# Supporting Superpages in Non-Contiguous Physical Memory*

Yu Du, Miao Zhou, Bruce R. Childers, Daniel Mossé, and Rami Melhem
Department of Computer Science University of Pittsburgh
{fisherdu,miaozhou,childers,mosse,melhem}@cs.pitt.edu

## Abstract

*For memory-intensive workloads with large memory footprints, superpages are effective to avoid address translation overhead, which can be a critical performance bottleneck. A superpage is a large virtual memory page that is mapped to an equivalently-sized amount of contiguous physical memory pages. Superpage mapping assumes physical memory does not contain retired pages, which is an important technique to improve memory resilience: the OS avoids allocating physical pages that have detected errors. Retired pages create unusable "holes" in the physical memory. We show that even a small percentage of retired pages makes it very difficult to find enough contiguous memory to form superpages.*

*To address this problem, we propose GTSM, or* gap-tolerant sequential mapping*, that allows superpages to be formed even in the presence of retired physical pages. A new page table format is also proposed to support GTSM. This format has similar storage efficiency as traditional superpaging to hold address translations in the last-level cache. To further compress the page table and improve cache hit rates for address translation in large memory footprint workloads, we also propose an extended format that reduces the page table size by 50%. In comparison to an ideal memory without any retired physical pages, we show that our technique, with retired pages, achieves nearly 96.8% of the performance of traditional 2MB superpaging.*

## 1. Introduction

With the rise of big data and cloud computing, workload memory footprints keep increasing, putting more pressure on the virtual memory subsystem, whose performance still needs to be improved for large applications [4]. Performance overhead includes a large number of Translation Lookaside Buffer (TLB) misses, which cannot be mitigated by simply increasing the number of TLB entries, given that increasing TLB size causes longer access latency and increases energy consumption. A solution for memory-intensive workloads with large memory footprints and random access patterns is to use superpages [4], which map a large contiguous virtual memory range to an equal-sized physical memory range. Therefore, the number of entries in the TLB and page tables is drastically reduced (by the ratio of superpage to page size, typically 2MB and 4KB, respectively). One study showed that up to 49.6% performance improvement can be obtained by utilizing 64KB superpages for SPEC CPU2006 workloads [16]. However, traditional superpages require the physical memory to be contiguous, which is problematic when *page retirement* is used to avoid uncorrectable errors in memory. Page retirement is a lightweight mechanism that retires/marks pages as unusable [1, 29] and prevents them from being allocated in the future; the drawbacks include creating holes, rendering physical memory not completely contiguous. We illustrate these issues with three scenarios.

First, when memory errors are considered, allocation of a large contiguous physical memory block (superpage) is more difficult. Recent studies show that modern DRAM error rates are orders of magnitude higher than previously reported [25, 14]. Error Correcting Codes (ECC) are commonly used to protect memory from one or multiple bit errors. Recent studies also showed that memory blocks that suffer from correctable memory errors are much likely to subsequently face uncorrectable errors [25]. A field study showed that retiring 1% of pages can cover 92% of memory errors [14].

Second, when techniques to reduce power are used (as is increasingly the case), more page retirement can be introduced. For example, the DRAM refresh interval can be increased [2, 17] to save static power. Given that the refresh interval of a DRAM cell must be shorter than its data retention time, and that cell retention time is not uniform (due to process variation), a small portion of cells have much shorter retention than other cells. Traditionally, DRAM refresh interval is determined by the cells with the shortest retention time. However, a recent study shows that retiring no more than 2% of memory pages can extend DRAM refresh interval from 64ms to 256ms [2].

Third, non-volatile memories are being introduced in (hybrid) main memory systems to reduce memory static power [22, 33, 23]. These non-volatile memories, however, have limited write endurance [21], and cells gradually become non-programmable "bad" cells. Mechanisms have been proposed for error correction [24], but their limited error correction resources can tolerate limited number of errors. When uncorrectable memory errors occur, page retirement is used.

Although page retirement is a simple and effective way to address different sources of memory errors, the retired pages also create many unusable holes in the physical address space and render the space non-contiguous. As we show in Section 2.3, even a small number of memory errors can make it very difficult to find enough contiguous physical memory

*This research is supported partially by National Science Foundation grants CNS-1012070 to the PCM@Pitt research group.

blocks to support traditional superpages. Page retirement is not the only reason to have non-contiguous physical memory. Memory fragmentation and non-migratable memory (e.g., IO memory holes) can also make it difficult to find contiguous physical memory blocks to construct superpages.

In this work, we propose a new approach to construct superpages from non-contiguous physical memory. By utilizing a block selection bitmap, a superpage is mapped to multiple equal-sized small memory blocks (i.e., physical pages) instead of a single large contiguous memory block. The approach is supported through a page table format that complements existing ones.

This paper makes the following contributions:

- In Section 2, we add to the knowledge that superpages are critical for workloads with large memory footprints and simply increasing the size of 4KB-page TLB is not enough.
- In Section 3, we describe a new *gap-tolerant sequential mapping* that includes (a) a new page table format, and (b) an access mechanism to support superpages for non-contiguous physical memory. We also describe a new page table compression scheme that uses (a) a variant of the new page table format to further reduce the page table size by half, and (b) a matching algorithm to construct the compressed page table.
- In Section 5, we present a quantitative analysis showing that our gap-tolerant sequential mapping can achieve 96.8% of the performance of an ideal 2MB superpages (without retired pages) for memory with retired pages.

Further, in Section 2 we describe the basics of x86-64 address translation, and in Section 4 we describe our experimental setting and workloads. Sections 6 and 7 summarize related work and conclusions.

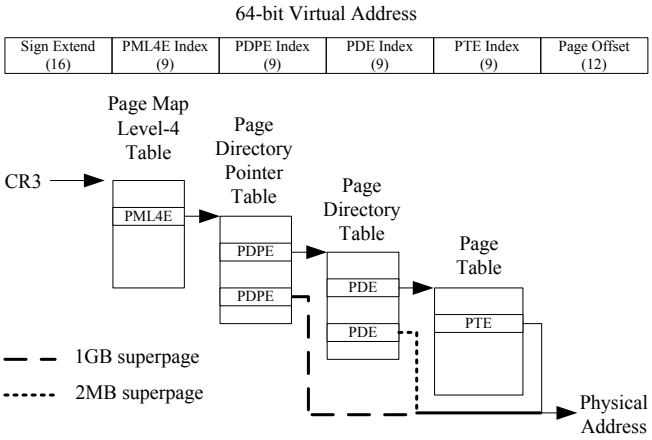
## 2. Background and Motivation

To simplify the presentation, we use x86-64 as our baseline architecture. The proposed ideas are applicable to other processor architectures that support superpages. To further simplify the presentation, a traditional *page* is 4KB and *superpages* are 2MB, unless otherwise noted. Our scheme can support other superpage sizes (e.g., 1GB superpages are also common).

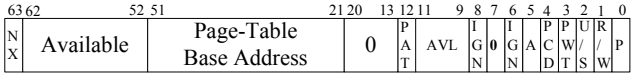
### 2.1. X86-64 Virtual Address Translation

Virtual memory mechanisms use page tables (PTs) to map between virtual pages and physical pages for every memory access. To speed up translation, physical addresses of recently-accessed virtual pages are cached in the TLB. On a TLB miss, a hardware page walker traverses the page table to translate the virtual page address.

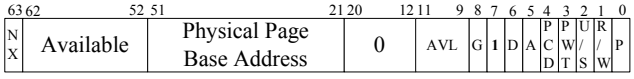
In x86-64, as shown in Figure 1(a), the PT has four levels, and a system register (*CR3*) points to the PT root node. The corresponding translation entry at each level is *Page-Map Level-4 Entry* (PML4E), *Page Directory Pointer Entry* (PDPE), *Page Directory Entry* (PDE) and *Page Table Entry*




(a) x86-64 page table




(b) 4KB-page PDE




(c) 2MB-page PDE

**Figure 1: VA-to-PA translation in the x86-64 architecture.**

(PTE). For each valid 4KB virtual page, the translation entries (PML4E, PDPE and PDE) point to the base address of the next level node. The size of the translation entry at each level is 8 bytes. With a 4KB page, there are 512 translation entries per node, which are indexed by 9 virtual address bits. Only 48 virtual address bits are used in current x86-64 implementations: the high 36 bits (9 x 4) are used to traverse the page table levels and the low 12 bits are the page offset.

Besides the TLB, recently accessed translation entries are also cached in the MMU as partial translations, which can be used to speed up page walking [3]. For example, PDE entries can be cached in a PDE cache. If the PDE of a virtual address hits in the PDE cache, the page walker needs to access only the last-level PTE to complete address translation.

X86-64 superpage implementation has a similar structure. Because the mapping is one-to-one, a 2MB superpage needs only a three-level PT (PML4, PDPE and PDE). The 7th bit of a PDE indicates whether the PDE points to a page table of PTEs, or to the physical base address of a 2MB superpage. Figures 1(b) and 1(c) show the format of PDE as a 4KB-page and 2MB-page PDE, respectively. Similarly, for a 1GB superpage, the 7th bit of a PDPE indicates whether a PDPE points to a page directory table of PDEs, or to the physical base address of the superpage.

### 2.2. Understanding Address Translation Overhead

There are three performance advantages to superpages. First, superpages increase TLB reach by the ratio of the size of a superpage to a normal-page-size page (i.e., TLB can cache

2-3 orders of magnitude larger address space), reducing TLB miss rate. Second, superpages reduce the number of levels during page walk, consequently reducing the latency of a TLB miss. Third, superpages significantly reduce the size of the PT; Table 1 shows PT sizes for different workload memory footprints and page sizes. Note that for a workload with a 16GB memory footprint, the PT size for a traditional 4KB page is 32MB, which is already beyond the capacity of the Last Level Cache (LLC) for most processors. With 2MB superpages, the size of the page table is reduced to just 64KB to easily fit in the cache.

| Memory Footprint | 1GB | 16GB |
|---|---|---|
| 4KB page | 2MB | 32MB |
| 2MB superpage | 4KB | 64KB |
| 1GB superpage | 8B | 128B |

**Table 1: Page table sizes for different workload memory footprints and page sizes assuming an 8-byte page table entry.**

To further understand the overhead of address translation, we characterized the performance of different problem sizes and TLB configurations for the GUPS workload [10], which is memory-intensive with a random access pattern. We use cycles per instruction (CPI) for performance. We study three TLB configurations: a 512-entry 4KB-page TLB, a 512-entry 2MB-page TLB and an 256K-entry 4KB-page TLB, which is much larger than any practical TLB design. Recent work shows that TLB reach can be improved by coalescing multiple TLB entries with similar contents [20, 19]. The effective TLB reach of a 256K-entry 4KB-page TLB is 1GB, which is an upper bound that can be achieved by TLB coalescing.

As shown in the left side of Figure 2, when memory footprint is 1GB, a 256K-entry 4KB TLB has similar performance as a 512-entry 2MB TLB, since the memory footprint is not larger than the 1GB TLB reach. When the memory footprint increases to 16GB (right side of Figure 2), which is much more than the 1GB TLB reach, the performance improvement from increasing TLB reach becomes very small, but superpages perform very well.

Figure 2 characterizes the CPU cycles for an instruction on average, to understand the sources of the address translation overhead. We note that approximately 15 cycles are needed to access data, execute the instruction and account for address translation overhead in the first 3 levels (up to PDE, labeled No PTE in the figure). Compared to a 512-entry 2MB-page TLB, the additional address translation overhead of a 4KB-page TLB mainly comes from accessing PTEs.

The number of cycles to access PTEs can be broken down into two components: the cycles to access PTEs assuming all PTEs always fit in the LLC (PTE-Ideal LLC hit, middle cycles), and the cycles to access main memory if a PTE is not cached (PTE-Memory, at the top). For workloads with large memory footprints (16GB), the performance overhead of accessing PTEs is dominated by main memory accesses.

In conclusion, to avoid address translation overhead from becoming a performance bottleneck, it is critical for workloads with large memory footprints to support superpages, which avoids accessing PTEs.

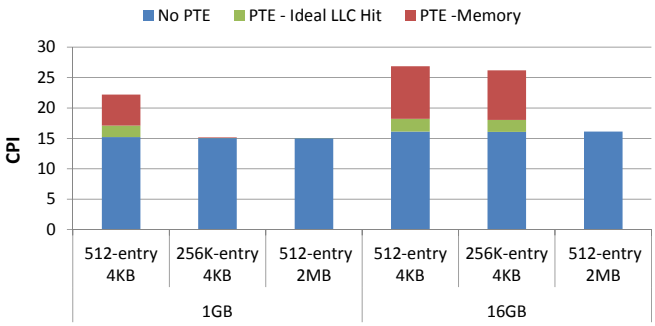



**Figure 2: CPI breakdown with different problem sizes and TLB configurations for the GUPS workload.**

### 2.3. Page Retirement and Memory Fragmentation

Given that superpages need contiguous physical memory, the physical memory can become fragmented when there is even a small percentage of retired pages. Figure 3 shows the probability of finding a contiguous memory block (of sizes 2MB, 128KB, 64KB and 32KB) as a function of percentage of retired 4KB pages (retired pages are uniformly distributed). The probability of allocating a 2MB superpage quickly approaches zero if the number of retired pages increases (e.g., for 0.5% retired pages, the probability is less than 8%). This implies that a traditional superpage implementation will be ineffective with retired pages. Nevertheless, it is relatively easy to find **small contiguous memory areas** when the percentage of the retired pages is small. To ensure that at least 60% of the memory blocks are contiguous, the threshold on the percentage of the retired pages for 128KB, 64KB and 32KB superpages is 1.6%, 3.1% and 6.1%, respectively.

In the next section we describe a new method to construct superpages from multiple small memory fixed-sized areas instead of a single large contiguous memory area.

## 3. Superpage in Non-Contiguous Memory

We develop a new mapping to support superpages in memory with retired pages, and the necessary hardware to implement this mapping.

### 3.1. Problem Definition

The goal is to find storage-efficient page table formats to accommodate superpages in the context of non-contiguous physical memory. We identified four requirements to achieve this goal:

1. Allow mapping a superpage to multiple non-contiguous memory area;

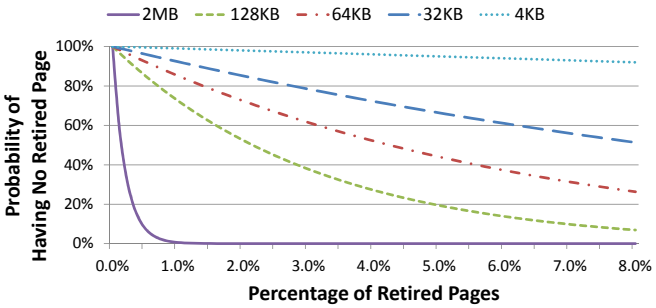



**Figure 3: Probability of a memory block (of sizes 2MB, 128KB, 64KB and 32KB) to be contiguous (no retired pages) for different percentages of retired 4KB pages.**

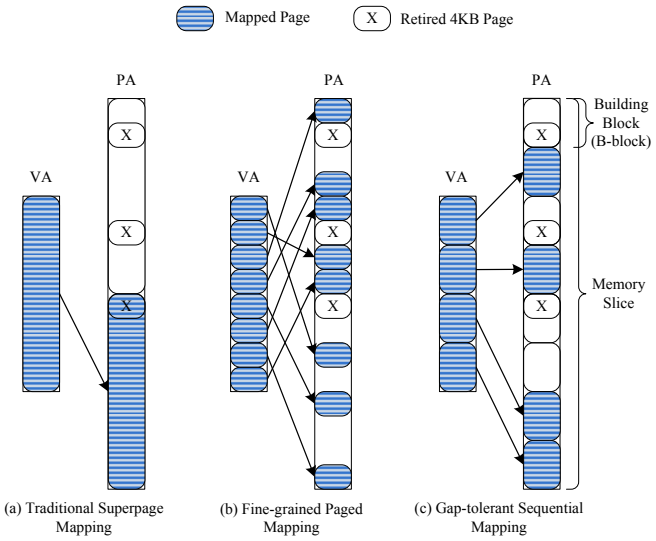



**Figure 4: Examples of different address mapping schemes.**

2. Have similar size as traditional superpage table format;
3. Guarantee that address translation is completed in a fixed number of steps; and,
4. Allow mixing superpages and traditional pages.

For backward compatibility and deployment, our new PT format is an optional extension to allow a portion of the memory to be mapped as superpages and the rest of memory to follow a traditional PT format.

### 3.2. Gap-tolerant Sequential Mapping (GTSM)

When the physical memory is littered with retired pages, it is problematic to find a large contiguous memory block to establish a mapping. We devised *Gap-tolerant Sequential Mapping* (GTSM) to support superpages in memory with retired pages. Figure 4 shows three ways to map a virtual memory space (VA) of the size of a superpage to physical memory (PA) that contains errors (marked with an X). Figure 4(a) shows traditional superpage mapping, that maps VA to contiguous PA (in this case, there is no contiguous physical space that can accommodate a superpage).

Figure 4(b) is the traditional page mapping, where each virtual page can be mapped to an arbitrary non-retired physical page. This flexibility is not free: the storage cost of fine-grained paged mapping is orders of magnitude higher than traditional superpage mapping. Figure 4(c) shows how GTSM divides a virtual superpage into multiple fixed-size smaller virtual blocks, which are sequentially mapped to memory (*B-blocks*, or building blocks). B-blocks are bigger than a regular page and together form a memory *slice*, whose size is twice the size of a superpage. Note that the utilization of memory is **not** only 50% with GTSM because remaining unmapped fragmented memory can still be used for traditional pages.

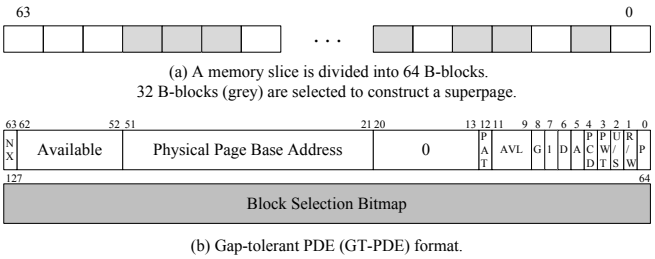


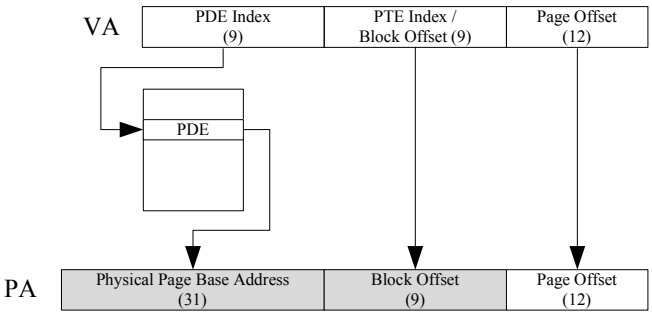

**Figure 5: Gap-tolerant PDE (GT-PDE) format.**

To maintain a one-to-one mapping between virtual blocks and B-blocks, exactly half of the B-blocks participate in the mapping, given the size of the memory slice. Any B-block that contains at least one retired page cannot be used for GTSM. Note that GTSM is a generalized form of traditional superpage mapping, but it is more flexible to take into account retired pages. If there is no retired pages in a memory slice, GTSM creates the same (i.e., contiguous) memory mapping as traditional superpage mapping. By sacrificing flexibility of small/traditional page mapping, GTSM tolerates retired pages and maintains a page table that has similar storage efficiency as superpage mapping.

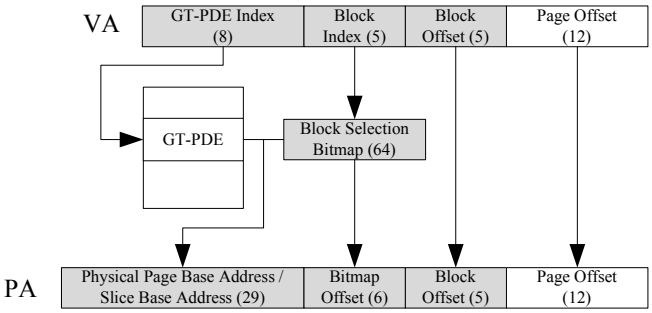
### 3.3. Gap-tolerant Page Directory Entry (GT-PDE)

For a 2MB superpage, PDE (page directory entry) is the last level of address translation; the PDE format contains the physical page frame base address and control flags of the superpage (present bit, access bit, dirty bit, etc.). To support GTSM, the 8-byte PDE is extended to a 16-byte GT-PDE (Gap-Tolerant PDE). Figure 5(a) shows a memory slice divided into 64 B-blocks with half of the B-blocks selected to construct a GTSM superpage. As shown in Figure 5(b), to minimize the impact on the OS, we keep the first 64 bits of a GT-PDE the same as a traditional 2MB-page PDE. An extra 64-bit B-block selection bitmap is appended for GTSM. The corresponding bit will be set to 1 if the B-block is selected in the mapping. Otherwise, the bit will be set to 0.

Figure 6(a) shows address translation of a traditional 2MB-page PDE. The 9-bit PDE index/block offset is used to select a PDE among 512 regular PDEs. The 9-bit PTE index will be kept unchanged in the translated physical address. Since the default size of each page directory table is 4KB, it can hold 256 GT-PDEs instead of 512 regular PDEs. To avoid changing the size of the page directory table (4KB) and the




(a) Address translation of a 2MB-page PDE.




(b) Address translation of a 4MB-page GT-PDE.

**Figure 6: Address translation using GT-PDE-4MB.**

size of the mapped memory range (1GB), each GT-PDE entry needs to map a 4MB superpage instead of a traditional 2MB superpage. Based on GTSM, a 4MB superpage is mapped to a 8MB memory slice. Since each memory slice has 64 B-blocks, the size of each B-block is 128KB.

Figure 6(b) shows the address translation of a 4MB-page GT-PDE. Only the upper 8 bits of the PDE index are needed to index a GT-PDE entry among 256 GT-PDEs. Since each B-block is 128KB, only the low 5 bits of PTE index are used as block offset and kept unchanged during address translation. The remaining 5 bits between GT-PDE index and block offset are treated as block index. Block index is translated using the block selection bitmap of the selected GT-PDE. Same as a 2MB-page PDE, the physical page base address field of a GT-PDE entry is 31 bits. Because the mapped sliced is aligned at an 8MB boundary, the low 2 bits of the physical page base address field are always zeros and ignored in the translated physical address. To translate block index *K* (0-31), the block selection bitmap is scanned to find the *K* selected bit, whose position in the bitmap (0 - 63) indicates the B-block that the virtual block is mapped to.

Because at least half of the B-blocks in the memory slice must have no retired pages, we show in Figure 7 the probability of constructing a valid mapping for different percentages of usable B-blocks, which are assumed to be randomly distributed in memory. A threshold of 60% is enough for most memory slices (93.3%) to find a valid mapping. Because only half of the B-blocks in a slice are used in GTSM, the memory capacity that can be mapped with GT-PDE is 46.6% with 60%

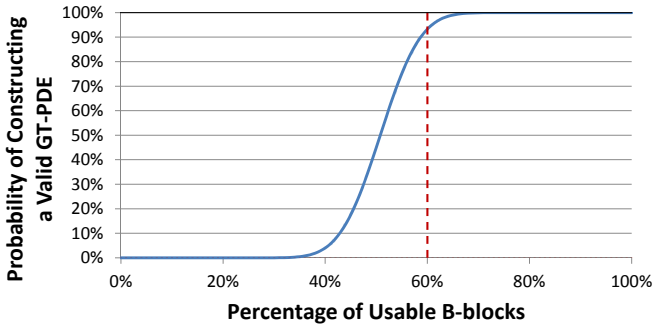



**Figure 7: Probability to construct a valid GT-PDE mapping for different percentages of usable B-blocks.**

B-blocks usable. Recall that the remaining memory capacity can be mapped as traditional pages.

### 3.4. Tolerating More Retired Pages

There is a trade-off between B-block size and number of retired pages allowed (robustness of mechanism). The B-block size used by a 4MB-page GT-PDE is 128KB. From Figure 3 we see that to tolerate more retired pages, a smaller B-block size should be chosen (to ensure enough usable B-blocks exist to find a valid mapping).

When the bit of the block selection bitmap represents a smaller B-block, the format of the GT-PDE is not changed and the size of the superpage represented by a GT-PDE is reduced. To avoid changing the size of the mapped memory range (1GB), the page directory table needs to be expanded to hold more GT-PDEs. This change to use large pages (e.g., 16KB) as the page directory table is feasible. Some architectures, like ARM, already have this capability.

We limit the B-block size to 128KB, 64KB or 32KB. Table 2 shows the basic parameters of GT-PDEs. Supporting 16KB or smaller B-blocks requires changing the GT-PDE format to have more bits for the physical page base address. Although supporting 256KB or larger B-block size is possible, it is not considered for two reasons. First, 256KB or larger B-block size implies tolerating fewer retired pages (< 1%), which is not our goal. Second, the size of the page directory table will be smaller than 4KB and become partially filled (i.e., wasted space) assuming a minimum page size of 4KB.

| GT-PDE Mode | B-block Size | Page Directory Table Size | Retired Page Threshold |
|---|---|---|---|
| 4MB-page | 128KB | 4KB | 1.6% |
| 2MB-page | 64KB | 8KB | 3.1% |
| 1MB-page | 32KB | 16KB | 6.1% |

**Table 2: Parameters of GT-PDEs with different B-block sizes.**

The translation process of GT-PDE with a smaller B-block size is similar to a 4MB-page GT-PDE. However, with a smaller B-block size, fewer address bits are used as block offset. At the same time, the page directory table is expanded to

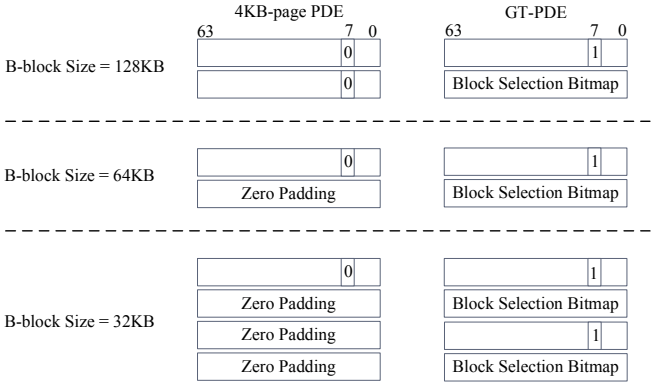



**Figure 8: Decoding GT-PDE.**

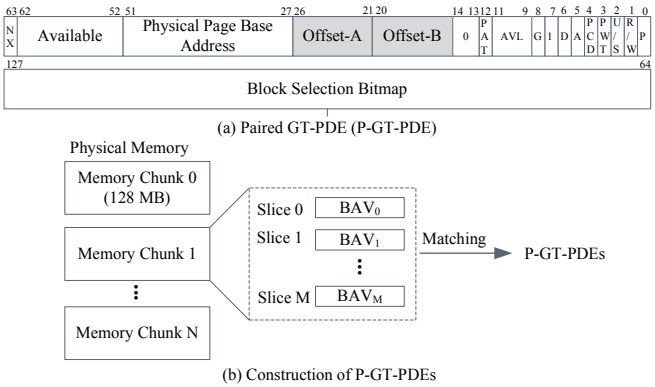



**Figure 9: P-GT-PDE and its construction.**

hold more GT-PDEs. More address bits are used to locate the GT-PDE in the page directory table.

### 3.5. Mixing Traditional Pages and Superpages

When GT-PDE is enabled, a page directory table can store both 4KB-page PDEs and GT-PDEs at the same time. Each 4KB-page PDE has the PT base address that points to the PT of PTEs. Each PTE will further point to each mapped 4KB pages. Each GT-PDE directly points to the physical base address of the mapped memory slice. 4KB-page PDEs need zero padding to fill the unused space if the page directory table is expanded. Figure 8 shows how to decode addresses using GT-PDE for various B-block sizes (see explanations below). When GT-PDE is enabled, the page directory table is accessed at aligned 16-byte granularity. Similar to a traditional 2MB-page PDE, the 7th bit of the accessed 16-byte is utilized to determine whether a PDE is a 4KB-page PDE or a GT-PDE. If the 7th bit is zero, the PDE should be decoded as a 4KB-page PDEs, otherwise the PDE should be decoded as a GT-PDE.

When B-block size is 128KB, every 16 bytes of the page directory table can store either two 4KB-page PDEs or one GT-PDE. There is no padding needed. When B-block size is 64KB, the size of the page directory table is doubled. To ensure 4KB-page PDEs are evenly distributed in the page directory table, each 4KB-page PDE needs to be padded with an 8-byte zero padding. When B-block size is 32KB, each 4KB-page PDE needs to be padded with a 24-byte zero padding. The storage overhead of the padding is small because the dominant storage cost of 4KB-page PDEs comes from their PTEs. When B-block size is 32KB, if the first access of a 4KB-page PDE points to the bottom-half 16 bytes, which are the zero padding, a second access to the top-half 16 bytes is needed. Similar to the storage cost, the performance overhead of the extra access is small because the dominant address translation overhead is from accessing PTEs.

### 3.6. Compressing GT-PDEs

Recent research showed that TLB entries with similar contents can be coalesced to store more address translations in the TLB [20, 19]. Similarly, it is possible to halve the size of the page directory table by coalescing every two adjacent GT-PDEs. Figure 9(a) shows a Paired GT-PDE (P-GT-PDE) format to support GT-PDE coalescing. To coalesce two GT-PDEs with P-GT-PDE, two GT-PDEs use memory slices in a smaller range (128MB). Only the low 6 bits of physical page base address are different for the two slices. Also, the two GT-PDEs need to have the same block selection bitmap. Therefore, the low 6 bits of physical page base address of the second GT-PDE can be stored in the unused field (bit 15-20) of the first GT-PDE to form a P-GT-PDE. As shown in Figure 9(a), when a P-GT-PDE is accessed, it is simple to restore the coalesced GT-PDEs by masking the Offset-B field as zeros and overriding the Offset-A field with the value of the Offset-B field.

Figure 9(b) shows the matching process to construct P-GT-PDEs. First, the physical memory is divided into multiple aligned 128MB memory chunks. Each chunk is further divided into memory slices. Based on the distribution of retired pages, each slice has a 64-bit Block Availability Vector (BAV), which indicates which B-blocks of the slices can be used in the P-GT-PDE mapping. If a B-block is usable, the corresponding bit in BAV is set to 1, otherwise it is set to 0. A matching algorithm finds memory slices that should be paired.

Algorithm 1 is our matching algorithm to find GT-PDE pairs to construct P-GT-PDEs. First, for each memory chunk, we can get a BAV for each memory slice. A 128MB memory chunk has 64 2MB-size slices or 32 4MB-size slices. Then, BAVs are sorted based on the number of usable B-blocks in ascending order. The algorithm sequentially scans the remaining BAVs to find two unprocessed BAVs that can be paired using the bitwise-AND. If the result has at least 32 bits set, the slices can share a valid block selection bitmap. Matching continues until there are no more unprocessed BAVs. A special case is a memory slice which does not have any retired page, whose top half and bottom half slices can be *Self-Paired*. For a Self-Paired P-GT-PDE, the block selection bitmap is all ones.

Because the matching algorithm is done locally for each

**Algorithm 1** Construction of P-GT-PDEs in a Memory Chunk

Parameters:
$B_0 ... B_N$: a group of BAVs

Initialize the state of each BAV to *Unprocessed*.
Sort BAVs based on the number of usable B-blocks of each BAV in ascendant order.
Any BAVs with all B-blocks usable are marked as *Self-Paired*.
**while** the number of *unprocessed* BAVs $\geq 2$ **do**
  Scan the BAV list to find the first *Unprocessed* BAV $B_i$.
  **for** each remaining *unprocessed* BAV $B_j$ **do**
    $B_{merged}$ = $B_i$ bitwise AND $B_j$.
    **if** the number of usable B-blocks of $B_{merged} \geq 32$ **then**
      Record $(B_i, B_j)$ as a valid P-GT-PDE.
      Mark $B_i$ *and* $B_j$ as *Paired*.
      Go to find next unprocessed BAV $B_i$ to process.
    **end if**
  **end for**
  Mark $B_i$ as *Discarded*.
**end while**

128MB memory chunk, the time complexity of the algorithm is proportional to memory capacity. We tested our serial version of the algorithm on a 2.8GHz Intel Xeon E5-2680v2 processor. It takes less than 0.1s to find all BAV pairs for 128GB memory.

### 3.7. Hardware Implementation

To support GTSM, we introduce three hardware changes. First, a new 64-bit Gap-Tolerant Page Table Control Register (GTPTCR) is used to manage the parameters of GTSM. Second, the hardware page walker is extended to support loading missed TLB entries from GT-PDEs. Third, the PDE cache in the MMU is extended to hold 16-byte GT-PDEs. Next, we describe these hardware changes in detail.

As shown in Figure 10(a), GTPTCR has three fields: GT, P and BS. GT indicates whether GTSM is enabled. The page table of a process can use both 4KB-page PDE and traditional 2MB-page PDE. Or the page table of a process can use both 4KB-page PDE and GT-PDE. To avoid adding an extra flag in the PDE, the page table of a process cannot use both traditional 2MB-page PDE and GT-PDE. P indicates whether P-GT-PDE format is used. BS indicates the B-block size. If BS is $n$, the corresponding B-block size is $2^n \times 4$KB. The remaining unused bits in GTPTCR are reserved for future extension. Table 3 shows the PDE modes that are defined by the GTPTCR.

| GT | P | BS | B-block Size | PDE Mode |
|---|---|---|---|---|
| 0 | 0 | 0 | - | 2MB-page PDE |
| 1 | 0 | 3 | 32KB | 1MB-page GT-PDE |
| 1 | 0 | 4 | 64KB | 2MB-page GT-PDE |
| 1 | 0 | 5 | 128KB | 4MB-page GT-PDE |
| 1 | 1 | 3 | 32KB | 2MB-page P-GT-PDE |
| 1 | 1 | 4 | 64KB | 4MB-page P-GT-PDE |

**Table 3: List of all PDE modes in GTPTCR.**

When a GT-PDE/P-GT-PDE is accessed, the hardware page table walker needs to translate the block index to a bitmap offset using the block selection bitmap. As shown in Figure 10(b), each PDE cache entry needs an extra 15-byte storage to store the block selection bitmap (8 bytes) and byte-granularity prefix sums (7 bytes). When a GT-PDE/P-GT-PDE is loaded into a PDE cache entry, byte-granularity prefix sums are calculated and cached to reduce translation latency. As shown in Figure 10(b), byte-granularity prefix sum, $S_i$, is the accumulated number of 1s of the first $i+1$ bytes of the block selection bitmap. Similar bit-counting logic is already implemented in modern processors and can be reused to reduce hardware implementation cost. For example, x86-64 POPCNT instruction counts the number of 1's of a 64-bit register in 3 cycles. $S_7$, which is the number of 1s of the whole block selection bitmap, is not stored because it is never used in address translation.

For each address translation using GT-PDE/P-GT-PDE, the byte-granularity prefix sums are compared with the value of *blockindex*. The $(i+1)^{th}$ byte of the block selection bitmap contains the matched bit position if $S_i > blockindex$ and $S_{i-1} \leq blockindex$. After the matched byte is determined, bit-granularity prefix sums are calculated for each bit position of the matched byte. The eight bit-granularity prefix sums are compared with the value of $blockindex - S_{i-1} + 1$, the bit position of the matched bit-granularity prefix sum is the translated bitmap offset. Also, address translation using the block selection bitmap can be done in parallel with other operations that are needed to fill a TLB miss (e.g., validating the access rights of the superpage). We assume that loading a TLB entry from a GT-PDE takes an extra 3 cycles than a traditional PDE. We also carried out a sensitivity study on this penalty (see Section 5).

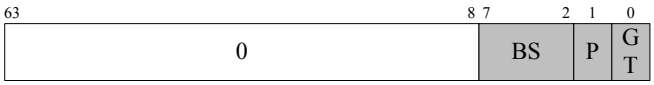
In this work, we assume that the baseline processor has a 32-entry PDE cache to store recently-accessed PDEs [3, 6]. The total storage overhead to support GT-PDE is 488 bytes: $8\ bytes\ (GTPTCR) + 32 \times 15\ bytes\ (PDE\ cache\ entries)$.

To minimize the changes to the MMU, our design does not change the TLB hierarchy. The hardware page table walker is enhanced to support GTSM. When address translation is completed, a 4KB TLB entry is inserted into the TLB hierarchy for the translated address. Early x86-64 processors have also used a similar method to support traditional superpages. Alternatively, the TLB hierarchy can be enhanced to provide native support for GT-PDE.

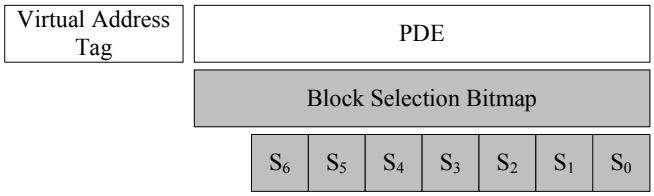
### 3.8. Software Support

To enable GTSM, the OS needs to support functions to 1) configure GTPTCR; 2) determine whether to use traditional or GT-PDE superpages based on the setting of GTPTCR; 3) track memory slices that can be mapped as GTSM superpages; and, 4) install and release GTSM superpages.

To track memory slices that can be mapped as GT-PDE superpages, a BAV array can be used to store the usability in-




(a) Gap-Tolerant Page Table Control Register (GTPTCR)




(b) PDE Cache Entry to support GT-PDE/P-GT-PDE

**Figure 10: Hardware implementation of GT-PDE.**

formation of B-blocks. Each memory slice has a dedicated 64-bit BAV. When the B-block size is 128KB, the memory storage cost of the BAV array is 128KB for 128GB main memory. If the B-block size is halved, the memory storage cost of the BAV array will be doubled.

When the OS boots, it initializes the BAV array using a fault map of pages with errors. The fault map can be either stored in a permanent storage or constructed with memory built-in self-test (mBIST) during boot. The OS needs to keep the BAV array updated by using information from kernel physical page allocator (e.g., Linux Buddy Allocator). Once a memory page of a B-block is allocated, the corresponding bit of the BAV needs to be set to 0. Once all the memory pages of a B-block are freed, the corresponding bit of the BAV needs to be set to 1. A memory slice can be used for GT-PDE memory allocation if more than half of its B-blocks are usable. To avoid scanning the BAV array for each GT-PDE memory allocation, all BAVs that can be used for GT-PDE allocations can be maintained in a dedicated list.

Unlike GT-PDE, P-GT-PDE should be used only for processes with very large memory footprints, and compressing GT-PDEs can further reduce TLB miss penalty. Because the matching algorithm described in Section 3.6 needs to be applied to the BAV array to find BAVs that can be paired, it is more expensive to make memory allocation with P-GT-PDE than GT-PDE. To utilize P-GT-PDE, physical memory should be allocated at the early stage of the process lifetime and released when the process is completed.

In this paper, we assume that all processes use the same B-block size. To support per-process B-block size, the OS needs to track BAVs at multiple granularities.

# 4. Experimental Methodology

## 4.1. Configuration

We use PTLsim [31], a cycle-accurate simulator, for performance evaluation. The simulation parameters are detailed in Table 4. The CPU is configured as a 3GHz out-of-order processor core with a 256KB L2 cache and a 2MB LLC cache slice. Main memory capacity is 128GB with 50ns access latency.

To evaluate different page table designs, we extended PTLsim with a TLB performance model. The L1 DTLB has 64 entries for 4KB pages and 32 entries for 2MB pages. The L1 ITLB has 64 entries for 4KB pages. The unified L2 TLB has 512 entries for both 4KB and 2MB pages. L1 TLB miss penalty is 7 cycles if it hits in L2 TLB.

Besides the two-level TLB, a MMU cache [3] is modeled. The MMU cache has 32 PDE/GT-PDE cache entries, 32 PDPE cache entries and 2 PML4E cache entries. Although the number of entries in the PDPE is larger than usual, we increased it to reduce TLB miss penalty for workloads with large memory footprints, as suggested in previous work [3, 6]. The MMU cache is indexed by virtual address and is concurrently looked up with L2 TLB [3]. We assume 5 cycles for the hardware page walker to access a PTE/PDE/PDPE/PML4E not including the cycles to load the entry from the cache/memory hierarchy. We assume that it takes an extra 3 cycles to access a GT-PDE due to the address translation latency using the block selection bitmap. The hardware page walker is not speculative (all configurations).

Since the B-block size is larger than a traditional page of 4KB (we experimented with 32KB, 64KB and 128KB), the selection of B-blocks in a memory slice has negligible impact on performance. For L1 and L2 cache, if a virtual page is mapped to different B-blocks, data at a given virtual address is still mapped to the same cache set. The selection of different B-blocks only affects the value of cache tags, the cache replacement sequence is kept unchanged. The memory pages for the page table are pre-allocated to simplify the simulation process. A similar reservation-based allocation policy has been used to allow MMU cache coalescing [6].

We use a Monte Carlo method to calculate the effective capacity of different GT-PDE/P-GT-PDE designs with different percentages of retired pages. To reduce the error introduced by the Monte Carlo method, we modeled randomly-distributed retired pages in a large physical memory sample (16PB capacity). We use Intel RdRand instruction [15] to generate different percentages of retired pages with a uniform random distribution.

## 4.2. Workloads

Since we study address translation overhead of virtual memory, we consider memory-intensive benchmarks with large memory footprints. We use these applications because they are becoming prevalent and suffer the most from address translation overhead. We choose *GUPS* [10], *Canneal* from PARSEC [8] and 7 benchmarks from Problem Based Benchmark Suite [26]. *GUPS* is a popular benchmark to test random memory access performance. *Canneal* is a cache-aware simulated annealing kernel to minimize the routing cost of a chip design. *Dict* is a benchmark to test performance of batch insertion, deletion and search operations with a dictionary

| | |
|---|---|
| CPU Core | 3GHz, out-of-order, 32KB L1 I/D |
| L2 Cache | 256KB, 8-way, 64-byte line size,<br>8-cycle latency |
| LLC | 2MB per core, 32-way,<br>64-byte line size, 20-cycle latency |
| L1 DTLB | 64-entry 4-way 4KB page<br>32-entry 4-way 2MB page |
| L1 ITLB | 64-entry 4-way 4KB page |
| L2 TLB | 512-entry 4-way 4KB/2MB page<br>7-cycle latency |
| MMU Cache | 32-entry 4-way PDE/GT-PDE cache<br>32-entry 4-way PDPE cache<br>2-entry PML4E cache<br>5-cycle PTE/PDE/PDPE/PML4E access<br>8-cycle GT-PDE access |
| Main Memory | 128GB DRAM, 50ns latency |

**Table 4: System settings.**

data structure. *BFS* runs a breadth first search in a directed graph. *SetCover* finds an approximate solution to the NP-hard set cover problem. *MST* finds the minimum spanning tree (MST) in an undirected graph. *SPMV* is multiplication between a sparse matrix and a dense matrix. *Matching* finds a maximal matching in an undirected graph. *MIS* finds a maximal independent set (MIS) in an undirected graph. With our current simulator, only single-threaded workloads are evaluated. Multi-threaded workloads should have similar results, given there will be even larger memory requirements by multiple applications or threads running concurrently. The pressure on the cache and sizes of page tables tend to be even bigger.

| Name | Memory Reads PKI | TLB Miss PKI | | Mem. Footprint(GB) | |
|---|---|---|---|---|---|
| | | 4KB | 2MB | Touched | Total |
| GUPS | 17.9 | 17.9 | 13.4 | 4.0 | 4.1 |
| Canneal | 24.4 | 21.2 | 2.9 | 3.7 | 4.0 |
| dict | 23.1 | 21.4 | 0.0 | 0.7 | 6.5 |
| BFS | 93.2 | 88.1 | 4.4 | 1.4 | 7.4 |
| setCover | 60.4 | 49.4 | 0.0 | 0.9 | 7.8 |
| MST | 50.0 | 43.2 | 0.0 | 1.0 | 13.0 |
| SPMV | 128.7 | 113.5 | 0.0 | 1.7 | 7.3 |
| matching | 119.1 | 109.7 | 0.0 | 0.9 | 6.2 |
| MIS | 144.4 | 124.3 | 0.0 | 1.3 | 7.3 |

**Table 5: Simulated workloads and PKIs.**

For the graph benchmarks, we use R-MAT graphs [9] as the input. For *Dict*, we use an uniform random distribution as the input. For each workload, we skipped the initialization phase and simulated 2 billion instructions. All benchmarks are 64-bit binaries, compiled with gcc 4.1.2. Table 5 shows the number of memory reads per 1000 instructions (PKI), TLB Miss PKI after a 512-entry L2 TLB (both 4KB pages and 2MB pages), and memory footprints of each workload (both the total footprint and the size of memory touched by the 2B-cycle simulation we ran). Most workloads can only touch a small portion of their total memory footprints during the simulation interval. A 512-entry 2MB-page L2 TLB can provide enough memory coverage (1GB). Most workloads have negligible TLB misses with ideal 2MB superpage.

**Characterizing Applications: PTE Access Breakdown** Since accessing PTEs is a major source of address translation overhead for workloads with large memory footprints, Figure 11 shows the breakdown of PTE accesses based on whether the PTE is accessed in memory, LLC, or L2 cache. We assume that the L2 cache is the fastest cache large enough to cache PTEs; essentially, caching PTEs in the L1 cache could cause significant adverse cache pollution and severely harm performance. The breakdown of PTE accesses is an inherent characteristic of each workload, and is sensitive to the memory footprints of the workloads. The workloads that we studied can be divided into two categories. The slower PT access category are those applications that have a large portion (i.e., more than 10%) of PTE accesses to memory, given that the access to memory is 7.5 times slower than LLC: *GUPS*, *Canneal*, *dict* and *BFS*. The second category are those applications that have faster time to access PTs, with few PTE access to memory: *setCover*, *MST*, *SPMV*, *matching* and *MIS*.

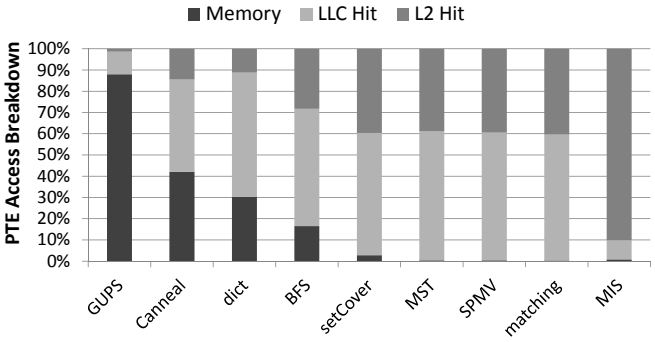



**Figure 11: PTE access breakdown.**

## 5. Results

This section presents simulation results of GT-PDE/P-GT-PDE superpages. We show how performance is improved in comparison to traditional 4KB pages. We also show *Ideal* case, that is, traditional 2MB superpages with no retired pages (in other words, no errors occur in memory). Traditional 2MB superpage is only suitable for memory where retired pages are rare. In the figures, we use GT-PDE-*x*MB to denote *x*MB-page GT-PDE. Similarly, we use P-GT-PDE-*x*MB to denote *x*MB-page P-GT-PDE.

### 5.1. TLB Miss Penalty

Since our proposed design does not change the TLB hierarchy, it has the same TLB miss PKI as the traditional 4KB page baseline. As a superpage table format, GT-PDE does not need to access a PTE for each page table walk, which significantly reduces the TLB miss penalty. Figure 12 shows the

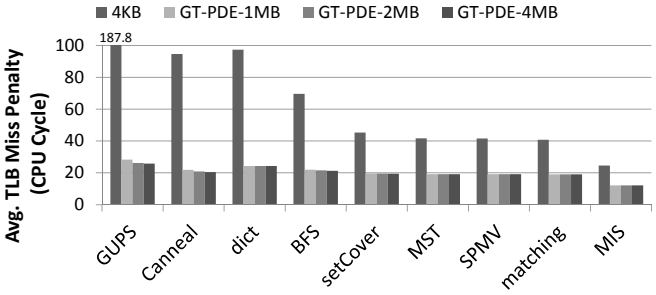



**Figure 12: Average TLB miss penalty.**

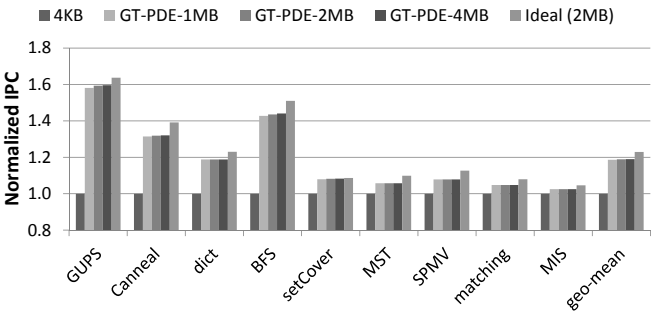



**Figure 13: IPC normalized to traditional 4KB page baseline.**

average TLB miss penalty of the traditional 4KB page baseline and GT-PDEs. Compared with Figure 11, we observe strong correlation between PTE access breakdown and the reduction of TLB miss penalty. For *GUPS*, *Canneal*, *dict* and *BFS*, average TLB miss penalty reduces by 40-160 CPU cycles because a significant portion of PTEs are accessed from memory for the traditional 4KB page baseline. For the workloads we studied, the average TLB miss penalty of GT-PDE for 1MB, 2MB, and 4MB are all similar because the page tables fit in the same cache level.

### 5.2. Performance

Figure 13 shows IPC improvement over the traditional 4KB page baseline. The graph shows the improvement for GT-PDE with different superpage sizes and *Ideal* superpages (i.e., superpages with no retired/faulty pages). Similar to Figure 12, we observe strong correlation between PTE access breakdown and IPC performance improvement. For *GUPS*, *Canneal*, *BFS* and *dict*, we observe significant performance improvement (20% to 60%) with GT-PDEs because PTEs are no longer accessed from memory. For *setCover*, *MST*, *SPMV*, *matching* and *MIS*, we observe moderate performance improvement because these workloads access PTEs that are mostly cached in the L2 and the LLC; the address translation overhead is not significant enough to cause a large difference in the IPC, which is approximately 2% to 8%. *MIS* has the lowest performance gain (2.6%). *MIS* has 90% PTE accesses to the L2 cache, and is less sensitive to address translation overhead. On average, GT-PDE-4MB achieves 96.8% performance of Ideal. The 3.2% overhead mainly comes from the extra 3 cycles to translate and access GT-PDE entries from the cache hierarchy. For the workloads we studied, the IPC

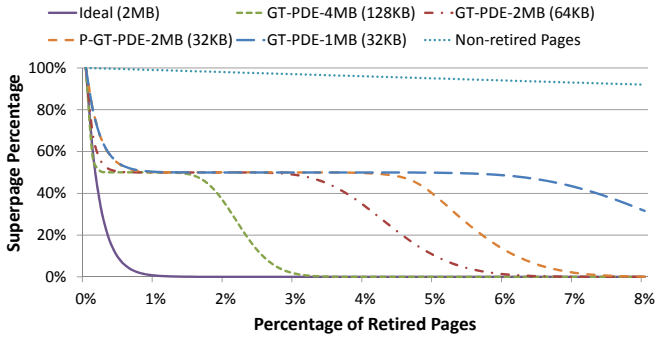



**Figure 14: Superpage percentage of different GT-PDEs.**

performance of GT-PDE for 1MB, 2MB, and 4MB are all similar because they use the same address translation procedure and have similar TLB miss penalty. Due to the same reasons (both fit in the L2 or LLC), even though the paired schemes (P-GT-PDE, not shown) reduce the page table size by 50%, they have similar performance as GT-PDE. We demonstrate the performance advantage of a smaller page table size in Section 5.4.

In our default configuration, the L2 cache is the highest level to cache the page table. We also evaluated the configuration that PTEs can be cached in the L1 cache. For the workloads that we studied, the performance change is very small (< 0.1%) compared to our default configuration for the traditional 4KB pages. On the other hand, there is extra 1% performance gain on average if GT-PDEs can be cached in the L1 cache instead of only in the L2 cache.

### 5.3. Memory Capacity Used as Superpages

Figure 14 shows *superpage percentage*, that is, the percentage of memory capacity that can be used as superpages using different page table formats. Using more superpages is beneficial due to the speedup achieved (recall that superpages do not need to traverse the last level of page tables). Note that the remaining non-retired memory pages can still be used and mapped using traditional 4KB-page PDEs. For traditional 2MB superpages, the percentage quickly drops to 0%. For GT-PDEs, the superpage percentage to utilize GT-PDEs drops to 50% with increased retired pages, because each memory slice is likely to have at least one retired page. When there is a retired page, only 50% of B-blocks of a memory slice can be used in the GT-PDEs. As shown in the figure, to have 50% superpage percentage, the thresholds of retired pages should be 1.4%, 2.8% and 5.5% for B-block sizes of 128KB, 64KB and 32KB, respectively. The superpage percentage of P-GT-PDE-2MB (paired approach) is bounded from below by GT-PDE-2MB and above by GT-PDE-1MB. Compared to GT-PDE-2MB, the smaller B-block size allows P-GT-PDE-2MB to tolerate more retired pages while maintaining the same page table size.

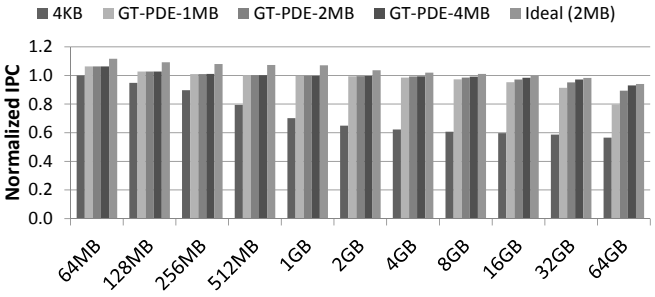



**Figure 15: IPC with different problem sizes normalized to a problem size of 64MB using traditional 4KB page.**

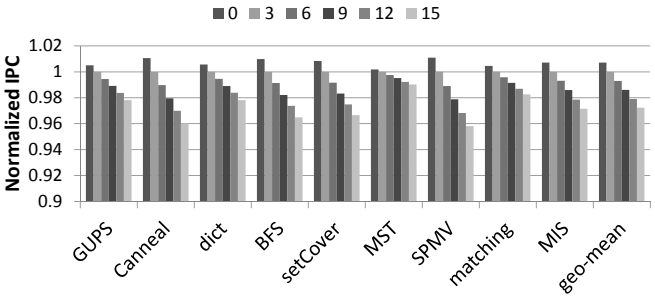



**Figure 16: IPC of GT-PDE-4MB with different translation latencies of GT-PDEs normalized to a default latency of 3 cycles.**

### 5.4. Sensitivity to Problem Size

Figure 15 shows IPC of *GUPS* with different problem sizes and page table formats. We choose to evaluate *GUPS* because it is a common benchmark in scalability studies, given that its memory footprint varies with problem size from 64MB to 64GB. As shown in the figure, the performance of traditional 4KB pages is sensitive to the problem size. When the problem size is increased from 64MB to 8GB or more, IPC reduces by 40% or more due to increased address translation overhead. On the other hand, the performance advantage of superpages is significantly increased with lager problem size. When the problem size is 8GB, the IPC of GT-PDE-4MB superpage is 63.4% better than 4KB page. The performance of GT-PDEs with different B-block sizes shows difference when the problem size is very large. When the problem size is 64GB, GT-PDE-4MB is 16.6% better than GT-PDE-1MB because of the smaller size of the page table. Since the major advantage of P-GT-PDE is to reduce the size of the page table, this also implies that P-GT-PDE should only be used for processes with very large memory footprints.

### 5.5. Sensitivity to GT-PDE Address Translation Latency

Figure 16 shows the IPC of GT-PDE-4MB assuming different translation latencies of GT-PDEs. All results are normalized to the default 3-cycle extra latency. As shown in the figure, the performance overhead is mostly consistent among the workloads and is less than 1% if GT-PDE translation latency is 6 cycles instead of 3 cycles.

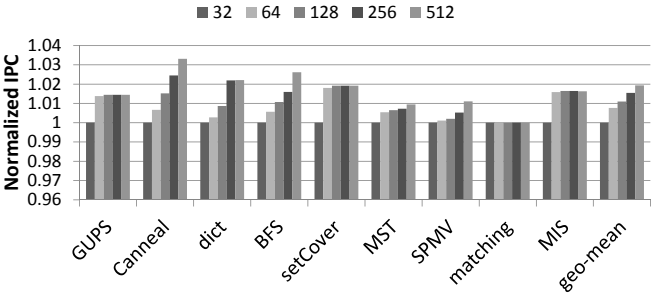



**Figure 17: IPC of GT-PDE-4MB with different PDE cache sizes normalized to 32-entry PDE cache.**

### 5.6. Sensitivity to GT-PDE Cache Size

Figure 17 shows the IPC improvement of GT-PDE-4MB with a larger PDE cache. The results are normalized to the 32-entry PDE cache baseline. As shown in the figure, the performance is not very sensitive to the size of the PDE cache (the range of the Y-axis is quite small); 32 or 64 entries are enough for the PDE cache. In fact, the maximum improvement of making cache sizes much larger is less by approximately 2% on average (see last column of the figure).

### 5.7. Comparing to TLB Coalescing

TLB coalescing is a technique which can substantially increase TLB reach by coalescing multiple adjacent PTEs into a single TLB [20, 19]. Note that for our target workloads, that is, those with large memory footprints, the gain of TLB coalescing is not as significant as for smaller applications.

Figure 18 shows the performance comparison between TLB coalescing and GT-PDE. We evaluated the configuration that the 512-entry L2 TLB supports 8x and 32x TLB coalescing, which merges adjacent 8 PTEs and 32 PTEs, respectively. In order to avoid favoring our own scheme, we assume no extra CPU cycles to load L2 TLB entry with TLB coalescing. As shown in the figure, the performance gain with TLB coalescing is virtually nonexistent because TLB reach is still limited even with 32x TLB coalescing considering workloads with large memory footprints. For address translations that are missed in the TLB, the dominant performance overhead is from accessing PTEs. To avoid address translation becoming a performance bottleneck, it is critical to eliminate PTE accesses by supporting superpages.

## 6. Related Work

Both software and hardware changes are necessary to support superpages. Talluri et al. discussed the tradeoffs and challenges to support superpages in hardware [28]. Ganapathy and Schimmel described possible ways to support superpages in the OS [11]. Navarro et al. described a design to transparently support superpages in the OS [18]. Zhang et al. described a design to map superpages to disjoint physical pages using traditional base page table format [32]. In their proposed design, page table still needs to be accessed when

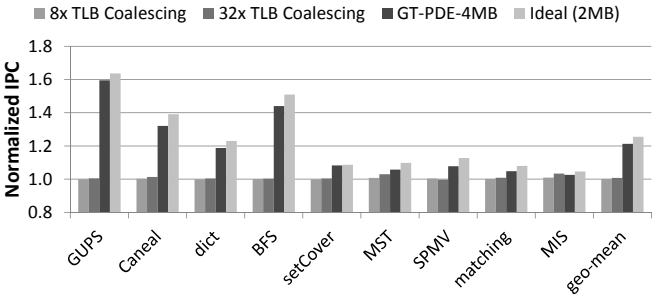



**Figure 18: IPC of TLB coalescing and GT-PDE normalized to traditional 4KB page baseline.**

there is a cache miss. To the best of our knowledge, this is the first work to propose a new storage-efficient superpage format designed for memory with retired pages.

There are much work on improving TLB performance. TLB hit rate can be improved by sharing TLB entries among CPU cores [27, 30, 5]. TLB miss penalty can be reduced by prefetching [7]. Recently, TLB coalescing has been studied to improve TLB reach [19, 20]. Similar to TLB coalescing, MMU cache coalescing has been proposed to reduce TLB miss penalty [6]. Our work does not require any changes to TLB and is orthogonal to the work proposed for TLB performance improvement. For workloads with large memory footprints, improving the TLB performance alone is not enough to solve the problem.

Memory errors can be tolerated using managed runtime systems [13], but this requires the program to be written in managed code (e.g., Java). Our work can be used for both managed code and unmanaged code. Gandhi et al. described an escape filter design which handles a total of 16 retired pages with a 256-bit on-chip bloom filter [12]. GTSM is designed to tolerate significantly more retired pages (e.g., 1.6%, or 2GB of retired pages in a 128GB main memory).

## 7. Conclusion

Superpages are critical for workloads with large memory footprints. Traditional 2MB superpages are not suitable for memory with retired pages, because a superpage must be mapped to large contiguous physical memory. We proposed gap-tolerant sequential mapping (GTSM) to allow mapping a superpage to memory with retired pages. We proposed GT-PDE which has a block selection bitmap to support GTSM. We also proposed P-GT-PDE, a variant of GT-PDE, which can reduce the size of the page table by 50%. Our evaluation shows that the performance of GT-PDE and P-GT-PDE is close to the ideal 2MB superpaging (i.e., with no retired pages). For large-footprint workloads, the 4MB-page GT-PDE achieves 96.8% of traditional 2MB superpaging, while tolerating memory faults.

## References


[1] "Linux memory page offlining," 2009, https://www.kernel.org/doc.

[2] S. Baek, S. Cho, and R. Melhem, "Refresh now and then," in *IEEE TC*, 2013.

[3] T. W. Barr, A. L. Cox, and S. Rixner, "Translation caching: Skip, don't walk (the page table)," in *ISCA*, 2010.

[4] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," in *ISCA*, 2013.

[5] A. Bhattacharjee, D. Lustig, and M. Martonosi, "Shared last-level tlbs for chip multiprocessors," in *HPCA*, 2011.

[6] A. Bhattacharjee, "Large-reach memory management unit caches," in *MICRO*, 2013.

[7] A. Bhattacharjee and M. Martonosi, "Inter-core cooperative tlb for chip multiprocessors," in *ASPLOS*, 2010.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[9] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *ICDM*, 2004.

[10] J. Dongarra and P. Luszczek., "Introduction to the hpc challenge benchmark suite." http://icl.cs.utk.edu/hpcc/pubs/

[11] N. Ganapathy and C. Schimmel, "General purpose operating system support for multiple page sizes," in *USENIX ATC*, 1998.

[12] J. Gandhi, A. Basu, M. M. Swift, and M. D. Hill, "Efficient memory virtualization," in *MICRO*, 2014.

[13] T. Gao, K. Strauss, S. M. Blackburn, K. S. McKinley, D. Burger, and J. Larus, "Using managed runtime systems to tolerate holes in wearable memories," in *PLDI*, 2013.

[14] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design," in *ASPLOS*, 2012.

[15] Intel, "Intel 64 and ia-32 architectures developer's manual." 1997.

[16] W. Korn and M. S. Chang, "Spec cpu2006 sensitivity to memory page sizes," *SIGARCH CAN*, vol. 35, no. 1, 2007.

[17] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *ISCA*, 2012.

[18] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, 2002.

[19] B. Pham, A. Bhattacharjee, Y. Eckert, and G. Loh, "Increasing tlb reach by exploiting clustering in page translations," in *HPCA*, 2014.

[20] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "Colt: Coalesced large-reach tlbs," in *MICRO*, 2012.

[21] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *MICRO*, 2009.

[22] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009.

[23] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *ICS*, 2011.

[24] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *ISCA*, 2010.

[25] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: A large-scale field study," in *SIGMETRICS*, 2009.

[26] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan, "Brief announcement: The problem based benchmark suite," in *SPAA*, 2012.

[27] S. Srikantaiah and M. Kandemir, "Synergistic tlbs for high performance address translation in chip multiprocessors," in *MICRO*, 2010.

[28] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson, "Tradeoffs in supporting two page sizes," in *ISCA*, 1992.

[29] D. Tang, P. Carruthers, Z. Totari, and M. W. Shapiro, "Assessment of the effect of memory page retirement on system ras against hardware faults," in *DSN*, 2006.

[30] C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramirez, A. Mendelson, N. Navarro, A. Cristal, and O. Unsal, "Didi: Mitigating the performance impact of tlb shootdowns using a shared tlb directory," in *PACT*, 2011.

[31] M. T. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *ISPASS*, 2007.

[32] L. Zhang, E. Speight, R. Rajamony, and J. Lin, "Enigma: Architectural and operating system support for reducing the impact of address translation," in *ICS*, 2010.

[33] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009.