses two counters to monitor the NM access rate: *AccessCounterCache* and *AccessCounterTotal*. While one access to the NM increments both counters, one access to the FM increments only the AccessCounterTotal counter. BATMAN selects cache sets as candidates that can be disabled. (b) All pre-selected sets at index lower than DSIndex are "disabled sets," which neither incur a tag look-up nor service any cache request. When the NM access rate is greater than the TAR, DSIndex increases and disables more cache sets. (c) When the NM access rate is lower than the TAR, DSIndex decreases and enables the disabled sets.

4.1 Idea: Controlling the NM Access Rate by Partially Disabling the Cache

The mechanism of BATMAN that controls the NM access rate becomes the regulation of the cache hit rate for DRAM caches. Although other cache operations, such as miss-related and writeback-related operations, also contribute to the NM accesses [11], the cache hit rate is a proxy of the NM access rate in the cache mode. For example, when the DRAM cache has a 100% hit rate, all memory requests are serviced by the NM (DRAM cache), which leads to a 100% NM access rate. To achieve the goal of regulating the NM access rate at TAR, BATMAN monitors the NM access rate at runtime and takes action based on either of the following two cases: (1) an NM access rate higher than the TAR or (2) an NM access rate lower than the TAR. In the first case, BATMAN intentionally lowers the cache hit rate to achieve the TAR. On the other hand, in the second case, BATMAN uses the baseline mechanism, which increases the cache hit rate. Therefore, in either case, BATMAN forces the system to approach the TAR.

One simple way of dynamically regulating the DRAM cache hit rate is via partial cache disabling. When a cache set is disabled, memory accesses to the disabled set would miss in the DRAM cache and use the FM to obtain data. In an extreme case in which all the cache sets are disabled, all memory accesses would miss in the cache and rely on the FM for data, which results in both a 0% cache hit rate and a 0% NM access rate. Therefore, explicitly controlling the number of disabled sets is the key to controlling the NM access rate. When the NM access rate is higher than the TAR, BATMAN disables more cache sets, which lowers the rate. On the other hand, when the NM access rate is lower than the TAR, BATMAN enables the disabled sets, which increases the NM access rate.

To convert an NM access to an FM access, memory accesses to disabled cache sets should not incur an NM access. However, to maintain data integrity between the DRAM cache and the memory, a memory access to a disabled set must check the DRAM cache (a tag lookup via an NM access), ensuring that the most recent copy of requested data is not in the cache, which consumes NM bandwidth. To conserve the bandwidth of such accesses, BATMAN supports the disabling of DRAM cache sets by pre-selecting a subset of the DRAM cache sets as candidates that can be disabled. With the knowledge of pre-selected sets, BATMAN avoids the tag look-up overheads (NM accesses) when memory requests go to disabled sets.

4.2 Design of BATMAN for DRAM Caches

Based on the idea of cache disabling, we develop BATMAN for systems in the cache mode. Figure 4(a) presents an overview of BATMAN for DRAM caches. The key attribute that controls the number of disabled cache sets is the *disabled sets index (DSIndex)*. The pre-selected sets at an index lower than DSIndex are the "disabled sets," which neither incur a tag look-up overhead nor service any cache requests, shown in Figure 4(b). The pre-selected sets at an index higher than DSIndex are enabled sets that can still service cache requests. By moving the DSIndex to different positions, BATMAN controls a fraction of cache sets that remain enabled and thus the NM access rate. BATMAN regulates the movement of DSIndex by monitoring the NM access rate and comparing it to the target access rate (TAR, 80% in our default parameters). We provision the system to monitor the NM access rate and to increase or decrease the DSIndex, as described below:

4.2.1 Structures. BATMAN monitors the access rate of the NM using two 16-bit counters: *AccessCounterCache* and *AccessCounterTotal*. While the AccessCounterCache counter tracks the total number of the NM accesses, including reads, tag look-ups and writebacks, the AccessCounterTotal counter tracks the total number of accesses to the NM and to the FM. The NM access rate is the value of the AccessCounterCache counter divided by the AccessCounterTotal counter. When the AccessCounterTotal counter overflows, we halve (right shift by one) both counters. Figure 4(a)

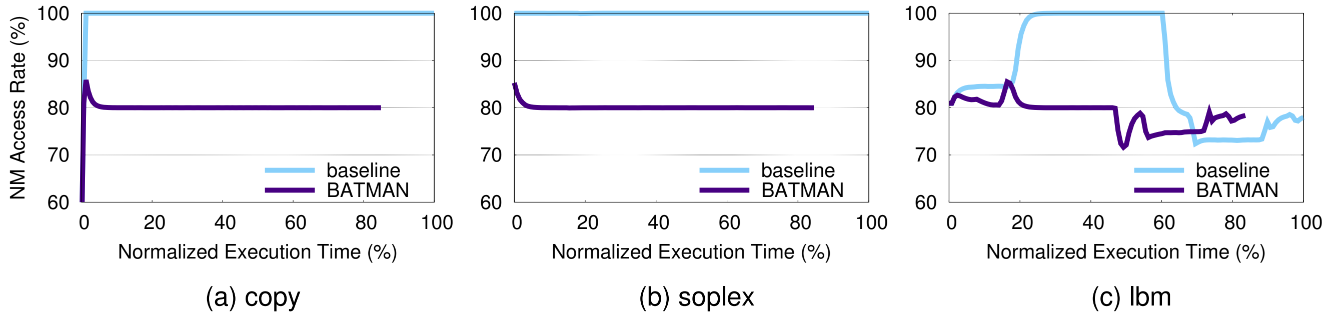

Figure 5: NM Access Rate Versus Time. In each figure, the x-axis is the execution time normalized to the baseline, and the y-axis is the NM access rate recorded per 20-million-cycle interval. We show the NM access rate of both the baseline and BATMAN for three workloads: (a) copy, (b) soplex, and (c) lbm.

shows the tracking counters and their operations. BATMAN requires only two 16-bit counters and a 32-bit DSIndex, which has a negligible storage overhead of eight bytes.

4.2.2 Operation. Memory requests are serviced by either the NM or the FM based on the status of cache set that they access in the DRAM cache. If a L3 miss goes to a disabled set, a pre-selected set whose index is smaller than the DSIndex, (e.g., the top request in Figure 4(b)), the request directly goes to the FM, without the need for a cache look-up. On the other hand, if a L3 miss goes to other sets, either normal sets or pre-selected sets whose index is larger than the DSIndex (e.g., the bottom request in Figure 4(b)), the request follows a normal operation: It looks up the cache set to find the corresponding data block. If the request hits in the cache, the request is serviced by the NM; otherwise, the request goes to the FM for data and installs the data in the cache.

4.2.3 Regulating DSIndex and Hysteresis. BATMAN continuously monitors the NM access rate by computing the ratio of AccessCounterCache to AccessCounterTotal to determine whether the NM access rate exceeds the TAR. If the NM access rate exceeds the TAR, BATMAN increases the DSIndex until the rate is within a 2% guard-band of the TAR. If the NM access rate is lower than the TAR, BATMAN decreases DSIndex until the rate is within the 2% guard-band. For a fast converge time, the length of each DSIndex movement, either an increase or a decrease, is proportional to the delta between the measure NM access rate and the TAR.[1] Before increasing the DSIndex, BATMAN flushes the sets that would be disabled because the index of the sets becomes lower than the DSIndex. Note that the cache flushing overhead includes reading the data from the DRAM cache and writing the data back to the memory should the block is dirty.

[1]We discuss the design of contiguity in disabled sets only for simplicity. In our implementation, BATMAN chooses the candidates (i.e., pre-selected sets) based on hashed indexing. That is, only every *Nth* set is eligible for cache disabling; also, the DSIndex changes by N on a movement. For the default parameters, we use N=5 in our design. The advantage of hashed indexing is the reduction of the traversal time, in which the DSIndex reaches to hot regions of sets that are far away from the DSIndex, by N times.

4.3 The NM Access Rate with BATMAN

As the goal of BATMAN is to regulate the NM access rate, we first show the NM access rate of workloads over time during the execution. We track the NM access rate for every 20 million cycles and show three sampled workloads in Figure 5. In each one, the x-axis is the execution time normalized to the baseline, the y-axis is the NM access rate, and two configurations, the baseline and BATMAN, are shown in the graph. Figure 5(a) and Figure 5(b) are *copy* and *soplex*, which have an almost 100% NM access rate in the baseline; BATMAN is effective at regulating the NM access rate at 80%, the TAR. In addition, Figure 5(c) is *lbm*, with different phases that have various access rates. BATMAN is effective at adapting to the phases and maintaining the NM access rate at the TAR.

In addition to the NM access rate per interval, we also show the average NM access rate for whole execution. Figure 6 presents the NM access rate for both the baseline system and BATMAN. Recall that the baseline system has no disabled cache sets and always installs cache lines in the cache after a cache miss. In the baseline, the access rate of the NM is 95% on average, with many workloads exceeding 90%. Workloads with a high NM access rate is consistent with the reported workloads of Intel Knights Landing [44]. These workloads have a working set size that is smaller than the NM capacity, so their working set fits in the NM, which results in a high NM access rate. In contrast, for workloads whose working set size is larger than the NM (SPEC_BIG), the NM access rate is not as high. For those workloads whose NM access rate is over the TAR, BATMAN consistently regulates the rate at the TAR, 80% in this case. For other workloads, BATMAN has negligible changes of the NM access rate (within 1%).

4.4 Performance Improvement from BATMAN

Figure 7 shows the speedup of BATMAN with respect to the baseline in the cache mode. BATMAN improves the overall system bandwidth in two ways: First, the bandwidth of the FM becomes usable. Second, misses to the disabled sets that would have been misses in the baseline too would now avoid the NM bandwidth of miss-related and writeback-related operations. As BATMAN

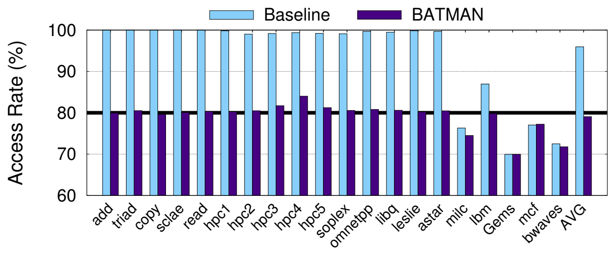

Figure 6: The NM Access Rate of the Baseline and BATMAN

improves overall system bandwidth, BATMAN provides an average speedup of 11%. More specifically, for applications whose working set fits in the cache, BATMAN consistently improves performance by as much as 23%. In addition, for applications whose working set is larger than the NM, BATMAN accurately captures and effectively reacts to the phases. For example, for *lbm*, we observe that adjusting the DSIndex captures different phases, shown in Figure 5(c), which results in 24% performance improvement.

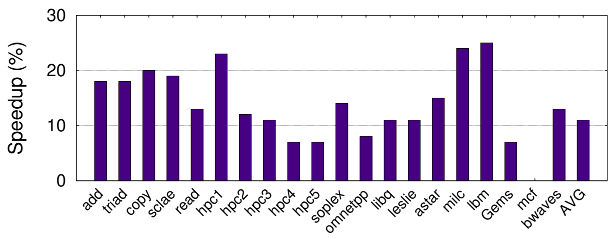

Figure 7: Speedup with BATMAN in the Cache Mode

5 BATMAN IN THE FLAT MODE

BATMAN works for systems not only in the cache mode but also in the flat mode. Recall that in the flat mode, the system uses the NM as part of the memory space and relies on the operating system to performs dynamic page migration for data locality [21; 35]. Page migration relieves the system from being sensitive to the initial placement of memory pages. For example, although a frequently access page may be initially placed in the FM, an access to the page transfers the accessed page from the FM to the NM, which allows subsequent memory accesses to the page to be serviced by the NM. When the size of the application working set is smaller than the capacity of the NM, a page migration scheme will eventually move the entire working set to the NM, thus underusing the FM bandwidth. Even when the working set does not fit in the NM, the page migration scheme would move frequently accessed data to the NM so that almost all memory requests are serviced by the NM. Ideally, we want page migration for data locality and yet ensure that the bandwidth utilization of both the NM and FM is balanced at the target access rate (TAR).

5.1 Idea: Regulate Direction of Migration

We apply the idea of BATMAN that explicitly controls data movement to meet the TAR to systems in the flat mode. In flat mode, page migration, in either direction from the NM to the FM or from the FM to the FM, is in the control of the OS. In our baseline system, the OS aggressively migrates a page from the FM to the NM on an access to the FM (also moves a page from the NM to the FM if NM is full). To control the data movement, BATMAN adds the constraint of the NM access rate when the OS is migrating the page because the NM access rate depends on the direction of the page migration: Migrating pages from the FM to the NM, referred to as *page upgrade*, increases the NM access rate; similarly, downgrading pages from the NM to the FM reduces the NM access rate. Therefore, BATMAN monitors the NM access rate at runtime and provides the information to the OS that decides the direction of page migration. Through controlling the direction of page migration, BATMAN explicitly controls data movement and enforce the NM bandwidth utilization at the TAR.[2]

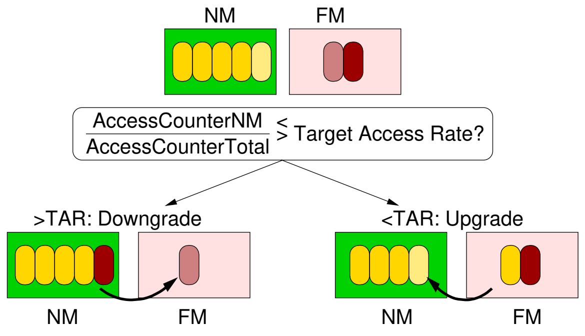
NM
FM
AccessCounterNM / AccessCounterTotal < > Target Access Rate?
>TAR: Downgrade
<TAR: Upgrade
NM
FM
NM
FM

Figure 8: Overview of BATMAN in the flat mode: monitor the access rate of the NM and change the direction of page migration

5.2 BATMAN Design for Flat Mode Systems

The key idea of BATMAN for flat-mode systems is to regulate the direction of page migration based on the NM access rate. If the NM access rate is lower than the TAR, BATMAN upgrades the accessed pages from the FM to the NM; otherwise, BATMAN downgrades pages from the NM to the FM. Figure 8 shows an overview of BATMAN for flat-mode systems. The default system has a 4x-bandwidth NM and a 1x-bandwidth FM, so the TAR is 80%. We provision the system to monitor the access rate of the NM and to decide to upgrade or downgrade pages, as described below:

5.2.1 Structures. BATMAN dynamically monitors the fraction of total memory accesses that are serviced by the NM by using two counters: *AccesssCounterNM* and *AccessCounterTotal*, which count the number of accesses to the NM and to the system (both the NM and the FM), respectively. The ratio of the AccessCounterNM counter to the AccessCounterTotal counter shows the fraction of memory accesses serviced by the NM. Note that both these counters account for all memory activity and increments on demand,

[2] Although we assume an aggressive page migration policy, BATMAN can be applied to other policies, too. For example, for a frequency-based page migration strategy [35], BATMAN allows up to 80% accesses to be serviced by the NM.

prefetch, or writeback memory requests. In our design, we use two 16-bit hardware registers for the counters. When the AccessCounterTotal counter overflows, we halve (right shift by one) both counters. Therefore, BATMAN requires a storage overhead of only four bytes (two 16-bit counters) to track the accesses to the NM and the system.

5.2.2 Operation. On each access, we compute the ratio of AccesCounterNM to AccessCounterTotal, which overlaps with the memory address translation. If this ratio is less than the TAR, we want to increase the access rate of the NM: If the memory request goes to FM, BATMAN upgrades the requested page from the FM to the NM. Similarly, if the ratio is greater than the TAR, we want to reduce the access rate of the FM: If the memory request accesses a page in the NM, BATMAN downgrades the requested page from the NM to the FM. BATMAN leverages the existing OS support for page migration between the NM and the FM. Regulating such downgrade and upgrade operation ensures that the NM access rate becomes close to the TAR. Note that BATMAN still utilizes all the memory capacity by downgrading a page to the FM, instead of evicting it to the storage.

5.2.3 Hysteresis on Threshold. Using a single threshold of the TAR leads to frequent switching between upgrade and downgrade modes. When the NM access rate is close to the TAR, the system can continuously switch between upgrade and downgrade operation. To avoid this oscillatory behavior, we provision a guard band of 2% in either direction in the decision of upgrade and downgrade. Therefore, page upgrades occur only when the measured access rate of the NM is less than (TAR-2%) and downgrades occur only when the measured access rate of the NM exceeds (TAR+2%). In the intermediate zone, between (TAR-2%) and (TAR+2%), BATMAN does not perform either upgrades or downgrades.

5.3 Effectiveness of BATMAN at Reaching TAR

Figure 9 shows the NM access rate for the baseline system with page migration and BATMAN. In the baseline, all workloads have a NM access rate close to 100%, even for the SPEC_BIG benchmarks suites whose working set is larger than the NM capacity because these workloads have high spatial locality within a page. For example, after an accesses to the FM upgrades a page to the NM, if the next 15 memory references go to other lines in the page, the NM access rate is as high as 15/16, or close to 94%. For other applications, as their working set fits in NM, all the pages are transferred to the NM and provides a 100% NM access rate. BATMAN redistributes some of the memory traffic to the FM, and balance the system based on the bandwidth ratio of the NM and the FM. For all workloads, BATMAN effectively obtains a NM access rate close to the target value of 80%. Thus, our proposed mechanism of BATMAN is effective at maintaining the NM access rate at the TAR, with a simple monitoring logic at runtime.

5.4 Performance Improvement from BATMAN

Figure 10 shows the speedup from BATMAN compared to the baseline system that always performs page migration between the NM and the FM on an access to the FM. BATMAN improves performance of all workloads by an average of 10%. The performance

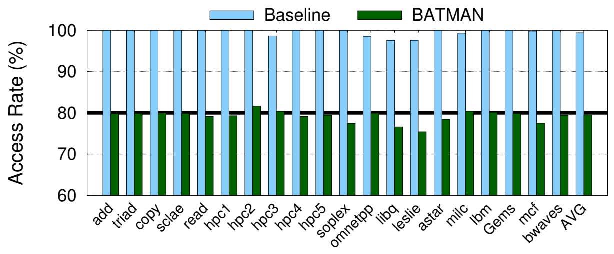

Figure 9: Access rate of the near memory. BATMAN enforces the NM access rate close to the TAR (80%) for all workloads.

improvement comes from two factors: First, for workloads whose working set fits in the NM, BATMAN effectively uses both NM and FM bandwidth and thus has higher throughput for those workloads. All HPC, STREAM and SPEC_SML (*soplex-astar*) workloads belong to this group. Second, for workloads form the SPEC_BIG category, whose working set is larger than the NM capacity, (e.g., bwaves-milc), BATMAN reduces the number of page migration between the NM and the FM because BATMAN allows only a certain number of page upgrades. As page migration consumes significant memory bandwidth on both the NM and FM, BATMAN helps reduce the bandwidth usage for this group of workloads. Therefore, BATMAN improves performance regardless of the working set of the application.

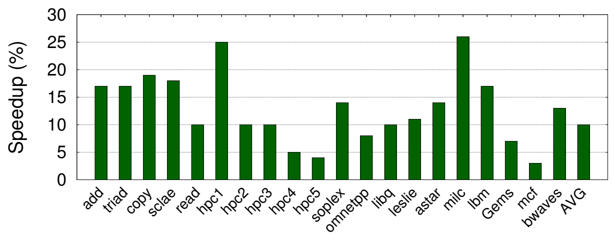

Figure 10: Speedup of BATMAN for systems in the flat mode

6 RESULTS AND ANALYSIS

6.1 Sensitivity Study for Bandwidth Ratio

In our default system, we assume the Near Memory has 4X bandwidth as that of the Far Memory, which is similar to the recent industry product [43–45]. A recent report indicates that the next generation of 3D-DRAM provides two lines of products [47]: high-end HBM3, which offers as approximately 8X high bandwidth as DDR4, and low-cost HBM2, which provides as 2X high bandwidth as DDR4. Therefore, we conduct a sensitivity study by varying the bandwidth ratio provided by the NM from 2X to 8X. As only the NM bandwidth changes, we vary the bandwidth ratio by fixing the number of DRAM channels in the FM and varying the number of DRAM channels in the NM. That is, for an 8X ratio, the number of channels in the NM is as eight times as that in the FM. Table 3 shows the comparison of configurations. Notice that when the bandwidth ratio is 2X, the bandwidth increase from BATMAN is as high as 50%, which strongly suggests that when the FM bandwidth accounts for a significant fraction of overall system bandwidth, we should optimize the system for overall bandwidth, not the NM access rate.

Table 3: Sensitivity Study of Bandwidth Ratio

NM BW	FM BW	Utilized System BW Baseline	Utilized System BW BATMAN	BW Increase
2X	1X	2X	3X	**+50%**
4X	1X	4X	5X	+25%
8X	1X	8X	9X	+12.5%

Shown in Figure 11, when the ratio is 2X, BATMAN improves performance by more than 30% because the aggregate bandwidth is as 1.5 times high bandwidth as that of the NM, providing a huge increase in available bandwidth. When the ratio is 8X, the room for improvement is small because using the FM bandwidth increases total bandwidth by 12.5% and provides 9% and 5% speedup for systems in the cache and flat mode, respectively.

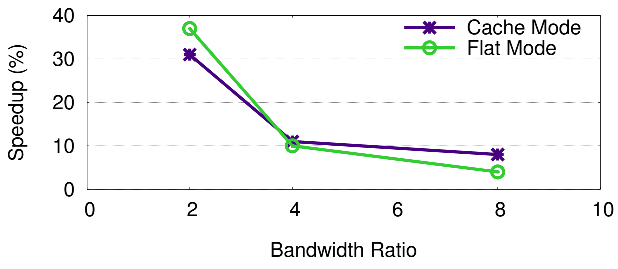

Figure 11: Sensitivity to the relative bandwidth of the FM: We fix the FM bandwidth and vary the NM bandwidth from 2X to 8X. Each configuration is normalized to the respective baseline. When the NM bandwidth is as 2X high as the FM bandwidth, BATMAN improves overall bandwidth by 50% and provides more than 30% speedup.

6.2 Power and Energy Analysis

We analyze the power consumption and the energy-delay product (EDP) for both the cache and flat modes. As the FM is based on DDR3 [37], we use the Micron DDR3 DRAM power calculator for the power estimation of the FM, and we estimate the NM power based on Micron 3D-DRAM technology [4]. For the power measurement, we conservatively assume the memory system consumes 30% of the system power and the remaining system consumes 70% of the system power [48].[3]

[3] For systems that have high memory intensity, the memory system may consume more than 30% of the system power. Although we conservatively assume 30%, BATMAN could further improve system energy and energy-delay product if the memory system consumes more power.

Figure 12 shows the power consumption and the energy-delay product for the baseline and BATMAN in two modes. Note that each configuration is normalized to the respective baselines. With BATMAN, the overall power consumption of the system increases by 7% and 10% for systems in the cache and the flat mode, respectively. The power consumption contributed by BATMAN comes from two reasons. First, BATMAN reduces the execution time, which increases power because the all system activity must happen within a reduced time interval. Second, BATMAN exploits the FM bandwidth in the system and incurs the active power of the FM. However, lower execution time reduces energy consumption; BATMAN reduces the energy consumption by 5% and 1% for two modes, respectively. Also, BATMAN improves the EDP of the system by 13% and 11% for two modes, respectively.

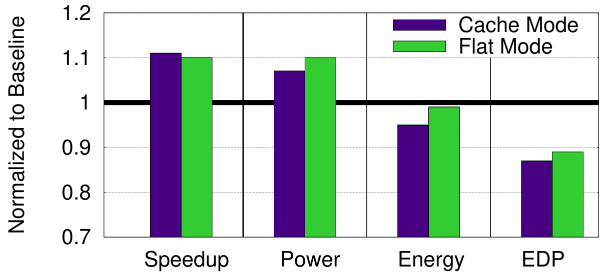

Figure 12: Speedup, power, energy, and EDP with BATMAN (numbers normalized to respective baseline)

7 RELATED WORK

7.1 Bandwidth Optimization

The work that is most closely related to ours is a recent study by Agarwal et al. [1], who investigate static page placement strategies for maximizing the bandwidth of a tiered-memory system for GPU workloads. They provide programmers annotations to the GPU codes that mark which data structures should be in the NM and which should be in the FM. They maximize the overall system bandwidth for systems with *static* page mapping. Our work differs from theirs in two key aspects:

1. BATMAN does not rely on programmer annotations or knowledge of the data structure of a given program. We show that explicitly controlling the direction of page migration is sufficient for the desired NM access rate, which makes our proposal applicable to a wide variety of applications, including those whose source codes may not be available.
2. Their proposed scheme is applicable to only systems that do not perform data migration and would not be useful for cases when the NM is used as a hardware-managed cache (Section 4) or when the memory system supports dynamic page migration (Section 5). However, we show that BATMAN is useful even if the system is in the cache or flat mode. Thus, BATMAN , which is applicable to a

much wider range of systems, is not restricted to systems with static page mapping.

We compare BATMAN to the prior work and show the performance in Figure 13. We model a system in the flat mode, and the baseline is a static page placement policy that places memory pages in the NM until the NM is full. (This baseline is referred to as *LOCAL* in the prior work.) We model the prior work that uses static bandwidth-aware page placement strategy (labeled *Static BW-AWARE*), and BATMAN in the flat mode with page migration. For workloads whose working set fits in the NM (*add-astar*), the performance of BATMAN is very close to that of the prior work (all within 1%), but BATMAN does not require any software modification. However, for workloads whose working set is larger than the NM (*milc-bwaves*), BATMAN significantly outperforms the prior work for two reasons: First, static page placement is ineffective when the working set that accounts for 80% of memory accesses is larger than the NM size. For example, we found that *lbm* and *Gems* have such working set larger than 4GB. Second, BATMAN is a runtime mechanism that is able to capture phases in programs and adjust the control of data movement. On average, the prior work improves performance by 14% while BATMAN provides a speedup of 29% over the baseline.

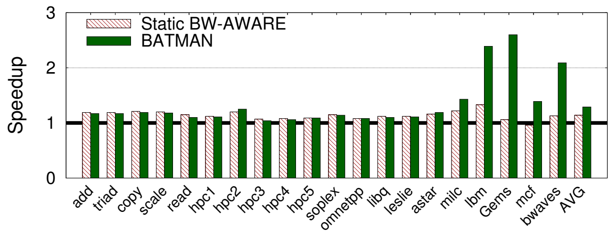

Figure 13: Performance comparison of prior work (labeled *Static BW-AWARE*) and BATMAN.

7.2 DRAM Cache

To use the FM bandwidth, Sim et al. proposes mostly-clean DRAM cache [42], which uses a cache-miss predictor and accesses the FM in parallel with the cache access. The goal of their work is to mitigate the high cache-miss latency in case the cache has significant amount of contention. However, as their scheme accesses both DRAM cache (NM) and the memory (FM) for one memory request, it does not save the NM traffic. In fact, the scheme increases the memory traffic and consumes more memory bandwidth (two accesses per request). Thus, their proposal does not increase the aggregate memory bandwidth of the system. In contrast, BATMAN avoids the tag lookup when the cache set is disabled and incurs only one access for each such request, thereby increasing the overall bandwidth of the system.

7.3 Software Optimization

In the context of multi-node systems, several page migration policies have been proposed in the last decades to improve the performance of non-uniform memory architecture (NUMA), where the local nodes are an order of magnitude faster in terms of latency than the remote nodes [5; 6; 29; 30; 35; 46]. As node traversal incurs a significant overhead in a multi-node system, optimizing the locality of compute (threads) and data (memory page allocation) effectively improves performance [30; 46]. However, in the context of tiered-memory system, all data are in the same node but different memory component. Meswani et al. proposes a hardware-software mechanism that tracks the hot pages in the FM and moves the pages to the NM to maximize the NM access rate. In constrast, BATMAN optimizes data allocation between two memory components in a single-node system to improve overall bandwidth. However, the studies in multi-node systems are orthogonal to BATMAN, and future studies of multi-node systems should account for a multi-node system, where each node has two memory components.

Other studies propose software optimization that optimizes for latency reduction in a NUMA environment. Golub and Van Loan uses cache blocking mechanism to fit the working set size in the NM [15] , while Blagodurov and Fedorova propose use-level thread scheduling to align compute and data in the same node [3]. However, in the case of tiered-memory systems with a high-bandwidth NM, which has a similar latency to that of the FM, we show that computer and system architects may need to revisit these proposals. For example, although cache-blocking for the NM is useful, an intentional placement of certain part of the blocked working set in the FM improves performance.

8 CONCLUSION

Emerging 3D-DRAM technology, such as HBM and HMC, provides as 4x to 8x high bandwidth as the commodity DDR-based DRAM. The technology is used in tiered-memory systems, in which the *near memory (NM)* is a high-bandwidth memory and the *far memory (FM)* is a high-capacity DDR-based memory. In such systems, conventional management approaches for tiered-memory systems focus on improving the number of memory requests serviced by the NM. However, such techniques under-utilize the FM bandwidth, especially when the frequently-accessed working set fits into the NM. We show that system bandwidth and performance are maximized when memory accesses are split between the NM and the FM proportional to the bandwidth of each memory. To this end, this paper makes the following contributions:

(1) To the best of our knowledge, this is the first study that proposes a runtime mechanism to maximize system bandwidth for tiered-memory systems. We leverage the key insight that the control of data movement can regulate the NM access rate and propose *Bandwidth-Aware Tiered-Memory Management (BATMAN)*, which explicitly controls the data movement between the NM and the FM to achieve the desired NM access rate.
(2) We apply BATMAN to systems that use the NM as a hardware-managed cache (cache mode). BATMAN tracks the NM access rate at runtime and disables a fraction of the cache sets to obtain the target access rate. We show that

adjusting the DSIndex at runtime regulates the NM access rate and also adapts to dynamic phases of workloads.

(3) We also apply BATMAN to systems that use the NM as part of memory space and migrate pages between the NM and the FM (flat mode). Monitoring the NM access rate at run-time and dynamically controlling the direction of page migration is highly effective at reaching the target access rate, thus improving performance.

We demonstrate that BATMAN are applicable to systems in both modes, and our proposed implementation is simple and highly effective at maintaining the NM access rate at the TAR. BATMAN incurs a storage overhead of only eight bytes and requires negligible software support. Our studies on a 16-core system with a 4GB NM and a 32GB FM show that BATMAN improves performance for systems in the cache and flat mode by an average of 11% and 10%, respectively; also, BATMAN improves the system energy-delay-product by 13% for systems in the cache mode and 11% for systems in the flat mode.

ACKNOWLEDGEMENTS

This work was supported in part by NSF grant 1319587 and the Center for Future Architecture Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

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. *SIGARCH Comput. Archit. News* 43, 1, 607–618. https://doi.org/10.1145/2786763.2694381

[2] Frank Bellosa. 2004. When physical is not real enough. In *Proceedings of the ACM SIGOPS European workshop.*

[3] Sergey Blagodurov and Alexandra Fedorova. 2011. User-level Scheduling on NUMA Multicore Systems under Linux. In *in Proc. of Linux Symposium.*

[4] Jag Bolaria. 2011. Micron Reinvents DRAM Memory. *Microprocessor Report* (2011).

[5] William J. Bolosky, Michael L. Scott, Robert P. Fitzgerald, Robert J. Fowler, and Alan L. Cox. 1991. NUMA Policies and Their Relation to Memory Architecture. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV).* ACM, New York, NY, USA, 212–221. https://doi.org/10.1145/106972.106994

[6] 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.org/10.1145/195473.195485

[7] D. W. Chang, G. Byun, H. Kim, M. Ahn, S. Ryu, N. S. Kim, and M. Schulte. 2013. Reevaluating the latency claims of 3D stacked memories. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific.* 657–662.

[8] Niladrish Chatterjee, Rajeev Balasubramonian, Manjunath Shevgoor, Seth H. Pugsley, Aniruddha N. Udipi, Ali Shafiee, Kshitij Sudan, and Manu Awasthi. 2012. *USIMM.* University of Utah.

[9] Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2015. *BATMAN: Maximizing Bandwidth Utilization of Hybrid Memory Systems.* Technical Report. School of Electrical and Computer Engineering, Georgia Institute of Technology.

[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 Microarchitecture (MICRO-47).* IEEE Computer Society, Washington, DC, USA, 1–12. https://doi.org/10.1109/MICRO.2014.63

[11] Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2015. BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15).* ACM, New York, NY, USA, 198–210. https://doi.org/10.1145/2749469.2750387

[12] S.K. De, R.A. Stewart, G.C. Cascaval, and D.T. Chun. 2015. System and method for allocating memory to dissimilar memory devices using quality of service. (July 28 2015). US Patent 9,092,327.

[13] 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

[14] Magnus Ekman and Per Stenstrom. 2004. A Case for Multi-level Main Memory. In *Proceedings of the 3rd Workshop on Memory Performance Issues: In Conjunction with the 31st International Symposium on Computer Architecture (WMPI '04).* ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/1054943.1054944

[15] G.H. Golub and C.F. Van Loan. 1996. *Matrix Computations.* Johns Hopkins University Press.

[16] Darryl Gove. 2007. CPU2006 Working Set Size. *SIGARCH Comput. Archit. News* 35, 1 (March 2007), 90–96.

[17] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan. 2014. Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture.* 38–50. https://doi.org/10.1109/MICRO.2014.36

[18] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17. https://doi.org/10.1145/1186736.1186737

[19] John L. Henning. 2007. SPEC CPU2006 Memory Footprint. *SIGARCH Comput. Archit. News* 35, 1 (March 2007), 84–89.

[20] HMCC. 2013. *HMC Specification 1.0.* http://www.hybridmemorycube.org

[21] M. A. Holliday. 1989. Reference History, Page Size, and Migration Daemons in Local/Remote Architectures. In *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III).* ACM, New York, NY, USA, 104–112. https://doi.org/10.1145/70082.68192

[22] Hai Huang, Padmanabhan Pillai, and Kang G. Shin. 2003. Design and Implementation of Power-aware Virtual Memory. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '03).* USENIX Association, Berkeley, CA, USA, 5–5. http://dl.acm.org/citation.cfm?id=1247340.1247345

[23] Intel. 2013. *Intel Core i7 Processor.* http://www.intel.com/processor/corei7/specifications.html

[24] JEDEC. 2013. *DDR4 SPEC.*

[25] JEDEC. 2014. *High Bandwidth Memory (HBM) DRAM, Gen 2.*

[26] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture.* 25–37. https://doi.org/10.1109/MICRO.2014.51

[27] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13).* ACM, New York, NY, USA, 404–415. https://doi.org/10.1145/2485922.2485957

[28] Dimitris Kaseridis, Jeffrey Stuecheli, and Lizy Kurian John. 2011. Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44).* ACM, New York, NY, USA, 24–35. https://doi.org/10.1145/2155620.2155624

[29] Richard P. Larowe, Jr. and Carla Schlatter Ellis. 1991. Experimental Comparison of Memory Management Policies for NUMA Multiprocessors. *ACM Trans. Comput. Syst.* 9, 4 (Nov. 1991), 319–363. https://doi.org/10.1145/118544.118546

[30] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. 2015. Thread and Memory Placement on NUMA Systems: Asymmetry Matters. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15).* USENIX Association, Berkeley, CA, USA, 277–289. http://dl.acm.org/citation.cfm?id=2813767.2813788

[31] Gabriel H. Loh and Mark D. Hill. 2011. Efficiently Enabling Conventional 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

[32] Gabriel H. Loh, Nuwan Jayasena, Jaewoong Chung, Steven K. Reinhardt, J. Michael OConnor, and Kevin McGrath. 2012. Challenges in Heterogeneous Die-Stacked and Off-Chip Memory Systems. In *3rd Workshop on SoCs, Heterogeneous Architectures and Workloads.*

[33] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05).* ACM, New York, NY, USA, 190–200. https://doi.org/10.1145/1065010.1065034

[34] John D. McCalpin. 1991. *STREAM: Sustainable Memory Bandwidth in High Performance Computer.* http://www.cs.virginia.edu/stream/

[35] M.R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G.H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*.

[36] Micron. 2010. *1Gb_DDR3_SDRAM*.

[37] Micron. 2012. *Calculating DDR Memory System Power Introduction*.

[38] Micron. 2014. *HMC Gen2*. Micron.

[39] NVIDIA. 2014. *NVIDIA Pascal*. http://blogs.nvidia.com/blog/2014/03/25/gpu-roadmap-pascal/

[40] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. 2003. Using SimPoint for Accurate and Efficient Simulation. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '03)*. ACM, New York, NY, USA, 318–319. https://doi.org/10.1145/781027.781076

[41] 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

[42] 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 International Symposium on Microarchitecture*. 11. https://doi.org/10.1109/MICRO.2012.31

[43] Avinash Sodani. 2015. Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. (Hot-Chips 2015). http://tinyurl.com/hotchips-2015-sodani

[44] Avinash Sodani. 2016. Knights Landing Intel Xeon Phi CPU: Path to Parallelism with General Purpose Programming. (Keynote Address HPCA 2016).

[45] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. 2016. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro* 36, 2 (Mar 2016), 34–46.

[46] Sharanyan Srikanthan, Sandhya Dwarkadas, and Kai Shen. 2015. Data Sharing or Resource Contention: Toward Performance Transparency on Multicore Systems. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15)*. USENIX Association, Berkeley, CA, USA, 529–540. http://dl.acm.org/citation.cfm?id=2813767.2813807

[47] Kevin Tran. 2016. The Era of High Bandwidth Memory. In *Hot Chips: A Symposium on High Performance Chips*.

[48] Thomas Vogelsang. 2010. Understanding the Energy Consumption of Dynamic Random Access Memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '43)*. IEEE Computer Society, Washington, DC, USA, 363–374.