Herniated Hash Tables: Exploiting Multi-Level Phase Change Memory for In-Place Data Expansion

Zhaoxia Deng, Lunkai Zhang, Diana Franklin, Frederic T. Chong
University of California, Santa Barbara
{zhaoxia, lunkai.zhang, franklin, chong}@cs.ucsb.edu

ABSTRACT
Hash tables are a commonly used data structure used in many algorithms and applications. As applications and data scale, the efficient implementation of hash tables becomes increasingly important and challenging. In particular, memory capacity becomes increasingly important and entries can become asymmetrically chained across hash buckets. This chaining prevents two forms of parallelism: memory-level parallelism (allowing multiple prefetch requests to overlap) and memory-computation parallelism (allowing computation to overlap memory operations). We propose, herniated hash tables, a technique that exploits multi-level phase change memory (PCM) storage to expand storage at each hash bucket and increase parallelism without increasing physical space.

The technique works by increasing the number of bits stored within the same resistance range of an individual PCM cell. We pack more data into the same bit by decreasing noise margins, and we pay for this higher density with higher latency reads and writes that resolve the more accurate resistance values. Furthermore, our organization, coupled with an addressing and prefetching scheme, increases memory parallelism of the herniated datastructure.

We simulate our system with a variety of hash table applications and evaluate the density and performance benefits in comparison to a number of baseline systems. Compared with conventional chained hash tables on single-level PCM, herniated hash tables can achieve 4.8x density on a 4-level PCM while achieving up to 67% performance improvement.

CCS Concepts
• Computer systems organization → Architectures;
• Hardware → Non-volatile memory;

Categories and Subject Descriptors
[Hardware]: Memory architecture, multi-level PCM; [Data structure]: Data structure storage, hash tables

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MemSys ’15 October 5–8, 2015, Washington, DC, USA
© 2015 ACM. ISBN 123-4567-24-5670/08/06. . . $15.00
doi: 10.475/123_4

1. INTRODUCTION
Compared with DRAM, phase-change memory (PCM) shows a number of promising attributes, including higher density, non-volatility and low static power consumption. Therefore, it is considered a possible substitute for the DRAM technique. Another interesting attribute of PCM is that a single PCM cell can be used to store multiple bits in an asymmetric manner. The deeper bits (less-significant bits) need more time to be read out and much more time to be updated than shallower bits (more-significant bits). However, for the long read latency, it is possible to develop a prefetching mechanism that manages to prefetch out the deep-level data from PCM before their usage with a step-wise multi-level read operation. For the long write latency, Qureshi et al. [17] have proposed schemes such as write cancellation and write pausing to avoid the delay of response to the following read requests. Therefore, the overall system needs not to suffer from the long access latency of deep-level data.
Conventional hash tables usually use chained structures to save memory space. The addresses of entries are allocated dynamically and distributed randomly in memory, which prevents prefetching and instruction-level parallelism. Moreover, with multi-level PCM, the performance of conventional hash tables will decrease dramatically because of the high read and write latency. In this paper, we introduce the herniated hash table, which exploits multi-level PCM and provides dense and efficient storage with support of dynamic in-place expansion and non-uniform growth of the hash tables. By designing an addressing scheme for the multi-level PCM and using multi-level PCM aware hash table structures and prefetching schemes, the herniated hash table performs better than the conventional hash table design from the following two perspectives: On one hand, the herniated hash table yields similar performance with less storage compared with the conventional hash table on single-level PCM; On the other hand, the herniated hash table achieves much better performance with similar storage overhead compared with the conventional hash table on multi-level PCM.

The rest of the paper is organized as follows. We first provide background information on multi-level PCM and its operations in Section 2. Then, Section 3 presents our herniated hash table design. Section 4 describes our experimental methodology, and Section 5 presents our results. Finally, Section 6 discusses related work, and Section 7 draws our conclusions.

2. BACKGROUND OF MULTI-LEVEL PCM

Phase-change memory (PCM) is one type of resistive memory, the resistance of whose cells can be changed in a wide range. An analog-to-digital circuit (ADC) reads the resistance as a digital value. Depending on the ADC setting, the same resistance of a PCM cell could be interpreted as a different number of bits. Alichar et al. [1] showed that a single resistive memory cell can be used to store up to seven bits of data. Figure 1 shows an example of a multi-level PCM cell: the same resistance of the cell can be interpreted as b0 when the ADC is using one-level accuracy and as b00 and b001, respectively, when the ADC is using two-level and three-level accuracy. This precision comes with a cost. Read latency increases exponentially with the number of bits stored in the PCM cell and write latency is even higher than that. However, as shown in Figure 1, multi-level PCM read can be operated in a step-wise approximate manner: that is, after reading a shallow bit of a PCM cell, the read operation can be continued to further read out its deeper level bits. Therefore, if the data are carefully organized in multi-level PCM, we can use this step-wise read attribute to develop an efficient prefetching mechanism.

Although write latency of deep-level PCM accesses is also very high, we assume that our target applications have more read accesses than write accesses to the hash tables. Moreover, as our scheme enables both instruction-level parallelism and memory-computation parallelism, the write latency (which is usually caused by the write back requests from the last level cache) doesn’t have significant effects on the critical path of the processor’s pipeline. If the write latency is too long that the following read requests are affected, write cancellation and write pausing techniques [17] will help by prioritizing the following read requests to remove potential pipeline stalls. We use the conservative configuration with the write cancellation technique in all of our systems that run on the multi-level PCM. Better combinations of write cancellation and write pausing are possible according to [17]. We’ll also discuss the case without write cancellation in Section 5.

In this paper, similar to previous work [25], we have an exponential latency model for multi-level PCM read accesses (the exponential base is 2.0) and a higher latency model for multi-level PCM write accesses: as shown in Figure 2, a PCM read takes 120 ns, 240 ns, 480 ns, and 960 ns respectively, and a PCM write takes 150 ns, 600 ns, 1800 ns, 4800 ns respectively for one, two, three, and four levels.

3. HERNIATED HASH TABLES

In this section, we firstly explain the inefficiency of the conventional hash table design on multi-level PCM and present the design of the herniated hash table (HHT), with emphasis on the HHT’s differences from a conventional hash table on the same PCM system. Then we introduce a new addressing scheme for the multi-level PCM system since traditional virtual to physical address mapping schemes are not efficient in multi-level PCM. Based on that, we design multiple prefetching schemes to improve the performance of memory accesses.

The conventional hash table is implemented with an array of linked lists. The length of the array is the number of buckets. Each array entry is an address that points to the linked list of hash entries that fall into the same bucket.

The single-level PCM system behaves similarly as traditional DRAM. Without loss of generality, we assume the single-level PCM capacity is 1 GB. The multi-level PCM system is assumed to be four levels, with 0–1 GB on the first level, 1–2 GB on the second level, 2–3 GB on the third level, and 3–4 GB on the fourth level. Multi-level PCM systems with more levels will also be analyzed in the sensitivity studies.

3.1 The Basic Herniated Hash Table Design

Figure 3 shows a snapshot of the organizations of both the conventional hash table and the HHT. Both of them contain N buckets.

The conventional hash table, as shown in Figure 3(a), starts with an array of N pointers (i.e., head pointers), and each head pointer points to the first valid entry of the corresponding hash bucket. Each conventional hash table entry contains three fields: key, which identifies the entry; value, which contains the information for the entry; and next entry ptr, which points to the next entry in the same bucket. This conventional implementation is not very efficient on a multi-level PCM. The main problem is that, since it is not multi-level PCM-aware, both the head pointers and entries can be placed randomly at any level in PCM, and therefore may need a long latency to access if they happen to be in deep levels. Another intrinsic problem of the conventional hash table is that it uses pointers to link the entries and the addresses of entries are dynamically allocated and randomly distributed in the physical memory address space. It is hard to either parallelize the computation and data retrieving because we don’t know the address of the next entry beforehand, or use a prefetching mechanism to retrieve the entries faster due to the lack of spatial locality in memory.

In Figure 3(b), we show the organization of the HHT. At first, the HHT only uses the first level of the PCM, and it initially contains an array of N hash entries (i.e., the first hash entry of each bucket). Whenever possible, the HHT
always puts the entries from the same bucket in the same PCM cells but in different levels (e.g., entries $A_1$ to $A_3$). If the current cells have already reached their level limit (in our case, four levels), new PCM cells will be allocated to store extra hash entries. The deepest level of the previous PCM cells will be transferred to a redirect pointer, which points to the newly allocated space at the first PCM level (e.g., entry $Q_4$ in the figure). This data organization creates spacial locality in memory and removes most of the links that blocks instruction-level parallelism. In general, compared with the conventional design, the benefits of the HHT are three-fold:

- **More Compact Hash Table Structure.** As shown in Figure 3, each entry in the HHT has only two fields, **key** and **value**, and it does not need a pointer field (like **next entry ptr** in a conventional hash entry). Moreover, the HHT does not need a separate array of **head pointers**, which is also needed in the conventional hash table. The HHT, however, has two storage drawbacks. First, it has to pre-allocate a hash entry for every bucket (even if a bucket is never used at all). Second, if there are too many entries in one bucket, then additional storage is needed for redirect pointers. However, we shall see in Section 5 that the storage merits of HHTs usually outweigh their drawbacks.

- **Shorter Latency to Access Buckets with Few Entries.** HHTs always insert a new hash table entry into the shallowest unused level. Therefore, for a hash bucket with few entries (i.e., 1-2 entries), only the shallow PCM levels are used. As a result, accessing these entries in these buckets will have a relatively short latency. Figure 4(a) shows such a situation: in the conventional hash table, entries $A_1$ and $A_2$ are buried in deeper levels (levels 4 and 3, respectively), whereas, in the HHT, these two entries are deliberately placed in the shallowest levels (levels 1 and 2, respectively). As a result, it takes less time to access entry $A_2$ in the HHT.

Note that, when accessing an entry in a bucket with...
many entries, the HHT without prefetching may take a longer time than the conventional hash table, as shown in Figure 4(b). In this case the HHT also needs to use deeper PCM levels, and it takes a long time to access the entries. Additionally, the HHT needs extra time to access redirect pointers. We improve the system performance in this situation with prefetching, which will be explained later.

- **Ability to Prefetch Entries.** In a hash table lookup function, to find the matched entry, the function needs to traverse through the hash entries of the corresponding bucket. In a conventional hash table, since there is no special relationship among the addresses of successive entries (because they are linked by pointers), prefetching cannot be accomplished without complex pointer-tracing hardware. The HHT, however, stores several entries contiguously for each bucket, which enables prefetching. With step-wise reading, we can get multiple entries ready with exponential latency. The prefetching is very effective for hash tables because the access pattern tends to be traversing all the entries in the same bucket. We will discuss the prefetching schemes for the HHT in detail in Section 3.3.

### 3.2 Addressing HHTs

A critical challenge in exploiting multi-level PCM lies in how the data will be addressed and stored in the cache hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate physical page for a given page of physical PCM cells. Correspondingly, four-times (for four-level PCM) of the virtual physical page for a given page of physical PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate hierarchy.

HHTs exploit a multi-level PCM aware addressing scheme. The HHT tries to store the hash entries belonging to the same bucket in different levels of the same PCM cells. In addressing, this corresponds to different virtual pages with the same offset. Only the most significant address bits differ in addresses for different virtual pages. In this manner, successive HHT entries in the same bucket can be directly addressed by software by using the corresponding addresses in a sequence of physical pages. We don’t need to increase storage for the page table since the addressing of different virtual pages for the same hash bucket follows a sequential pattern.

This scheme requires us to pay for potential hash table growth in terms of physical address, but allows us to defer allocating physical space in the HHT as needed for each hash table entry. For example, an HHT based upon four-level PCM would be addressed by four-times the physical addresses, but it would begin with only one-time the actual storage and grow as needed as the number of hash entries increases in one bucket.

An important advantage of this physical addressing scheme is that it allows multi-level PCM to be seamlessly integrated into the hash hierarchy. Data at multiple levels can be easily fetched (and prefetched) into caches. To avoid cache conflicts, we sequentially add a shift: the cache block size to the addresses of different virtual pages for the same hash bucket so that entries in this bucket will fall into different cache sets.

### 3.3 Prefetching Schemes for Herniated Hash Tables

The prefetching schemes for the HHT are based on two observations. First, the hash entries in the same bucket tend to be placed in the same PCM cells. Second, to read out a deep entry, the shallower entries in the same PCM cells (e.g., entries A_1 and A_2) need to be read out first. Therefore, the basic idea of prefetching schemes in the HHT is that, when accessing a shallow-level entry, we perform a deep-level read that reads out all the entries on the corresponding memory cells and buffers them for future use. Figure 4 illustrates the
On-Chip vs. Off-Chip. In a modern processor, the memory controllers are usually implemented on-chip and the main memory (PCM in our case) is placed off-chip. The interface between these two components involves long latency and limited bandwidth. If the herniated prefetcher is implemented on-chip, as shown in Figures 5(b) and 5(c), the processor can get the prefetched data from the HHT with relatively low latency, but this will require higher memory bandwidth to transfer all the prefetched data from off-chip memory. If the prefetcher is implemented off-chip, as shown in Figures 5(a), the situation is just the opposite—there is no need for additional memory bandwidth, but the processor may suffer longer latency to access the prefetched data.

Buffering vs. Bufferless. A prefetcher often comes with an additional buffer to store the prefetched data, as shown in Figures 5(a) and 5(b). Some recent commercial processors, however, use bufferless prefetchers, which directly prefetch the data into the last level cache (LLC). Compared with buffering, the bufferless prefetcher reduces on-chip storage cost. Bufferless options, however, may cause cache pollution when prefetching the wrong data or the right data at the wrong time.

Figure 5 shows three possible prefetching schemes for the HHT, namely (a) an off-chip prefetcher that prefetches data to an off-chip buffer; (b) an on-chip prefetcher that prefetches data to an on-chip buffer, and (c) a LLC prefetcher that prefetches data to LLC. In Section 5, we will quantitatively compare these prefetching schemes.

4. METHODOLOGY

We simulate our HHT systems on gem5 [3], a detailed, event-driven system simulator. We used the system emulation (SE) mode, as it is more efficient and it also supports simple address translation in the TLB, which helps simulate our addressing scheme. Section 4.1 explains the system simulation parameters in detail. Section 4.2 describes multiple benchmark kernels to drive our evaluation. Finally, Section 4.3 discusses several baseline systems that will be compared to our HHT systems.

4.1 Simulation Configuration

The parameters of the processor and memory system used in our experiments are listed in Table 1. The processor configuration is similar to the ARM Cortex A9 single core processor [5], which has a superscalar pipeline that forwards eight instructions per cycle to the decoder. Two levels of caches are used with this processor. As the HHT system uses PCM, we simulate a multi-level PCM and a single level PCM (for base line systems) on the nvmain [16] memory simulator, which supports modeling of realistic timing and queuing delay of multi-level PCM. For single-level PCM accesses, the memory latency includes the memory controller latency, memory bus transmission latency, and PCM read latency. For the multi-level PCM, the simulator models exponential read latency as more bits are read out. All data levels are available after the highest level is read out. For write requests to the multi-level PCM, the simulator needs to read out all existing bits first, modify the bits to be updated, and write all the bits back to the PCM cells in the end. Table 1 lists the timing configuration in detail. For most of our experiments, we used four-level PCM for the multi-level PCM, but we will also discuss using more levels in Section 5.4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2 GHz, single core, out-of-order, alpha ISA, 8-issue width, 64-byte cacheline size</td>
</tr>
<tr>
<td>L1 $</td>
<td>32 KB I/D-cache, 4-way, 2-cycle latency, 6 MSHR</td>
</tr>
<tr>
<td>L2 $</td>
<td>512 KB, 8-way, 12-cycle latency, 16 MSHR</td>
</tr>
<tr>
<td>Memory Controller</td>
<td>10 ns latency</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB; 1 channel, 1 rank and 4 banks; Row buffer size 1KB</td>
</tr>
</tbody>
</table>

Table 1: Simulation parameters, similar to ARM Cortex A9 single-core processor
Table 2: Hashtable benchmark workload characterization

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Number of Hash Entries</th>
<th>Number of Lookups</th>
<th>Simulated Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Routing</td>
<td>50,000</td>
<td>2,050,000</td>
<td>14.5</td>
</tr>
<tr>
<td>Wordcount</td>
<td>44,998</td>
<td>2,000,000</td>
<td>6.7</td>
</tr>
<tr>
<td>Dedup</td>
<td>45,941</td>
<td>1,139,905</td>
<td>0.82</td>
</tr>
</tbody>
</table>

the Wordcount benchmark. With fewer buckets, the average number of entries in the bucket increases and the bucket size becomes more nonuniform. In this situation, it is more challenging to efficiently store the hash table and traverse all entries in each bucket at the same time. Furthermore, we take snapshots of the hash table bucket length distributions in the middle of each benchmark’s execution. Figure 7 shows snapshots of the Wordcount benchmark with 1,024 buckets. These snapshots are taken in the middle of the program execution when 20%, 40%, 60%, and 80% of the total execution is finished respectively, to give a picture of the intermediate hash table structure when lookups happen (how many entries to traverse in each bucket). We’ll show that HHT can get both good storage efficiency and good performance when the bucket length scales dynamically.

Figure 6: Wordcount using different numbers of buckets

4.3 Baseline Systems

We compare our HHT schemes with several baseline systems with different software and hardware implementations. We list the baseline systems and their difference in software implementation, hardware storage, and latency models in Table 3.

For chained hash tables (CHTs), we use a common implementation based on linked lists. The hash table maintains an array of linked lists, each of which stores hash entries that fall into the same bucket. Each linked list node contains three elements: the key, the value, and the pointer to the next entry in the same bucket. Chained hash tables on the single-level PCM is similar to the traditional hash table implementations on DRAM. The single-level PCM access latency is the sum of memory control latency, memory bus latency, and memory read latency.

To increase the storage efficiency of hash tables, we implement chained hash tables on the multi-level PCM as a baseline system in the following way: hash entries are dynamically allocated and randomly distributed in the multi-level PCM address space. In this system, the memory latency model follows the exponential read and write timing model as shown in Figure 2.

As discussed in the previous sections, the chained hash tables prevent both memory-level and instruction memory-level parallelisms. In the herniated hash tables (HHT) implementation, each entry contains just two elements: the

Table 3: Hashtable system configuration

<table>
<thead>
<tr>
<th>Baseline systems</th>
<th>Hash Node Structure</th>
<th>Memory Latency Model</th>
<th>Multi-level Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHT on Single-level PCM</td>
<td>key, value, pointer to next</td>
<td>single-level</td>
<td>No</td>
</tr>
<tr>
<td>CHT on Multi-level PCM</td>
<td>key, value, pointer to next</td>
<td>multi-level</td>
<td>Yes</td>
</tr>
<tr>
<td>HHT without Prefetching</td>
<td>key, value</td>
<td>multi-level</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 7: Snapshots of Wordcount with 1,024 buckets
key and the value. We use an array to maintain hash entries in the same bucket. There are no links between entries except when the number of collisions in one bucket exceeds the maximum level of the PCM. Taking an HHT implementation without prefetching as a baseline, we can see how much overhead is caused by the exponential latency model of the multi-level PCM and the effectiveness of prefetching schemes to tolerate this latency.

5. RESULTS

In this section, we present evaluation results on HHTs, focusing on several issues.

First, storage efficiency analysis results are explained in Section 5.1, comparing HHTs with CHTs on single-level PCM and multi-level PCM, respectively.

Second, we present the performance analysis of HHTs in all benchmarks in terms of execution time and IPC (instructions per cycle). We also study the impact of the write cancellation technique and present the performance results with and without write cancellation.

Third, the impact of different prefetching schemes discussed in earlier sections is analyzed, in terms of execution time, LLC miss rate, average LLC miss latency and the memory traffic.

Finally, we explore the sensitivity of systems to using different numbers of buckets and deeper levels of PCM. All of the normalized metrics take HHTs without prefetching as the baseline.

5.1 Storage Efficiency Analysis

We define storage efficiency as the number of used memory cells here. Let \( N \) be the number of buckets in our hash tables and \( M \) be the number of unique keys in the workload. A HHT entry takes \( 16B \) and a CHT node takes \( 24B \). HHTs use the space of the array of head entries plus the space for redirected entries, which equals \((N + \text{redirect pointers}) \times 16B\). CHTs on the single-level PCM takes the array of head entry pointers plus the actual storage of all entries, which is \((N \times 8B + M \times 24B)\). Chained hash tables on the four-levels PCM has four times the address space of the single-level PCM, so it takes \(1/4\) of the storage as on the single-level PCM.

Figure 8 shows the normalized storage efficiency results for HHTs and CHTs with 2048 buckets on the single-level PCM and on the multi-level PCM in three benchmarks. The HHTs get around 4.8x the benefits of the CHTs on the single-level PCM and 1.14x the benefits of CHTs on the multi-level PCM. When \( M \) is much bigger than \( N \), HHTs have significant density benefits. We will discuss situations when \( M \) is close to \( N \) in the sensitivity study in Section 5.3.1.

5.2 Performance Analysis

5.2.1 Instructions per Cycle

For a given processor, the number of instructions per cycle (IPC) reflects the instruction-level parallelism that different systems can achieve. As discussed in the previous sections, herniated hash tables enable both instruction-level parallelism and instruction memory-level parallelism, so we can see substantial improvements in IPC.

Figure 9 plots the normalized IPC of three benchmarks with different hash table systems. We use 2048 buckets for all hash table systems to simplify the comparison. Performance with different number of buckets is shown in the sensitivity study in Section 5.3.2. The normalized IPC results in Figure 9 show that HHTs can achieve a 60% improvement over CHTs on the single-level PCM and about a 4x improvement over the CHTs on the multi-level PCM in terms of IPC. HHTs with LLC prefetching can achieve up to 1.66x improvement over HHTs without prefetching.

5.2.2 Execution Time

We also plot the execution time of benchmarks with different hash table systems to conduct a general performance evaluation. As shown in Figure 10, HHTs with LLC prefetching are up to 1.66x faster than HHTs without prefetching and more than 2.4x faster than the CHTs on the multi-level PCM. Compared with the CHTs on the single-level PCM, The HHTs with prefetching are 44% faster while achieving significant density benefits, as indicated in Section 5.1.

5.2.3 Write cancellation

Write cancellation [17] is an existing work to reduce read latency by canceling the already scheduled write requests if a read request arrives to the same bank within a period. As the write latency of multi-level PCM is very high, the write requests could significantly increase the effective latency of later arriving read requests. In most of our experiments, we use write cancellation as a built-in technique in multi-level PCM. In this section, we study the performance of our hash table systems if using multi-level PCM without write cancellation.

Figure 11 and Figure 12 show the IPC and execution time results without write cancellation. For the Internet routing and Dedup benchmarks, IPC and execution time have no much difference using multi-level PCM with or without write cancellation because the read requests are much more than write requests. However, in the Wordcount benchmark, the number of write requests is similar to the number of read requests. The long write latency may cause a decrease in the system performance. From Figure 11, HHTs with LLC prefetching still achieve higher IPC than CHTs on single-level PCM but the improvement decreases from 60% to 52%.

From Figure 12, the correspondent speedup of HHTs with LLC prefetching over CHTs on single-level PCM decreases from 44% to 27% for the Wordcount benchmark.

5.3 Sensitivity to Different Hash Table Configurations

![Figure 8: Storage efficiency of HHTs](image-url)
Hash table behavior varies considerably when we configure them with different numbers of buckets, as shown in the workload characterization histograms. When fewer buckets are used, the number of collisions in each bucket increases. In this part of the experiments, we study the scalability of hash tables in both storage efficiency and performance in tolerating the increased number of collisions. We choose one representative benchmark, Wordcount, and study the hash table behavior with different numbers of buckets.

### 5.3.1 Storage
Varying the number of buckets can have the following effects on the storage. On the one hand, HHTs need more redirect pointers when fewer buckets are used while CHTs don’t have this storage overhead. On the other hand, each hash table needs to keep an array of bucket head entries that is linear with respect to the number of buckets. In CHTs, this array stores just pointers (eight bytes) to the head entries. In the HHTs, this array stores real keys and values (sixteen bytes). When the number of buckets is very large, there are many bucket head entries to store, even though some of the buckets are empty. HHTs have higher storage overhead for these head entries than CHTs. However, these two effects are in the opposite directions. Generally considering them together results in little difference in storage efficiency when the number of buckets changes, as shown in Figure 13. The HHTs are more storage efficient in all settings compared with CHTs on both the single-level PCM and the multi-level PCM.

### 5.3.2 Performance
Figure 14 shows the performance in terms of both execution time and IPC of Wordcount with different hash table systems and different numbers of buckets. For 8192 buckets, the HHTs with LLC prefetching are 12%, 59.4%, and 5.5x faster than CHTs on the single-level PCM, HHTs without prefetching, and CHTs on the multi-level PCM respectively. When the number of buckets decreases, the number of collisions starts to increase. The HHTs can tolerate more collisions because there are more prefetching opportunities. The CHTs cannot prefetch because of the dependencies between data accesses in the chained structure, so their performance degrades more quickly compared with the HHTs. For 1,024 buckets, HHTs with LLC prefetching show much better performance compared with CHTs on the single-level PCM by achieving 1.87x IPC and 49% speedup in execution time.

### 5.4 Sensitivity to PCM Levels
When the PCM has more levels, herniated hash tables can yield more benefits by deeper prefetching and more overlap between computation and memory accesses. However, since latency is exponential with respect to the depth of the level accessed, there is a significant overhead when the highest level is accessed from PCM. In this section we study the performance of HHTs when using PCM with more levels.

Figure 15 shows the execution time of hash table systems using 4 levels, 5 levels, and 6 levels PCM respectively. Comparing the execution time of HHTs with and without prefetching with different levels, we can see that the con-
distribution of the prefetching scheme increases by 12% from 4 levels to 5 levels because of more prefetching opportunities. However, the prefetching is less effective with 6 levels because the prefetching of one highest level entry will increase system latency significantly if the prefetched data is not used. In this situation, CHTs on the single-level PCM can get the best performance, but without density benefit. One future work is to study how to utilize different amount of levels in PCM dynamically for HHTs to yield the best performance for different applications while getting the density benefit as well.

5.5 Impact of Different Prefetching Schemes

We discussed three possible prefetching schemes for herminated hash tables: off-chip prefetcher, on-chip prefetcher, and LLC prefetcher. These three options were motivated by the idea that we want to prefetch the data close to the processor to maximize the parallelism between instructions and memory. The off-chip prefetcher operates alongside the memory row buffer, which reads out all levels of data iteratively on a low-level access. This prefetcher saves the memory read latency in the multi-level PCM. The on-chip prefetcher prefetches all levels of data to an on-chip buffer in the memory controller. It saves the transmission latency on the memory bus at the expense of a little more memory traffic. The LLC prefetcher moves the prefetched data into the last-level cache. It is the closest to the processor but may cause the cache pollution problem. Figure 16 shows the experimental results of the prefetchers, with 2,048-bucket hash tables.

We evaluate three benchmarks with all prefetching schemes. The HHTs without prefetching are taken as the baseline. From the execution time and IPC results as shown in Figure 16, we can see that the LLC prefetcher exhibits the best performance in general. The main reason is shown in the LLC miss rate result in Figure 17. With the LLC prefetcher, the LLC miss rate in the HHTs is reduced by about 6%, while the LLC average miss latency is similar to that of the other two prefetchers. Both the LLC prefetcher and the on-
A great deal of work has been done to optimize the performance, storage, and scalability of hash tables. The use of concurrent hash tables [15] is popular which are featured with lock-free extensive hash tables enabled by techniques such as “recursive split ordering” [21]. There has been other work to utilize techniques such as the read-copy update mechanism [23] or software transactional memory [4] to achieve better performance, scalability, or both. Additionally, there has also been work focusing on novel hash tables, such cuckoo hash [14], for multiple concurrent writers and readers.

7. CONCLUSIONS

Datastructure accelerations are of great importance in the big data era. Hashtables are a widely-used datastructure due to their average constant complexity for accesses. However, when more and more collisions occur, the traditional chaining implementation prevents two kinds of parallelism: memory-level parallelism and instruction-memory level parallelism. Multi-level PCM is an emerging memory technique that may be an alternative of DRAM, as it brings significant density benefits by supporting multiple resistance levels at each cell. In this paper, we introduced Herniated Hash Tables, which utilizes the multi-level PCM to store multiple hash collisions in the same bucket at the same PCM cells. However, the density benefits of multilevel PCM come with the exponential latency overhead when reading more than 1 bit per cell. To avoid this latency, we propose three prefetching options: Off-Chip prefetcher, On-Chip prefetcher and LLC prefetcher. Experimental results show that Herniated Hashtables can get up to 4.8x density benefits while achieving up 67% performance speedup over traditional chained hash tables on single-level PCM.

8. ACKNOWLEDGMENTS

We acknowledge support from the Center for Scientific Computing from the CNSI MRL: an NSF MRSEC (DMR-1121053) and Hewlett Packard.

9. REFERENCES


