PRISM: Architectural Support for Variable-granularity Memory Metadata

Rachata Ausavarungnirun
King Mongkut's University of Technology North Bangkok

Timothy Merrifield
VMware

Jayneel Gandhi
VMware Research Group

Christopher J. Rossbach
University of Texas at Austin
VMware Research Group

ABSTRACT

Modern architectures track memory accesses using page granularity metadata such as access and dirty bits, leading to fundamental tradeoffs for system software that uses this metadata. Larger page sizes reduce address translation overheads and page table footprints. However, coarse metadata bits for larger pages limit software's visibility into application-level memory usage, resulting in memory bloat and performance pathologies. As DRAM capacity continues to expand, we expect software to react by aggressively mapping with larger page sizes, making this tradeoff space more challenging to navigate.

We study the relationship between metadata granularity and *fidelity*, the degree to which metadata correctly approximates actual access patterns. We focus on 2MB page support on x86-64 and GPUs, measuring fidelity across a wide range of benchmarks. Fidelity can be poor at a coarse granularity, and high variance occurs even within applications. To address this problem, we propose Prism, which provides architectural support for variable–granularity access and dirty bits. Evaluation of Linux/x86-64 and GPU prototypes of Prism show modest additional hardware can reduce metadata fidelity loss by up to 65% and 55% at a performance cost of less than 1% and 2% on CPUs and GPUs respectively. We show that the recovered fidelity can eliminate performance pathologies and improve the performance of GPGPU applications using demand paging by 29.8% on average.

CCS CONCEPTS

• **Software and its engineering** → **Virtual memory**; • **Computer systems organization** → *Parallel architectures.*

KEYWORDS

address translation; virtual memory management; memory protection; large pages

PACT '20, October 3–7, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8075-1/20/10…$15.00
https://doi.org/10.1145/3410463.3414630

ACM Reference Format:
Rachata Ausavarungnirun, Timothy Merrifield, Jayneel Gandhi, and Christopher J. Rossbach. 2020. PRISM: Architectural Support for Variable-granularity Memory Metadata. In *Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques (PACT '20), October 3–7, 2020, Virtual Event, GA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3410463.3414630

1 INTRODUCTION

System software depends on memory access metadata such as access and dirty bits to manage memory, using metadata to estimate how frequently a page is accessed or modified. These metadata are essential for subsystems such as transparent huge page support (THP [27, 38]), page migration between NVM and DRAM [30], and same-page merging (KSM [55]). For example, OS-level page reclamation reallocates infrequently accessed pages to processes that can better utilize them. The accuracy of the metadata used to estimate usage patterns determines the ability of such subsystems to improve application performance and system utilization. Metadata *granularity* has a first-order impact on that accuracy.

Current systems rely on architectural support for per-page access, and dirty metadata bits, set by the hardware and read by software. In a traditional system, using uniform, small page sizes (4KB), page-granularity bits suffice to accurately characterize memory access patterns. However, recent explosive growth in DRAM capacities has forced hardware vendors and system software developers to revisit larger (e.g. 2MB[1]) page sizes to keep address translation overheads in check. Relative to 4KB pages, tracking metadata at 2MB-page granularity can entail up to a 512× accuracy loss due to approximation because a single bit must characterize access patterns over a much larger extent of memory. Recent studies exemplified this accuracy loss for dirty bits for the use cases of VM live migration and RDMA based remote memory [25] and for access bits used by OSes and hypervisors [59, 71].

Historically, OSes have struggled to support multiple page sizes efficiently due to inaccurate, noisy, or unavailable metadata. This exposes workloads to performance overheads, memory bloat, and unfairness problems so severe that sysadmins often disable huge pages in production [1, 28, 62, 65, 73, 78, 88, 95]. Without accurate metadata, critical subsystems can easily exhibit unpredictable or pathological behavior because the OS is unable to discern whether large pages are frequently accessed or whether contiguity is well-utilized. Huge page management remains an active area of research [59, 67, 71] because of the lost accuracy that is fundamental

[1]We follow Linux parlance by calling 2MB pages "huge pages".

to coarse metadata granularity. However, as memory capacities continue to escalate, the performance price of simply turning off huge pages will become unacceptable.

This paper proposes to decouple memory metadata granularity from page size. We focus on access and dirty bits for 2MB huge pages on CPUs and GPUs. We explore the granularity requirements of modern applications by measuring their metadata *fidelity loss*, defined as the percentage of bits incorrectly approximated at coarse granularity relative to 4KB base pages. Empirically, fidelity loss has high variance across and within individual workloads. We propose a design for software-controlled variable granularity access and dirty bits. We extend page table formats to enable optional additional page table pages that store additional metadata bits at a granularity chosen by the software. For legacy software that uses traditional page-granularity metadata, the design is transparent and incurs no performance or memory overheads. For emerging software that leverages finer-grain metadata, performance, area, and energy overheads are negligible with fidelity recovered. We make the following contributions:

- We provide a comprehensive study of metadata fidelity loss for huge pages for CPUs and GPUs. We demonstrate high inter- and intra-application variance, motivating the need for system-controlled metadata granularity.
- We propose a design to decouple memory metadata from page size, enabling per-page control over granularity for 2MB huge pages. The design reduces the fidelity lost on average from 65% and 55% to 14% and 13% for access and dirty bits respectively on CPUs, enabling accurate software visibility into memory access patterns while preserving performance benefits of large pages.
- We show that our design generalizes across CPUs and GPUs. Our design reduces fidelity loss on a GPU and improves demand-paging on a GPU by 29.8% on average.

2 MOTIVATION

Commodity processors have supported multiple page sizes for decades [83]. For example, x86-64 supports 1GB, 2MB, and 4KB mappings. Recent technology trends such as explosive DRAM capacity growth and anticipated large NVM capacities incentivize the use of larger pages to improve performance [6, 10, 15, 34, 59, 63] for large memory workloads by reducing TLB-miss related overheads.

The burden of managing multiple page sizes falls on software, whether decisions about which page size to use are relegated to applications [56], or made *transparently* by the OS or hypervisor [27, 42, 59, 67, 71]. Modern OSes and hypervisors aggressively map pages with larger page sizes [27, 42, 57, 59] to increase TLB reach and reduce TLB misses. However, improperly managed, huge pages can cause memory bloat, performance variability, and unfairness, often doing so by allocating huge pages to applications that cannot actually benefit from them (e.g., due to high locality or small working sets). Huge page mappings can reduce the effectiveness of system software that depends on per-page metadata.

Memory access frequency: Access frequency approximation or "hotness" of a memory region informs several OS subsystems including page reclamation and working set estimation (`kswapd` and `kpageidle` in Linux, respectively). For future system designs such as multi-tier memory hierarchies that treat DRAM as a cache for persistent memory [30], access frequency will determine placement in the memory hierarchy to hide the increased latencies of persistent memory. Application-level trends toward larger memory footprints have driven the emergence of multi-socket servers with up to 100s of TB of memory, with NUMA effects that cause bandwidth and latency to differ by 2× to 8×. Similar effects will impact newer architectures using multi-chip modules (MCMs) [29, 48, 51, 98] and will likely begin to impact GPUs [9, 64].

Migration and replication of data pages and page-tables are commonly used to ameliorate the performance impact of NUMA [2, 23, 87, 94] effects, but policies depend critically on access frequency metadata. When a single access bit is read periodically to determine the hotness of an entire 2MB region, pages can easily appear artificially hot. Figure 1 shows the measured hotness of huge pages for a variety of benchmarks using access bits read once per 100 million instructions retired. For the majority of benchmark programs, *nearly all* huge pages are identified as hot (e.g. between 0.9-1.0). For some programs, the results are bi-modal since huge pages are identified as either hot or cold. But except for `blackscholes`, huge pages rarely fall in between. We show in §3 that this characterization of hotness is dramatically inaccurate because of poor access metadata fidelity.

Fragmentation and Bloat: Applications may be unable to completely utilize larger pages allocated to them by the OS. Linux THP, for example, allocates huge pages optimistically and greedily, despite the risk of underutilized huge pages causing internal fragmentation. A process using only part of a huge page still reserves the entire region, and page granularity metadata makes it impossible for the OS to detect such situations. The resulting memory bloat leads administrators to disable THP in production workloads that could otherwise realize performance boosts [1, 28, 62, 65, 73, 78, 88, 95]. Access metadata maintained at finer granularity by the hardware could easily solve the problem [59, 71].

Page Migration: System-level services such as swapping to disk and VM migration transparently migrate the contents of a physical page from one location to another (e.g., from DRAM to hard disk or from one physical host to another). A single dirty bit per 2MB or 1GB huge page gives system software a very coarse hint about whether the contents of a page have been updated, forcing software to choose between blindly transferring page contents which may not have been updated, or inducing overheads to detect dirty subpages in software [67]. Finer grain dirty metadata can dramatically improve the efficiency and simplicity of subsystems depending on memory migration primitives.

Huge page promotion/demotion: OS-level transparent huge page management treats memory contiguity as a first-class allocatable resource [59, 67, 71]. Discerning whether a huge page is well-utilized requires information about memory access patterns (i.e., metadata) so the OS or the hypervisor can allocate a huge page fairly and performance-profitably. Pathologies in transparent huge page support are the inevitable result of poor metadata fidelity of current architectures.

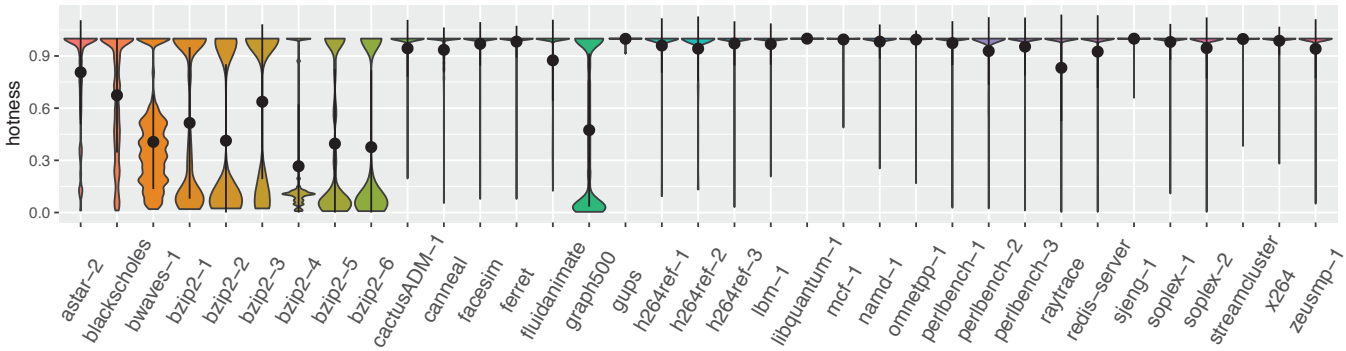

Figure 1: Large page "Hotness" for SPEC, Parsec, Redis, and Graph500 with access bits read once per 100 million instructions.

Why software cannot solve the problem? OS techniques such as Ingens [59], Hawkeye [71], and huge pages in FreeBSD [67] implement OS heuristics to detect when a process *may* benefit from a huge page promotion but induce software overheads and additional complexity while failing to completely solve the problem. Heuristics do not help with the symmetric problem of deciding when to *demote* huge pages based on poor utilization. Better hardware support is likely the only viable alternative to complex schema such as costly proactive, speculative, and/or probabilistic demotion [42]. PTE emulation, splintering, and sampling are some options that induce complexity and performance overheads without completely solving the problem.

benchmark suite	description
Parsec [22]	Multi-threaded/shared-memory.
SPEC [90]	SPEC CPU 2006
Redis [77]	Redis server benchmark
Graph500 [41]	Graph processing benchmark
GUPS	Memory intensive stress benchmark
Rodinia [26]	GPU benchmarks
Parboil [91]	GPU benchmarks
CUDA-SDK [68]	GPU benchmarks

Table 1: Workloads used in this paper.

3 METADATA FIDELITY

To understand what granularity metadata is needed by applications, we measure fidelity loss for access and dirty metadata bits on a comprehensive range of workloads shown in Table 1, which include several modern huge page-sensitive applications for CPUs and GPUs, including Parsec [22], SPEC [90], GUPS, graph500 [41] and Redis [77] and GPU benchmarks from the MOSAIC GPU simulator [10]. The *fidelity loss* for a 2MB region over a time-period is the percentage of bits maintained at 4KB granularity incorrectly approximated by the single-bit maintained for 2MB pages.

We perform measurements on real hardware for CPUs, and in simulation for GPUs, capturing and comparing metadata maintained at different granularities. To enable a meaningful comparison, we create a deterministic runtime system which ensures that the address space layout for an application is identical whether the OS backs its address space with 2MB or 4KB pages. We periodically pause and capture memory metadata at points in the execution that are the same in both execution cases.[2] For GPUs, capturing metadata and establishing correspondence is considerably simpler, as the prototype runs in a simulator: a single trace of memory references and instruction counts is captured and post-processed to produce a set of epochs and snapshots of per-page metadata.

The runtime has two components: a user-space *region scanner* library that scans a program's virtual address space collecting metadata for analysis, and modifications to the Linux kernel to retrieve page-level metadata. The scanner produces a trace of page-level metadata that can be used for analysis across executions. We divide the program execution into epochs and execute a scan once per epoch. The scanner walks the process' virtual address space in 2 MB increments, making a system call to our modified Linux kernel to collect mapping sizes and page table entries.

Enforcing Determinism: For metadata traces across executions to be comparable, their corresponding epochs must represent the same period of program execution, and their corresponding memory segments must be mapped to the same regions of the virtual address space. To enforce epoch consistency, we compute the current epoch number based on the total retired instruction count provided by hardware performance counters. When there are multiple threads, we use the maximum retired instruction count of all threads and delay thread execution until the scanner has completed. To keep memory mappings consistent, we disable address space layout randomization (ASLR). Large allocation requests to `malloc()` that exceed the threshold to use `mmap()` (`M_MMAP_THRESHOLD`) can be mapped at different virtual addresses across executions. To avoid this, we set aside a large part of the virtual address space for these allocations at program initialization, maintain a bump pointer, and interpose all memory allocation requests. For those that exceed `M_MMAP_THRESHOLD`, we invoke `mmap()` with our bump pointer as the starting address.

Page-level Metadata: The scanners capture access, dirty, present, user/supervisor, and read/write bits from page table entries. For each 2MB extent, we maintain bit arrays to store the captured metadata bits. We find that fidelity loss for P/R/U (PRU) bits is generally low: usually, around 1-2%, and in the worst case up to 9.3% (`fluidanimate`). Moreover, the semantics for protection bits

[2]Our CPU runtime currently supports exclusively C/C++ programs and multithreaded programs using pthreads. Fortran programs from SPEC CPU and non-pthreads programs from Parsec are omitted.

CPU						
	epochs	2M-pgs	type	loss	stdev	entropy
blackscholes	6727	304	A	57	32	11
			D	14	30	11
canneal	890	289	A	26	26	17
			D	12	22	16
dedup	1113	685	A	34	38	10
			D	0	1	2
facesim	15745	127	A	75	35	16
			D	17	34	14
ferret	17912	43	A	42	43	16
			D	3	16	6
fluidanimate	16868	250	A	51	36	14
			D	48	36	14
raytrace	11332	909	A	88	17	17
			D	6	24	13
streamcluster	2335	53	A	2	13	11
			D	0	5	7
x264	4500	32	A	89	8	17
			D	15	35	12
astar-1	4052	135	A	31	30	15
			D	32	32	15
astar-2	4500	13	A	75	38	13
			D	74	39	13
bwaves-1	4500	412	A	43	46	13
			D	26	42	12
bzip2-1	4141	422	A	49	38	13
			D	29	36	12
bzip2-2	1786	45	A	44	39	11
			D	30	49	10
bzip2-3	2985	47	A	33	41	10
			D	27	41	10
bzip2-4	4500	422	A	42	38	12
			D	32	37	12
bzip2-5	4500	422	A	35	37	12
			D	32	34	12
bzip2-6	3321	302	A	48	39	12
			D	29	36	12
cactusADM-1	4500	327	A	69	24	13
			D	30	39	12

CPU						
	epochs	2M-pgs	type	loss	stdev	entropy
h264ref-1	4500	10	A	79	17	15
			D	79	33	12
h264ref-2	3324	8	A	82.8	14.0	14
			D	79	30	12
h264ref-3	4500	29	A	90	11.4	15
			D	50	48	13
hmmer-1	4500	11	A	61	23	12
			D	61	23	12
hmmer-2	4500	1	A	70	11	9
			D	71	11	9
lbm-1	4500	204	A	4	14	10
			D	2	11	10
libquantum-1	4500	31	A	3	12	8
			D	4	16	9
mcf-1	3120	836	A	77	25	21
			D	68	43	20
namd-1	4500	9	A	89	10	13
			D	72	41	12
omnetpp-1	4500	242	A	65	42	16
			D	21	30	17
perlbench-1	4500	44	A	83	27	16
			D	76	34	17
perlbench-2	3809	195	A	92	15	15
			D	76	36	15
povray-1	4500	1	A	84	5	12
			D	97	2	7
sjeng-1	4500	84	A	57	23	19
			D	69	27	19
soplex-1	3609	58	A	54	29	16
			D	27	40	15
soplex-2	3814	132	A	32	39	16
			D	22	39	15
graph500	2106	2046	A	3	15	10
			D	1	9	9
redis	2775	1337	A	88	12	21
			D	4	6	14
gups	2000	4096	A	20	4	20
			D	0.0	1	6

GPU						
	epochs	2M-pgs	type	loss	stdev	entropy
BLK	152	23	A	87.0	5.4	11.6
			D	32.7	42.0	10.2
BP	14	6	A	91.2	10.5	5.2
			D	66.9	41.2	4.9
CFD	5	6	A	79.3	20.3	3.9
			D	46.5	37.1	3.5
CONS	136	30	A	81.1	14.1	8.7
			D	31.6	38.6	7.5
FWT	690	20	A	84.2	14.1	10.8
			D	83.6	19.0	10.8
JPEG	3	4	A	89.7	17.6	2.3
			D	29.1	45.8	1.6
LPS	15	1	A	99.8	0.0	0.0
			D	0.0	0.0	0.0
NN	119	16	A	98.3	4.1	7.6
			D	27.3	44.4	4.8
QTC	14	6	A	65.1	46.3	3.8
			D	12.4	33.4	2.0
SC	287	40	A	90.4	9.6	12.6
			D	6.2	23.7	8.5
SPMV	27	8	A	69.1	17.4	7.1
			D	13.5	32.9	4.5
TRD	2	6	A	98.5	1.1	2.6
			D	33.0	51.1	1.6

Table 2: Aggregate characterization of all benchmarks. Metadata type is shown in type. Fidelity loss mean and standard deviation are in loss and stdev. The number of 100 million instruction intervals is in epochs, 2M-pgs is the number of 2MB pages. The entropy column is the theoretical minimum number of bits required for zero fidelity loss.

are more fundamentally coupled with pages, and changing their granularity would be more complex. In the remainder of the paper, we focus on access and dirty bits because unlike PRU bits, they are software hints, and we find that their granularity has a significant performance impact.

Measuring fidelity: We post-process traces captured by the scanners, establishing per epoch-region correspondence between traces collected using 4KB mappings and traces collected using 2MB mappings. Due to OS-level decision-making (Linux THP is heuristic), regions may or may not be backed by a 2MB huge page. Because we force deterministic epoch boundaries and virtual address space layout, accesses to a region during an epoch are identical regardless of physical backing. Thus, the difference between metadata bits for a huge page region vs. base page region during an epoch provides a measure of fidelity loss for maintaining metadata at a coarser granularity. The number of base page granularity metadata bits are wrongly approximated by the single bit for a huge page metadata bit divided by the total number of bits represents the fidelity loss for a single epoch-region.

3.1 Fidelity Loss Statistics

Table 2 characterizes the workloads' 2MB page usage and metadata fidelity loss. For CPUs, most workloads' address spaces are mapped by a combination of 2MB and 4KB pages because we rely on Linux THP. We do not present data for workloads whose memory footprint is so small that THP never allocates 2MB pages for them. GPU workloads use the Mosaic [10] GPU memory manager to transparently manage huge pages; the majority of mappings use 2MB pages.

For each workload, the table shows the number of epochs, the number of 2MB aligned regions backed by 2MB pages in the THP execution, along with the epoch count. An epoch is a period of 100 million retired instructions. [3] Because Linux THP promotes pages asynchronously as well as at allocation time, some fraction of regions may not be backed by 2MB pages in all epochs where they appear: the percentage of epoch-region pairs for which this occurs is low, so we do not report it.

[3]We empirically set this epoch length to 1 million GPU instructions for GPGPU workloads.

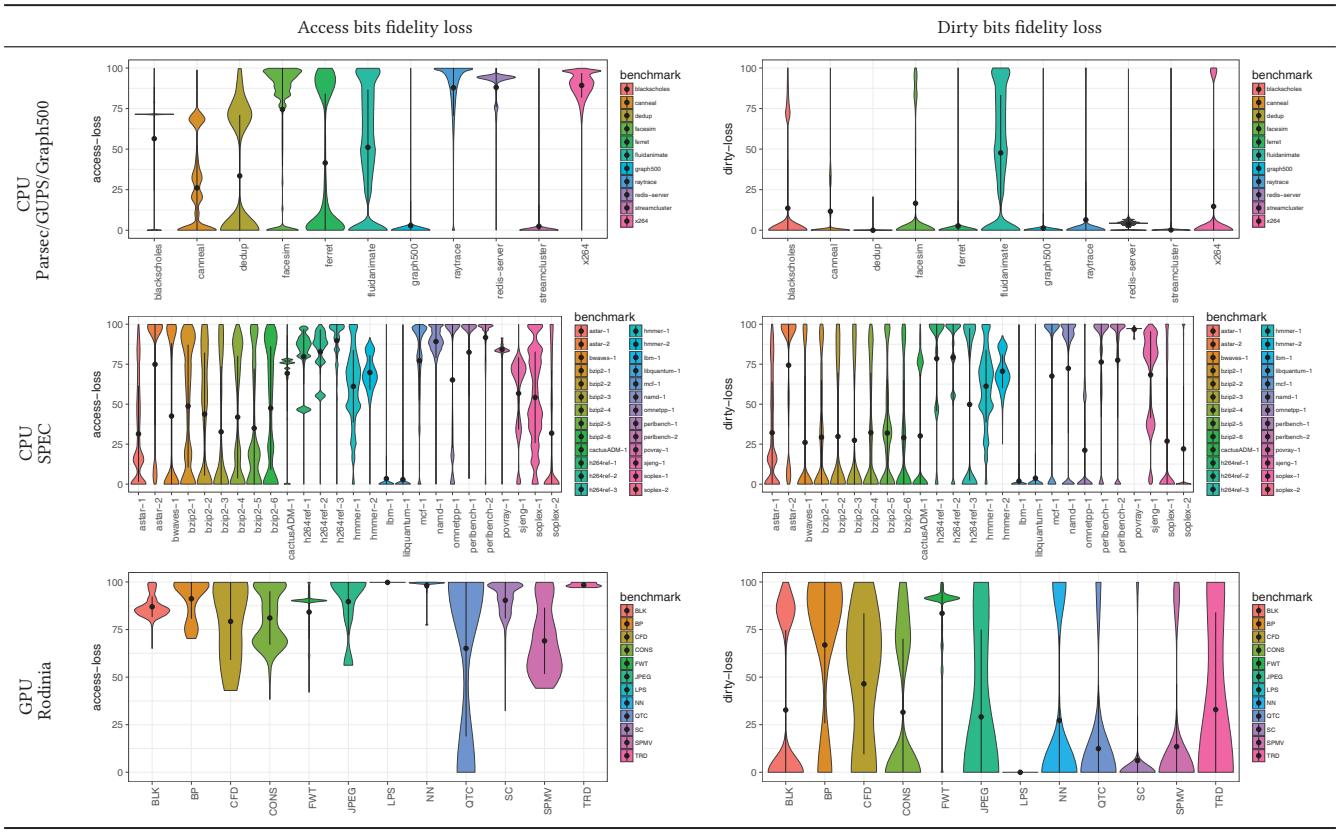

Figure 2: Fidelity loss for access, and dirty bits for all benchmarks, shown using violin plots, which represent the density of the distribution proportionally as the width of the bar at a given loss percentage. Dots represent the mean.

The average and standard deviation fidelity loss over all epoch-region pairs are in the **loss** and **stdev** columns, grouped by the metadata **type** column. The **entropy** column shows the theoretical minimum number of metadata bits that would be required to represent each metadata type with no fidelity loss for 4K page granularity, computed as the log base-2 of the number of unique bit-vectors observed over the application's runtime for the given metadata type.

The data show that fidelity loss varies by metadata type and is highly application-specific. Fidelity loss for dirty bits is generally lower than access bit loss, with a high variance within and across workloads. In some cases (e.g., `facesim`), dirty bit fidelity loss is negligible, but it can be higher than access bit fidelity loss (e.g., `sjeng-1` with 68.4%). Access bit fidelity loss is typically higher, but with similarly high variance. For example, `graph500` has only 2.8% loss over a multi-minute run time, while `perlbench-2` and `x264` have around 90% loss. The standard deviation of loss is usually quite large, showing that some loss varies per region and over time. The entropy measurements range from 2 to 21 bits ((`mcf-1` and `redis`). Dirty and access bit entropy is often different within a workload.

Fidelity Loss Distribution: Variance over time and per region is high, as shown by standard deviation losses in Table 2. Figure 2 shows the distribution for access and dirty bit loss for all workloads as violin [44] plots. The density at a given percentage is represented by the width of bars, dots show the mean. Distributions are multi-modal for many workloads. For example, `ferret`'s access bit loss is dominated by a large number of epoch-region pairs with very high (90% or greater) loss and a large number with very small loss (20% or lower). The variance reflects both spatial diversity (some regions have much higher loss) and temporal diversity (over time, regions tend to be accessed differently). Multi-modal distributions appear to be more common for access bits than for dirty bits.

Spatial and Temporal Fidelity: Metadata fidelity is sensitive to type, use case, and the time-frame for which it is maintained. For example, OS defragmentation (to increase contiguity) may examine access bits at intervals on the order of many seconds. Accumulation of access bits over that interval can inform a clear picture of what pages have not been accessed at all but obscures information about how recently individual pages have been touched.

We study the spatial and temporal fidelity of metadata bits for time scales from 200ms to 12.8 seconds. Figure 3 shows the spatial and temporal fidelity loss for access bits for all benchmarks. Dirty bits follow similar trends. The data reflect a fundamental tension between temporal and spatial loss. As the time scale increases, so does the period between the consecutive clearing of metadata bits. Consequently, set bits for 4k pages will tend to accumulate, making the single bit for a huge page a better approximation of their 512 bits over time. In the limit, as the time scale approaches infinity, assuming a uniform random distribution of accesses, a single bit becomes

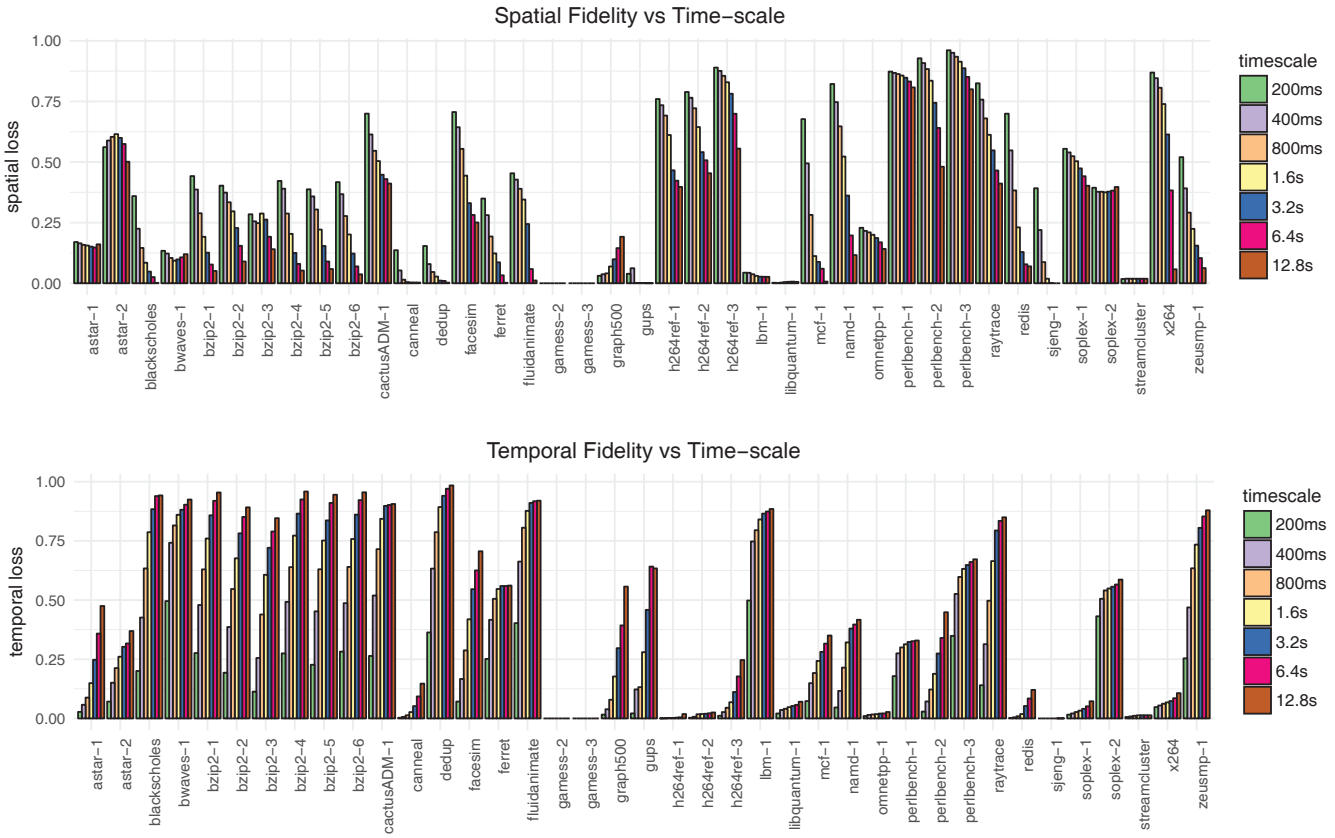

Figure 3: Average spatial and temporal access bit fidelity loss versus time scale for all benchmarks.

a perfect spatial approximation (except when pages go completely unreferenced). Conversely, as time scales increase, temporal fidelity for each set bit may decrease, as the single bit allocated for it in the base page mappings can approximate any number of accesses during the sampling period. Figure 3 shows that the average per-bit percentage of times the single bit over a longer time scale would have been incorrect at the lowest (100ms) time scale (e.g. because a page was only referenced during a subset of the 100ms epochs that comprise the larger time scale).

Adding hardware-supported metadata bits: To study the potential benefit of adding more hardware supported metadata bits, we post-process base page backed traces, OR'ing together metadata bits to produce what hardware would have maintained given a larger number of evenly distributed bits for that metadata type. We subsequently compare against the original base page backed traces to characterize fidelity loss, using a range of 2 to 256 additional bits per huge page.

Our data show that improving fidelity requires increased granularity. Fidelity loss varies dramatically, incentivizing per-huge page control of metadata granularity. Increased temporal fidelity is achievable in software with a low overhead as demonstrated in [59], so our design (§4) targets improved *spatial* fidelity for access and dirty bits with support for multiple granularities.

4 PRISM

In this section, we present a hardware-software co-design to gain additional visibility into memory access patterns at sub-page granularity while maintaining huge page mappings. We propose a few modest hardware changes to support finer-grain metadata bits for large pages. We introduce a structure called a *metadata registry* by extending the page table, extend the TLBs to support extra metadata bits, and enhance the hardware page table walker to load and update the metadata registry. We will discuss each of these changes in detail for the rest of the section.

4.1 Metadata Registry

In today's processors, hardware-managed metadata bits are stored as part of page table entries. The viability of simply adding more bits by extending page table formats is limited at best. Thus, we propose a hardware design that can allocate additional per-address space memory as part of the page table structure dedicated to metadata storage. To simplify the discussion, we label this area the Metadata-registry.

In a 4-level page table, we add a physically contiguous metadata registry page right next to the level 2 page table pages, which consists of 512 64-bit metadata registry entries (MREs) shown in Figure 4. We take advantage of 9 unused bits in the level 2 entry format for 2MB pages to encode the presence or absence of a valid

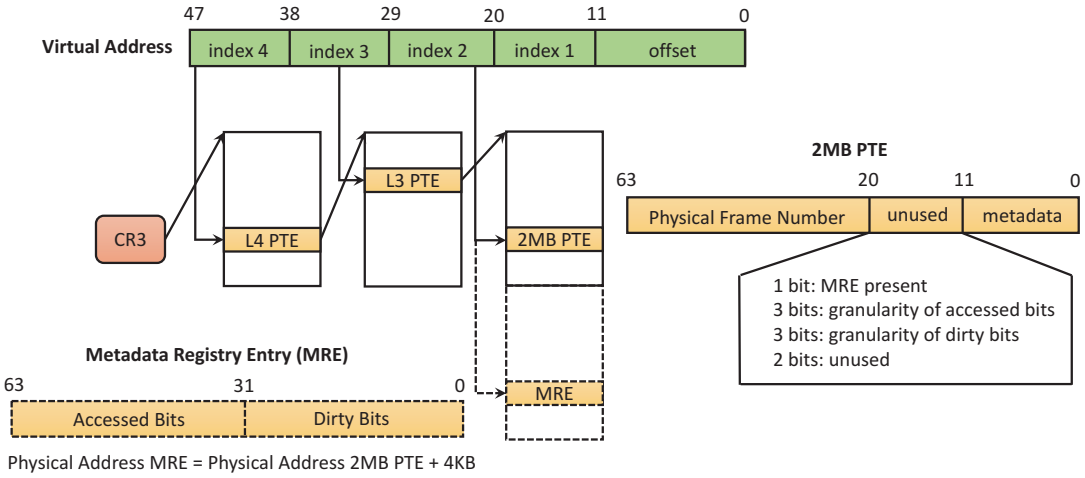

Figure 4: Structure of the Metadata-registry, which extends level 2 page table pages with an additional physically contiguous page containing 64-bits of additional metadata storage per entry in the level 2 page. We appropriate one of the 9 unused bits in the x86-64 format for 2MB page entries as flag which indicates the presence or absence of a valid metadata registry entry (MRE) at the same offset in the metadata page. The remaining 8 unused bits are appropriated to encode the number of access and dirty bits the hardware should manage for the corresponding 2MB page.

MRE for the page, as well as the metadata granularity. With one bit used for the flag, 8 bits remain to encode valid combinations of metadata granularities: 3 bits for access metadata granularity, 3 for dirty metadata, and 2 bits remain unused.

Table 3 shows encodings for metadata granularities, used in the level 2 2MB page descriptors to tell the hardware how to interpret metadata registry entries. Access and dirty bits can be set to power-of-2 granularities ranging from 1 to 32. The entropy measurement in the previous section showed that every workload ideally requires different granularities of bits for access and dirty bits. See data in Section 6 for the sensitivity studies used to select these granularities.

encoding	granularity
000	1 bit
001	2 bits
010	4 bits
011	8 bits
100	16 bits
101	32 bits
11X	Not allowed

Table 3: Metadata granularity encodings.

TLB Support: For applications to make use of the metadata registry, TLB and hardware page table walkers need to support caching and loading the corresponding metadata registry entry on a TLB miss. We extend the 2MB TLB hardware to support storing the 64-bit MRE and the MRE present bit along with the 2MB standard PTE. We store a fully decompressed MRE (granularity 32) with the 2MB PTE in the L1 and L2 TLBs, so we do not have to store the size encoding in the TLB and decompress the MRE during latency-critical TLB lookup.

Modern Intel x86-64 address translation hardware supports a multi-level TLB hierarchy (L1 and L2 TLBs) along with page walk caches [18] to reduce the cost of 4-level page walk on a TLB miss. L1 TLBs are split according to page sizes (separate TLBs for 4KB and 2MB) and more recent Broadwell architecture supports a combined L2 TLB for caching 4KB and 2MB PTEs. We extend all structures that can store the 2MB PTE (See Figure 5). The area and energy analysis of extending the TLB resources is performed in §6.

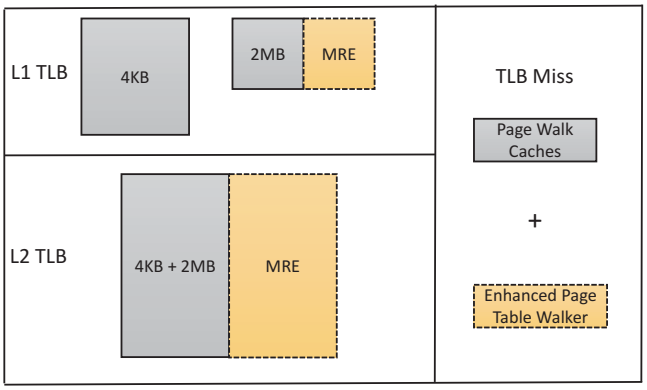

Figure 5: Hardware TLB to support finer-grain metadata. Boxes in gray show the original TLB hierarchy. Dashed boxes in yellow show our moderate additions to cache metadata registry entries (MREs).

On a TLB lookup, if a 2MB PTE hits, the MRE is read along with the entry since the MRE is stored along with the PTE. The access and dirty bits are looked up in the MRE. If the access and dirty bits are appropriately set, the TLB hit proceeds normally. But, if the access or dirty bit is not set in the MRE, the correct bit is set in

the MRE in the TLB for the application to proceed ahead and page table walk is issued in the background to write the correct bit to the MRE in the metadata registry. Hardware-managed metadata bits are updated using the same technique when access or dirty bit is not set in the PTEs today. We extend that functionality in the page table walker to support updating the metadata registry.

In addition, the page walker needs to be enhanced to fetch the MRE from the metadata table on a 2MB TLB miss. In current hardware, the page table walker will use the page walk caches to skip some levels of the page table and access memory (cached or in DRAM) to fetch the PTE corresponding to a TLB miss and introduces it in the TLB. With our design, if the page table walker gets a 2MB PTE, it analyzes if the MRE present bit is set in the 2MB PTE. If the MRE present bit is not set, it inserts the 2MB standard PTE into the TLB. In case, the MRE present bit is set, then it knows the physical address of the MRE since the physical address of the MRE is just 4KB away from the PTE. The page table walker fetches the MRE and then decompresses the MRE based on the size encoding in the 2MB PTE and contents of the MRE. The MRE is easy to fetch since it is physically contiguous to the level 2 page of the page table, but cost another memory access (which maybe cached in data caches) on a page table walk. Our experiments show that the added cost of extra memory access on a 2MB TLB miss is negligible in comparison to the benefits of 2MB page size.

Extension to 1GB page size: Figure 6 shows how Prism can be extended to enable finer-granularity metadata for 1GB page size. In this case, Prism can use the level 3 page table level to store the additional pointers to required metadata bits. This design has the same contiguity requirement as the 2MB design and requires one contiguous 4KB page for the MRE entry in the L3 page table level, and this design contains 512 MRE pointers inside the L3 page table. However, while storing the MRE in level 3 page table is straightforward, scalable caching of those additional metadata bits in the TLB would require a highly compressed TLB line format, which is a challenge we will explore in future work.

Overall, Prism gives system software detailed information about access and dirty bits to take better decisions on various tasks that it wants to perform like swapping a smaller granularity page. The system software can use the information stored in the MRE just like it uses the access and dirty bits today.

Note that all the hardware modifications are optionally enabled by the system software. For these changes to be used, the system software needs to set the 2MB PTE correctly with the granularity to be used for a huge page along with its present bit (See §4.3). The only cost to be paid is the area overhead.

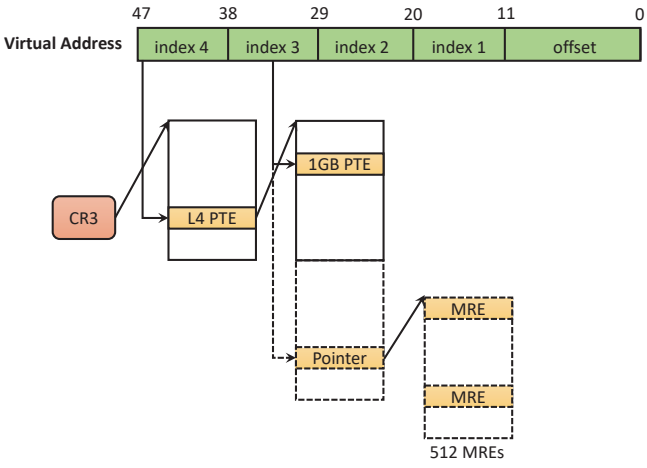

Figure 6: Prism extension for 1GB page size. The encoding of bits is the same in 1GB PTE with unused bits.

4.2 Support for Arbitrary Granularity

We considered designs capable of supporting the full range of granularities between 1 and 512 bits for each metadata type, and even user-defined metadata types. However, while our experiments show that there are applications capable of benefiting from very fine granularity, they are not the common case. Moreover, the introduction of additional metadata entries impacts page table walk design and latency, as well as raising the question of whether additional metadata can be cached in extended TLB entries versus requiring a separate hierarchy to avoid unreasonable impact on TLB design.

Arbitrary granularity metadata can be supported for example by reserving a bit of MREs that indicates whether the MRE contains metadata, or a pointer to a larger structure capable of encoding more bits. Similarly, very large numbers of bits raise some difficult questions about how to allow the hardware to update metadata bits in response to loads and stores. In modern architectures, metadata bits are written in both the TLB and main memory, enabling the TLB to effectively filter redundant writes to the page table to set metadata bits that are already set. Supporting large metadata bitcounts could either force the addition of a caching structure to perform such filtering or saturating the memory subsystem with redundant metadata updates that arise because no mechanism exists to filter them.

Collectively, these considerations constitute strong incentive to limit additional metadata bits to 64: it simplifies page table walk, makes it practical to deal with metadata read/write by extending TLB entries (we evaluate the impact of a range of design points in Section 6, in particular, Figure 8), and still enables most applications to reduce metadata fidelity loss for huge pages to very low levels.

4.3 Prism Software Interface

Prism supports application-level, or OS-level control over metadata granularity through extensions to the system call interface similar to mprotect called `mmdconfig()`, as follows:

```
int mmdconfig(void *addr, size_t len,
              int mdtype, int bits);
```

Processes can set metadata granularity for *all* 2MB pages in their address space by passing NULL for the `addr` parameter, or for all/any parts of the region described by `addr` and `len` backed by 2MB pages.

5 METHODOLOGY

Performance Model: To measure the overheads of Prism's metadata registry and hardware modifications, we combine performance counter measurements from native executions with a linear performance model (see Table 4). Compared to cycle-accurate simulation

on these workloads, this approach reduces weeks of simulation time by orders of magnitude. Previous virtual memory system performance studies use this approach [15, 18, 33, 34, 52].

Using hardware performance counters, we collect the following metrics for each workload: total execution cycles (C), number of 2MB TLB misses (M_{2M}), and total page table walk cycles (C_{PTW}). We also collect the number of page table walker loads by the level of the memory hierarchy: L1 (PW_{L1}), L2 (PW_{L2}), L3 (PW_{L3}), and DRAM (PW_{mem}). We weigh the cost of accessing L1 cache, L2 cache, L3 cache, and memory for each workload by 3 cycles, 11 cycles, 40 cycles, and 200 cycles (their access latency) respectively, which is an approximation of recent Intel CPUs [45, 46].

On a 2MB TLB miss with metadata, the hardware must retrieve both the PTE and the corresponding MRE. Given this access pattern, we expect a PTE and its MRE to be typically collocated at the same level of the memory hierarchy. We, therefore, base the MRE access cycles (MRE_{access}) on the observed memory characteristics of PTE accesses during a page table walk (PW_{L1} through PW_{mem}). Since the physical address of the MRE is a constant offset from the PTE, the MRE load requires just single memory access.

MRE updates are performed when access and dirty bits need to be set in response to a retiring load or store. These updates in our performance model are performed synchronously with the load or store. We also model performance if these could be de-prioritized and scheduled in the background. To estimate the maximum number of possible MRE updates, our model assumes all MRE bits are cleared at the start of each epoch (E) and that during the epoch all MRE_{len} bits (64) for every 2MB page in the application (P_{2MB}) will need to be updated. Unlike the MRE load, the updates will require a page walk and here we pessimistically model 2 memory loads to retrieve the MRE.

MRE_{access}	$\frac{PW_{L1}*3+PW_{L2}*11+PW_{L3}*40+PW_{mem}*200}{PW_{L1}+PW_{L2}+PW_{L3}+PW_{mem}}$
$C_{MREload}$	M_{2MB} * MRE_{access}
$C_{MREupdate}$	E * P_{2MB} * MRE_{access} * 2 * MRE_{len}
O(MRE load)	$\frac{C_{MREload}}{C}$
O(+MRE update)	$\frac{C_{MREload}+C_{MREupdate}}{C}$

Table 4: Performance model for emulating hardware overheads.

GPU Evaluation: We modify Mosaic [10, 12], which is an open–source variant of GPGPU-Sim [13] that models the behavior of NVIDIA Unified Virtual Address Space [69] support. Table 5 shows the GPU hardware configuration. Mosaic adds a memory allocator into cuda-sim, the CUDA simulator within GPGPU-Sim, to handle all virtual-to-physical address translations and to provide memory protection, an accurate model of address translation to GPGPU-Sim, including TLBs, page tables, and a page table walker. The page table walker is shared across all SMs and allows up to 64 concurrent walks. Both the L1 and L2 TLBs have separate entries for base pages and huge pages [33, 52, 53, 72, 75, 76]. Each TLB contains 64 miss status holding registers (MSHRs) [58] to track in-flight page table walks. The simulation infrastructure is modified to support demand paging, by detecting page faults, faithfully modeling the system I/O bus (i.e., PCIe) latency based on measurements from NVIDIA GTX 1080 cards [70]. We utilize the state-of-the-art unified memory design with prefetching and the capability to pro-actively evict unused pages out of the GPU memory [60].

GPU Core Configuration	
Shader Core Config	30 cores, 1020 MHz, GTO warp scheduler [80]
Private L1 Cache	16KB, 4-way associative, LRU, L1 misses are coalesced before accessing L2, 1-cycle latency
Private L1 TLB	128 base page/16 huge page entries per core, fully associative, LRU, single port, 1-cycle latency

Memory Partition Configuration (6 memory partitions in total, with each partition accessible by all 30 cores)	
Shared L2 Cache	2MB total, 16-way associative, LRU, 2 cache banks and 2 ports per memory partition, 10-cycle latency
Shared L2 TLB	512 base page/256 huge page entries, non-inclusive, 16-way/fully-associative (base page/huge page), LRU, 2 ports, 10-cycle latency
DRAM	GDDR5, 1674 MHz, 6 channels, 8 banks per rank, FR-FCFS scheduler [79, 100], burst length 8 capacity is set to 90% of the working set size LRU page replacement policy

Table 5: Configuration of the simulated system.

We modify the page replacement policy to use Prism's metadata granularity. When the GPU needs to evict a page, the GPU searches pages according to the configured metadata granularity including the huge page baseline with 1 metadata bit per huge page. When multiple pages have the same access metadata, one is randomly chosen from among them as a target for eviction. We leverage both 8-bit and 32-bit metadata registry to improve GPU's page replacement policy. We execute 15 randomly selected GPGPU applications from various benchmark suites [26, 68, 91]. To emulate the oversubscribed systems, we configure the GPU memory to 90% of its application's memory footprint.

Memory Allocator: For CPU evaluation, we used the default glibc allocator derived from ptmalloc. For GPU evaluation, we use state-of-the-art memory allocator from previous works [10–12].

6 EVALUATION

This section reports sensitivity studies of temporal and spatial fidelity as a function of time scale and number of metadata bits. Based on these data, we report derived performance for a system extended with Prism, along with estimates of additional area, power, latency, and DRAM overheads.

6.1 Recovering Fidelity

Figure 7 shows the potential benefit of adding more hardware-managed access bits, showing spatial access bit fidelity loss as a function of the number of bits per 2MB extent, ranging from 2 bits to 256-bits (a single bit is identical to THP, while 512 is identical to using 4KB mappings). The data show that applications benefit differently from more bits. For many applications, fidelity loss is significantly reduced with 8-16 bits (e.g. `bzip-*`), while others require a large number of additional bits to realize an improvement.

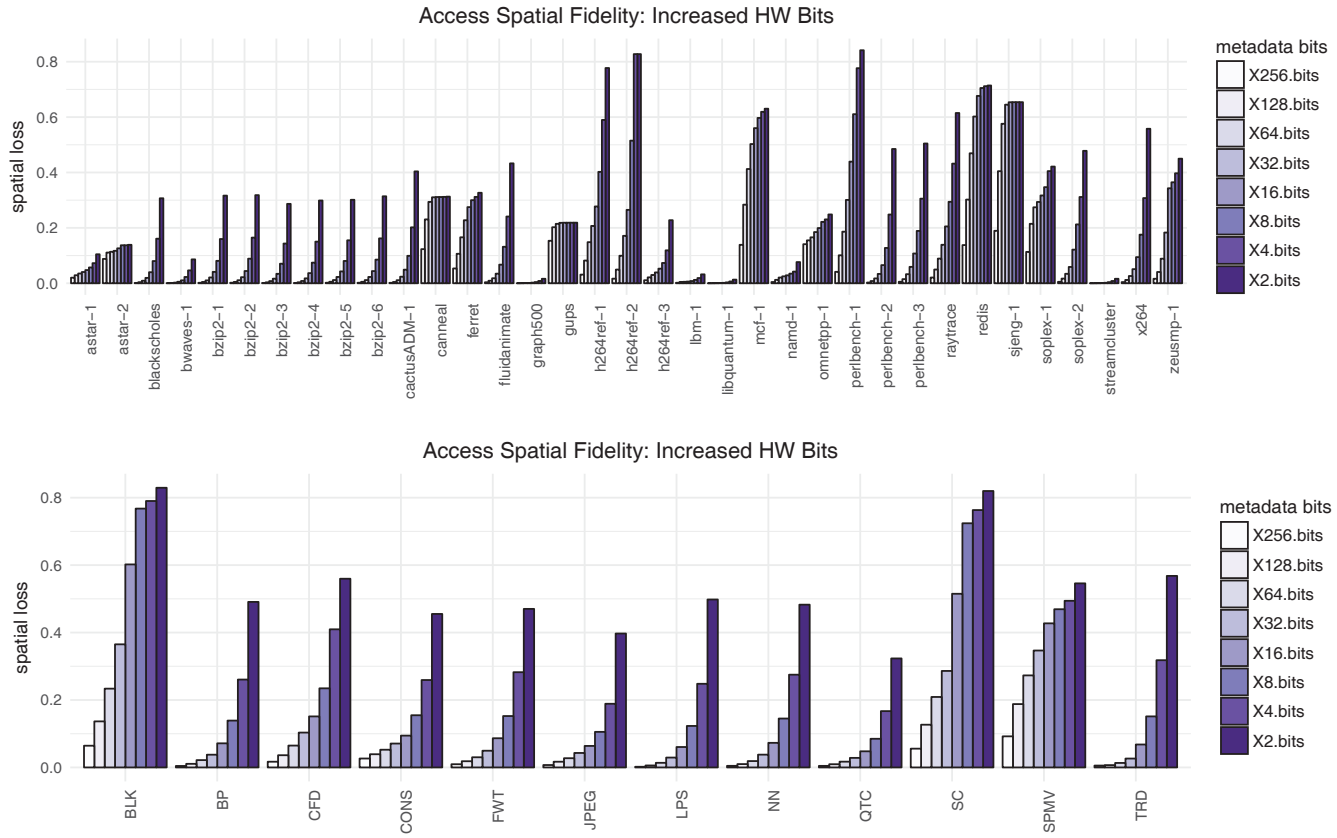

Figure 7: Average access bit fidelity loss using increasing number of hardware-managed access bits

For example, `redis`, `canneal`, and `sjeng-1`, the loss does not drop below 20% until 256 bits. Others, such as `blackscholes` require little additional fidelity and are dramatically improved by a handful of additional bits.

Selecting the range of metadata granularities to support in hardware must balance fidelity gains against the impact in complexity, area, energy, and memory overheads. Limiting total additional metadata bits to 64 per huge page is desirable because it provides some per-page flexibility while preserving a simple page table walker and simple offset-based lookup in the metadata registry. Figure 8 shows the harmonic mean loss across all benchmarks as a function of the number of metadata bits. While further fidelity improvements are obtained at granularities greater than 64, *the fidelity loss* for all metadata types is reduced to under 15% at 32-bits. We conclude that the hardware complexity for registry entries with greater than 64 bits is not justified by the potential benefit, and consider only organizations that use 64-bits across access and dirty metadata types.

Using fewer than 64 bits is desirable: extending TLB entries to include additional metadata is less complex than introducing separate metadata caching hardware which must be kept coherent with TLB entries and would require additional content-addressable memories. Based on Figure 8, using 32-bits each for access and dirty bits consumes 64 bits and reduces loss to 13% and 12%, respectively. The combination of 16 each for access and dirty, requires 32 additional bits per TLB entry and reduces loss to about 20% for access and dirty metadata. At the frugal end of the spectrum, we evaluate augmenting the TLB with 16 access bits, and a single dirty bit.

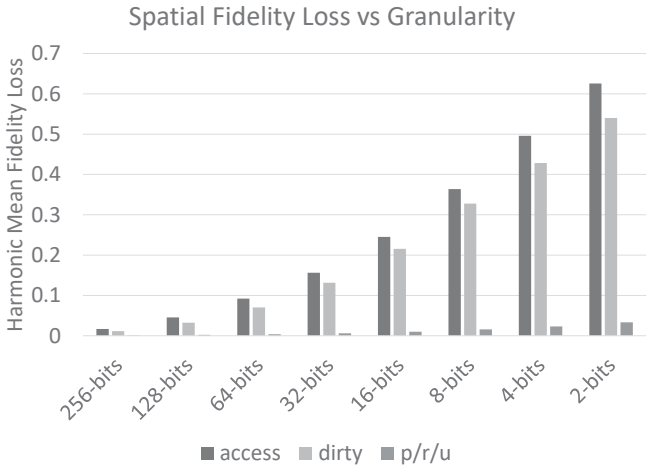

Figure 8: Harmonic mean spatial fidelity loss.

	blackscholes	fluidanimate	canneal	dedup	raytrace	facesim
MRE loads	0.00% (0.53%)	0.00% (8.20%)	0.05% (0.92%)	0.00% (0.30%)	0.00% (0.13%)	0.00% (0.19%)
+ MRE updates	0.04% (321.77%)	0.10% (2420.36%)	0.14% (2.64%)	0.10% (10.37%)	0.09% (118.69%)	0.02% (21.27%)
	streamcluster	**x264**	**ferret**	**gups**	**redis**	**graph500**
MRE loads	0.00% (4.77%)	0.00% (0.19%)	0.00% (0.09%)	12.58% (24.03%)	0.03% (14.32%)	0.22% (42.00%)
+ MRE updates	0.01% (582.26%)	0.00% (4.76%)	0.00% (0.90%)	13.14% (25.09%)	0.66% (316.00%)	0.47% (90.56%)

Table 6: Performance impact of the metadata registry, reported in: % increase in total cycles (% increase in page walker cycles).

6.2 Area, Power, Latency Impact

To evaluate the impact on TLB geometry and resource consumption, we use CACTI [66] models of L1 2MB TLBs and L2 unified TLBs based on the TLB parameters for Intel Skylake processors–32 entries (4-way associative) and 1536 entries (12-way associative) respectively. Because CACTI supports only power-of-2 associativity, we round up to 16 for the L2 TLB model. Figure 9 shows access latency, dynamic read energy, and total area for baseline Skylake-like L1 and L2 TLBs as well as three designs that extend each entry with additional metadata bits. The 32/32, 16/16, and 16/1 indicate the number of access/dirty bits. The worst-case area growth and access latency growth is around 5%, while the worst-case access energy increases by about 9%.

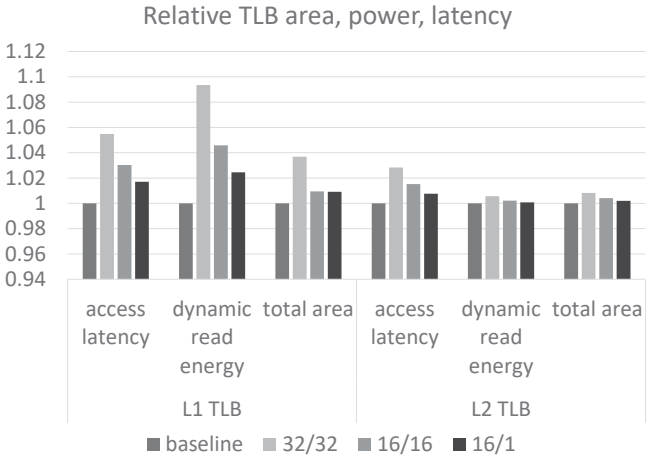

Figure 9: Relative L1 and L2 TLB access latency, read energy, and area for various extended metadata capacities.

6.3 Memory Impact

To characterize memory overhead, we consider the total number of 2M pages mapped by each application from Table 2. Metadata registry entries are stored in the 4KB page immediately following the level 3 page table page.

To compute a conservative lower bound on page table expansion, we assume that all level 3 page table pages require subsequent metadata registry pages. The average bloat over all benchmarks using this estimate is about 1.2MB, while the maximum is 8.38MB.

6.4 CPU Performance

To estimate the performance impact of the Prism, we observe that additional overheads will arise from two operations: first, MRE loads to construct extended TLB entries on a TLB miss, and second MRE updates to set metadata bits. The first cause minor performance loss since it is in the critical path of loading a TLB entry on a TLB miss. The second does also impact performance slightly since most of the metadata updates happen synchronously in our model. We can reduce the impact of MRE updates by performing them in the background asynchronously.

We pessimistically model both components of overheads and include them as performance overheads (see §5 for more details). To model MRE updates, we simulate an operating system (or hypervisor) clearing the metadata bits once per second. Table 6 shows slowdown across most benchmarks broken down by MRE loads/updates. As we can see, except for GUPS, even a pessimistic model of metadata registry performance results in an overhead of less than 1%. GUPS spends more than 60% of its execution time in TLB misses, and an increase in the TLB miss latency impacts it. We exclude SPEC workloads in this table to conserve space, and they do not show any specific new insights.

We conclude that with modest additions to the hardware, metadata fidelity loss can be reduced by about 50% on average relative to page-granularity metadata bits, with a near-negligible impact on performance.

6.5 Improving GPU Page Replacement

We demonstrate the performance benefit of Prism on GPGPU applications in over-subscribed memory conditions. Figure 10 shows the normalized performance of Prism on 15 randomly selected GPGPU applications compared to two baselines: baseline that utilizes all 512 metadata bits per huge page, which is equivalent to using 4KB mappings, and the other baseline that utilize 1 metadata bit per huge page, which is identical to THP. Base on this result, we provide four observations. First, the Prism-32bit configuration provides 29.8% performance improvement on average over the 1-bit metadata baseline while the Prism 8-bit configuration provides a 7.9% performance improvement over the 1-bit metadata baseline. Second, Prism 8-bit only performs 1.8% worse, on average, compared to the ideal 512-bit metadata baseline. Third, we observe that while using 8 bits per huge page for metadata is sufficient in all but one benchmark, the fidelity lost in SC leads to 75.4% performance lost when only 8 bits of metadata is available, and 80.6% performance lost when only 1-bit metadata is available. In contrast, Prism mitigates this performance loss by allowing the GPU to adapt to various metadata granularity based on each application's

tolerance toward fidelity loss. Fourth, we found that previously proposed techniques to mitigate memory over-subscription [60, 99] are not sufficient to mitigate the overhead of page eviction when a huge page is used on multiple workloads (FWT, QTC, and SC) due to the fidelity loss. In these cases, we found that the eviction policy incorrectly evicts pages that are likely to be used again, leading to severe performance degradation. In contrast, Prism allows the page eviction policy to identify the correct pages to be evicted and minimizes the performance overhead of demand paging in GPUs.

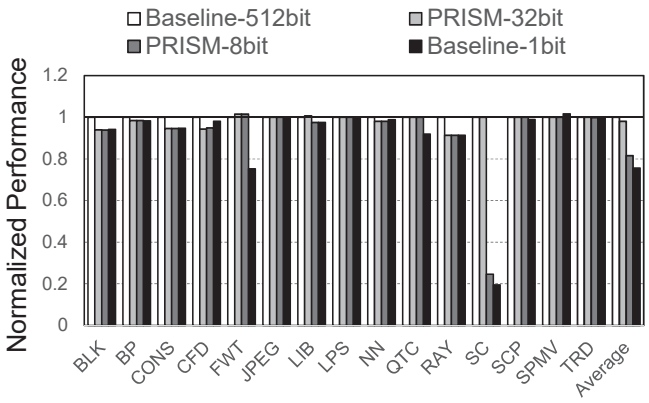

Figure 10: GPU performance on a system with Prism normalized to a baseline with 512-bit metadata granularity.

7 RELATED WORK

Performance degradation from address translation overheads is well-established [15–17, 20, 21, 34, 37, 52–54, 61, 63], and is exacerbated by hypervisor-based virtualization [4, 7, 17, 24, 34–36, 96]. Our work is related through the shared goal of reducing the translation overheads and by our own goal of reducing fidelity to loss to better inform OS policy.

Larger page sizes have been used widely to reduce translation overheads [31, 32, 43, 82, 92, 93]. Most modern processors like x86-64, ARM, MIPS, ALPHA, PowerPC, and UltraSPARC support multiple page sizes [49]. Navarro et al. [67] implement OS support for multiple page sizes, and the ideas are widely used [27]. Gorman et al. [39] propose a placement policy for an OS's physical page allocator that mitigates fragmentation and promotes contiguity. Subsequent work [40] proposes a software-exposed interface for applications to explicitly request huge pages like `libhugetlbfs` [38].

More recently, there have been proposals for compact mappings [74, 75, 93] that map multiple pages with a single TLB entry, improving TLB reach by a small factor (e.g., to 8 or 16); but much greater improvements to TLB reach are needed to deal with modern memory sizes. Direct segments [15, 34] extend standard paging with a large segment to map the majority of address space to a contiguous physical memory region but require application modifications and are limited to workloads able to a single large segment. Redundant memory mappings [7, 37, 52] extend TLB reach by mapping *ranges* of virtually and physically contiguous pages in a range TLB.

Additionally, TLB miss overheads can be reduced by accelerating page table walks [12, 14, 18] or reducing their frequency [33]; by reducing the number of TLB misses (e.g. through prefetching [19, 21, 50, 81], prediction [7, 72], or structural change to the TLB [74, 75, 93], TLB hierarchy [4, 5, 15, 20, 34, 52, 61, 89]) or the page table structure [2, 84, 85].

Many have observed problems with OS support for large pages [8, 32, 43, 59, 67, 71, 93, 97]. Features in modern OSes such as memory compaction, same-page merging [55], swapping, working set prediction, and hotness detection [3] pose new challenges. Better hardware support for larger page sizes or compact mappings ultimately reduces translation overheads, which further incentivizes their use. This, in turn, further exacerbates the need for finer-grain metadata. Starting at Ice Lake, Intel added support for sub-page write protection of Extended Page Table (EPT) pages. Permissions are enforced on 128-byte boundaries and are controlled by a separate page table and permissions vector. Though, this mechanism does not currently support access or dirty bits [47].

Hardware support for prefetching can rely on spatial locality and are related through architectural support for finer tracking of spatial accesses. SMS [86] tracks spatial memory regions at 8KB boundaries, but its prefetch-specific design relies on many tables to track patterns and their history.

8 CONCLUSION

Metadata fidelity in the presence of multiple page sizes varies significantly both spatially and temporally, across differing metadata types, and both across and within individual workloads. Access and dirty metadata have a high variance that is difficult to compensate efficiently with uniform bit increases or software-based strategies. The fidelity inherent in small page sizes for access and dirty metadata can be recovered with modest additional hardware, providing system software with much-needed visibility to improve performance.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers from PACT 2020. We acknowledge the support of our industrial partners, especially VMware and ARM. This research is partially supported by the NSF (grants CNS-1618563, CNS-1846169, and CNS-2006943).

REFERENCES

[1] Adam Abrevaya. Linux Transparent Huge Pages, JEMalloc and NuoDB. http://www.nuodb.com/techblog/linux-transparent-huge-pages-jemalloc-and-nuodb. (2014).

[2] Reto Achermann, Ashish Panwar, Abhishek Bhattacharjee, Timothy Roscoe, and Jayneel Gandhi. 2020. Mitosis: Transparently Self-Replicating Page-Tables for Large-Memory Machines. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[3] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-transparent Page Management for Two-tiered Main Memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[4] Jeongseob Ahn, Seongwook Jin, and Jaehyuk Huh. 2012. Revisiting Hardware-Assisted Page Walks for Virtualized Systems. In *International Symposium on Computer Architecture (ISCA)*.

[5] Jeongseob Ahn, Seongwook Jin, and Jaehyuk Huh. 2015. Fast Two-Level Address Translation for Virtualized Systems. In *IEEE Transactions on Computers*.

[6] Hanna Alam, Tianhao Zhang, Mattan Erez, and Yoav Etsion. 2017. Do-It-Yourself Virtual Memory Translation. In *International Symposium on Computer Architecture (ISCA)*.

[7] Chloe Alverti, Stratos Psomadakis, Vasileios Karakostas, Jayneel Gandhi, Konstantinos Nikas, Georgios Goumas, and Nectarios Koziris. 2020. Enhancing and Exploiting Contiguity for Fast Memory Virtualization. In *International Symposium on Computer Architecture (ISCA)*.

[8] Nadav Amit. 2017. Optimizing the TLB Shootdown Algorithm with Page Access Tracking. In *USENIX Annual Technical Conference (USENIX ATC)*.

[9] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. 2017. MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability. In *International Symposium on Computer Architecture (ISCA)*.

[10] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J. Rossbach, and Onur Mutlu. 2017. Mosaic: A GPU Memory Manager with Application-transparent Support for Multiple Page Sizes. In *International Symposium on Microarchitecture (MICRO)*.

[11] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J. Rossbach, and Onur Mutlu. Mosaic: Enabling Application-Transparent Support for Multiple Page Sizes in Throughput Processors. 52, 1 (2018).

[12] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J. Rossbach, and Onur Mutlu. 2018. MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[13] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[14] Thomas W. Barr, Alan L. Cox, and Scott Rixner. 2010. Translation Caching: Skip, Don'T Walk (the Page Table). In *International Symposium on Computer Architecture (ISCA)*.

[15] Arkapravu Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. 2013. Efficient Virtual Memory for Big Memory Servers. In *International Symposium on Computer Architecture (ISCA)*.

[16] Arkaprava Basu, Mark D. Hill, and Michael M. Swift. 2012. Reducing Memory Reference Energy with Opportunistic Virtual Caching. In *International Symposium on Computer Architecture (ISCA)*.

[17] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating Two-dimensional Page Walks for Virtualized Systems. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[18] Abhishek Bhattacharjee. 2013. Large-reach Memory Management Unit Caches. In *International Symposium on Microarchitecture (MICRO)*.

[19] Abhishek Bhattacharjee. 2017. Translation-Triggered Prefetching. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[20] Abhishek Bhattacharjee, Daniel Lustig, and Margaret Martonosi. 2011. Shared Last-level TLBs for Chip Multiprocessors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[21] Abhishek Bhattacharjee and Margaret Martonosi. 2009. Characterizing the TLB Behavior of Emerging Parallel Workloads on Chip Multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[22] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University.

[23] William J. Bolosky, Robert P. Fitzgerald, and Michael L. Scott. 1989. Simple but Effective Techniques for NUMA Memory Management. In *ACM Symposium on Operating System Principles (SOSP)*.

[24] Jeffrey Buell, Daniel Hecht, Jin Heo, Kalyan Saladi, and Reza H. Taheri. Methodology for performance analysis of VMware vSphere under Tier-1 applications. *VMware Technical Journal* 2, 1 (2013).

[25] Irina Calciu, Ivan Puddu, Aasheesh Kolli, Andreas Nowatzyk, Jayneel Gandhi, Onur Mutlu, and Pratap Subrahmanyam. 2019. Project PBerry: FPGA Acceleration for Remote Memory. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*.

[26] Shuai Che, M. Boyer, Jiayuan Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, and K. Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *International Symposium on Workload Characterization (IISWC)*.

[27] Jonathan Corbet. 2009. Transparent Hugepages. https://lwn.net/Articles/359158/.

[28] Couchbase, Inc. 2014. Often Overlooked Linux OS Tweaks. http://blog.couchbase.com/often-overlooked-linux-os-tweaks.

[29] Yigit Demir, Yan Pan, Seukwoo Song, Nikos Hardavellas, John Kim, and Gokhan Memik. 2014. Galaxy: A High-performance Energy-efficient Multi-chip Architecture Using Photonic Interconnects. In *Proceedings of the 28th ACM International Conference on Supercomputing (ICS)*.

[30] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data Tiering in Heterogeneous Memory Systems. In *ACM European Conference in Computer Systems (EuroSys)*.

[31] Zhen Fang, Lixin Zhang, John B. Carter, Wilson C. Hsieh, and Sally A. McKee. 2001. Reevaluating Online Superpage Promotion with Hardware Support. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[32] Narayanan Ganapathy and Curt Schimmel. 1998. General Purpose Operating System Support for Multiple Page Sizes. In *USENIX Annual Technical Conference (USENIX ATC)*.

[33] Jayneel Gandhi, , Mark D. Hill, and Michael M. Swift. 2016. Exceeding the Best of Nested and Shadow Paging. In *International Symposium on Computer Architecture (ISCA)*.

[34] Jayneel Gandhi, Arkaprava Basu, Mark D. Hill, and Michael M. Swift. 2014. Efficient Memory Virtualization. In *International Symposium on Microarchitecture (MICRO)*.

[35] Jayneel Gandhi, Mark D. Hill, and Michael M. Swift. 2016. Agile Paging: Exceeding the Best of Nested and Shadow Paging. In *International Symposium on Computer Architecture (ISCA)*.

[36] Jayneel Gandhi, Mark D. Hill, and Michael M. Swift. Agile Paging for Efficient Memory Virtualization. *IEEE Micro* 37, 3 (2017), 80–86.

[37] Jayneel Gandhi, Vasileios Karakostas, Furkan Ayar, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift, and Osman Ünsal. Range Translations for Fast Virtual Memory. *IEEE Micro* (May 2016).

[38] Mel Gorman. 2010. Huge Pages Part 2 (Interfaces). https://lwn.net/Articles/375096/.

[39] Mel Gorman and Patrick Healy. 2008. Supporting Superpage Allocation Without Additional Hardware Support. In *International Symposium on Memory Management (ISMM)*.

[40] Mel Gorman and Patrick Healy. 2010. Performance Characteristics of Explicit Superpage Support. In *Workshorp on the Interaction between Operating Systems and Computer Architecture (WIOSCA)*.

[41] graph500 [n.d.]. http://www.graph500.org/.

[42] Fei Guo, Seongbeom Kim, Yury Baskakov, and Ishan Banerjee. 2015. Proactively Breaking Large Pages to Improve Memory Overcommitment Performance in VMware ESXi. In *International Conference on Virtual Execution Environments (VEE)*.

[43] Faruk Guvenilir and Yale N. Patt. 2020. Tailored Page Sizes. In *International Symposium on Computer Architecture (ISCA)*.

[44] Jerry L. Hintze and Ray D. Nelson. Violin Plots: A Box Plot-Density Trace Synergism. *The American Statistician* 52, 2 (1998), 181–184.

[45] Intel. 2017. 8th Generation Intel® Core™, Pentium® and Celeron® Processor Families (Coffee Lake S). https://www.intel.com/content/www/us/en/design/products-and-solutions/processors-and-chipsets/coffee-lake-s/overview.html.

[46] Intel. 2020. 10th Generation Intel Core Processor Families. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/10th-gen-core-families-datasheet-vol-1-datasheet.pdf.

[47] Intel Corporation 2020. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel Corporation. https://software.intel.com/en-us/articles/intel-sdm.

[48] Subramanian S. Iyer. Heterogeneous Integration for Performance and Scaling. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 6, 7 (July 2016), 973–982.

[49] Bruce Jacob and Trevor Mudge. Virtual Memory in Contemporary Microprocessors. *IEEE Micro* 18, 4 (July 1998), 60–75.

[50] Gokul B. Kandiraju and Anand Sivasubramaniam. 2002. Going the Distance for TLB Prefetching: An Application-driven Study. In *International Symposium on Computer Architecture (ISCA)*.

[51] Ajaykumar Kannan, Natalie E. Jerger, and Gabriel H. Loh. 2015. Enabling interposer-based disintegration of multi-core processors. In *International Symposium on Microarchitecture (MICRO)*.

[52] 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 *International Symposium on Computer Architecture (ISCA)*.

[53] Vasileios Karakostas, Jayneel Gandhi, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift, and Osman Ünsal. 2016. Energy-Efficient Address Translation. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[54] Vasileios Karakostas, Osman Ünsal, Mario Nemirovsky, Adrián Cristal, and Michael M. Swift. 2014. Performance analysis of the memory management unit under scale-out workloads. In *IEEE International Symposium on Workload Characterization (IISWC)*.

[55] The kernel development community. [n.d.]. Kernel Samepage Merging. https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html. [Accessed April, 2016].

[56] Michael Kerrisk. [n.d.]. mmap(2) — Linux manual page. http://man7.org/linux/man-pages/man2/mmap.2.html. [Accessed September, 2017].

[57] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. KVM: The Linux Virtual Machine Monitor. In *Linux Symposium*.

[58] David Kroft. 1981. Lockup-Free Instruction Fetch/Prefetch Cache Organization. In *International Symposium on Computer Architecture (ISCA)*.

[59] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. 2016. Coordinated and Efficient Huge Page Management with Ingens. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. 705–721. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kwon

[60] Chen Li, Rachata Ausavarungnirun, Christopher J. Rossbach, Youtao Zhang, Onur Mutlu, Yang Guo, and Jun Yang. 2019. A Framework for Memory Oversubscription Management in Graphics Processing Units. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[61] Daniel Lustig, Abhishek Bhattacharjee, and Margaret Martonosi. TLB Improvements for Chip Multiprocessors: Inter-Core Cooperative Prefetchers and Shared Last-Level TLBs. *ACM Transactions on Architecture and Code Optimization (TACO)* (2013).

[62] Mark Mumy. 2014. SAP IQ and Linux Hugepages/Transparent Hugepages. http://scn.sap.com/people/markmumy/blog/2014/05/22/sap-iq-and-linux-hugepagestransparent-hugepages.

[63] Timothy Merrifield and H. Reza Taheri. 2016. Performance Implications of Extended Page Tables on Virtualized x86 Processors. In *International Conference on Virtual Execution Environments (VEE)*.

[64] Ugljesa Milic, Oreste Villa, Evgeny Bolotin, Akhil Arunkumar, Eiman Ebrahimi, Aamer Jaleel, Alex Ramirez, and David Nellans. 2017. Beyond the Socket: NUMA-aware GPUs. In *International Symposium on Microarchitecture (MICRO)*.

[65] MongoDB, Inc. Disable Transparent Huge Pages (THP). https://docs.mongodb.org/manual/tutorial/transparent-huge-pages/. (2017).

[66] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. CACTI 6.0: A Tool to Understand Large Caches.

[67] Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan Cox. 2002. Practical, transparent operating system support for superpages. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[68] NVIDIA Corp. 2011. CUDA C/C++ SDK Code Samples. http://developer.nvidia.com/cuda-cc-sdk-code-samples.

[69] NVIDIA Corp. 2011. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf.

[70] NVIDIA Corp. 2017. NVIDIA GeForce GTX 1080. https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf.

[71] Ashish Panwar, Sorav Bansal, and K. Gopinath. 2019. HawkEye: Efficient Fine-grained OS Support for Huge Pages. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[72] M.-M. Papadopoulou, Xin Tong, A. Seznec, and A. Moshovos. 2015. Prediction-based superpage-friendly TLB designs. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[73] Peter Zaitsev. 2014. Why TokuDB Hates Transparent HugePages. https://www.percona.com/blog/2014/07/23/why-tokudb-hates-transparent-hugepages/.

[74] Binh Pham, Abhishek Bhattacharjee, Yasuko Eckert, and Gabriel H. Loh. 2014. Increasing TLB reach by exploiting clustering in page translations. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[75] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. 2012. CoLT: Coalesced Large-Reach TLBs. In *International Symposium on Microarchitecture (MICRO)*.

[76] Binh Pham, Jan Vesely, Gabriel Loh, and Abhishek Bhattacharjee. 2015. Large Pages and Lightweight Memory Management in Virtualized Systems: Can You Have it Both Ways?. In *International Symposium on Microarchitecture (MICRO)*.

[77] Redis Labs. [n.d.]. Redis. http://redis.io/. [Accessed April, 2016].

[78] Redis Labs. [n.d.]. Redis Latency Problems Troubleshooting. http://redis.io/topics/latency.

[79] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory Access Scheduling. In *International Symposium on Computer Architecture (ISCA)*.

[80] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2012. Cache-Conscious Wavefront Scheduling. In *International Symposium on Microarchitecture (MICRO)*.

[81] Ashley Saulsbury, Fredrik Dahlgren, and Per Stenström. 2000. Recency-based TLB Preloading. In *International Symposium on Computer Architecture (ISCA)*.

[82] A. Seznec. Concurrent Support of Multiple Page Sizes on a Skewed Associative TLB. *IEEE Trans. Comput.* 53, 7 (July 2004), 924–927.

[83] Richard L. Sites and Richard T. Witek. 1998. *ALPHA architecture reference manual*. Digital Press, Boston, Oxford, Melbourne.

[84] Dimitrios Skarlatos, Umur Darbaz, Bhargava Gopireddy, Nam Sung Kim, and Josep Torrellas. 2020. BabelFish: Fusing Address Translations for Containers. In *International Symposium on Computer Architecture (ISCA)*.

[85] Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas. 2020. Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[86] Stephen Somogyi, Thomas F. Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. 2006. Spatial Memory Streaming. In *International Symposium on Computer Architecture (ISCA)*.

[87] Vijayaraghavan Soundararajan, Mark Heinrich, Ben Verghese, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. 1998. Flexible Use of Memory for Replication/Migration in Cache-coherent DSM Multiprocessors. In *International Symposium on Computer Architecture (ISCA)*.

[88] Splunk Inc. 2013. Transparent Huge Memory Pages and Splunk Performance. http://docs.splunk.com/Documentation/Splunk/6.1.3/ReleaseNotes/SplunkandTHP.

[89] Shekhar Srikantaiah and Mahmut Kandemir. 2010. Synergistic TLBs for High Performance Address Translation in Chip Multiprocessors. In *International Symposium on Microarchitecture (MICRO)*.

[90] Standard Performance Evaluation Corp. 2006. SPEC CPU2006 Benchmarks. (2006). http://www.spec.org/cpu2006.

[91] J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu. 2012. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing*. Technical Report IMPACT-12-01. Univ. of Illinois at Urbana-Champaign, IMPACT Research Group.

[92] Mark Swanson, Leigh Stoller, and John Carter. 1998. Increasing TLB Reach Using Superpages Backed by Shadow Memory. In *International Symposium on Computer Architecture (ISCA)*.

[93] M. Talluri and M. D. Hill. 1994. Surpassing the TLB Performance of Superpages with Less Operating System Support. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[94] Ben Verghese, Scott Devine, Anoop Gupta, and Mendel Rosenblum. 1996. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[95] VoltDB, Inc. [n.d.]. VoltDB Documentation: Configure Memory Management. https://docs.voltdb.com/AdminGuide/adminmemmgt.php.

[96] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo, and Xiaoming Li. 2011. Selective Hardware/Software Memory Virtualization. In *International Conference on Virtual Execution Environments (VEE)*.

[97] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Translation Ranger: Operating System Support for Contiguity-Aware TLBs. In *International Symposium on Computer Architecture (ISCA)*.

[98] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh. 2018. Modular Routing Design for Chiplet-Based Systems. In *International Symposium on Computer Architecture (ISCA)*.

[99] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler. 2016. Towards High Performance Paged Memory for GPUs. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[100] William K. Zuravleff and Timothy Robinson. 1997. Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order. US Patent No. 5,630,096.