Logging in Persistent Memory: to Cache, or Not to Cache?

Mengjie Li, Matheus Ogleari, Jishen Zhao
Persistent Memory

These nonvolatile devices are able to retain the data in a consistent state in case of power loss.
Logging in Persistent Memory

Update persistent memory with transactions

```
Tx_begin
  do some reads
  do some computation

write C

Tx_commit
```

Micro-ops:
- store C'₁
- store C'₂
- ...

---

Transaction: 
```
Rlog(addr(C), new_val(C))
memory_barrier
write C
```

---

Time
---
To cache, or not cache? That is the question.

[Mengjie Li+, Memsys 2017]
Experimental Setup

- **Desktop** – Dell OptiPlex 7040 Tower
  - CPU – 4-core 3.4GHz Intel Core-i7
  - Cache – 8 MB last-level cache

- **Measurement Tools** – Perf & rdtsc

- **Micro-benchmarks** – run 20 times and report the average performance without initialization time
  - Various working set sizes
  - Various transaction sizes and write intensity
  - Various data structures: hashtable, rbtree, array, …
Microbenchmarks Example

//initialization
Create an array of strings

//Uncacheable log
for (i = 0; i < array_size; ++i) {
    value = random_string;
    key = i;

    // Log updates
    // Intrinsic functions to invoke movnti
    _mm_stream_si32(&log[2 * i], key);
    _mm_stream_si32(&log[2 * i + 1], value);
    asm volatile ("sfence");

    array[i] = value;
}

//Cacheable log
for (i = 0; i < array_size; ++i) {
    value = random_string;
    key = i;

    // Log updates
    log[2 * i] = key;
    log[2 * i + 1] = value;
    asm volatile ("sfence");

    array[i] = value;
}
Issue with Cacheable log

Core

L1i Cache  L1d Cache

...  ...

Core

L1i Cache  L1d Cache  Log

Last-Level Cache

Log

Memory Bus

DRAM

NVM

Log

Cache pollution
LLC Miss Rate and Execution Time

![Chart showing LLC Miss Rate and Execution Time](image-url)
How about uncachable log performance?
How do we make log uncachable?

Example:
x86 processors provide uncachable write instructions (movnti, movntg, etc)
Instructions can be invoked by
• Inline functions (__asm__) 
• Intrinsic functions(_mm_stream_si32)
Write Combining Buffer (WCB)

- L1 Cache
- Core
- WCB
- Last-Level Cache
- Memory Bus
- DRAM
- NVM
- Log
- Core
- WCB
- Log

4-6 cache lines
Issues with Uncacheable Log

• Existing uncacheable writing schemes are sub-optimal
  o Partial writes in WCB
  o Overhead of uncacheable write instructions
  o Limited WCB size
Partial writes are inefficient, because they underutilize the memory bus bandwidth.
Execution Time vs. Transaction Size
— Partial Writes

Partial writes:
String Size – 4B
Iterations – 2097152
Total Data – 8MB

Full writes:
String Size – 64B
Iterations – 131072
Total Data – 8MB

<table>
<thead>
<tr>
<th>Execution Time</th>
<th>Partial Writes</th>
<th>Full Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncacheable</td>
<td>1.28E09 Cycles</td>
<td>1.15E08 Cycles</td>
</tr>
<tr>
<td>Cacheable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overhead of Uncacheable Write Instructions

Macrobenchmarks Example

// Uncacheable
for (i = 0; i < array_size; ++i) {
    value = random_string
    key = i;
    Log(string1)
    mmstream32(&key, 1, value);
    mmstream32(&key + i, value);
    array[i] = value
}

// Cacheable
for (i = 0; i < array_size; ++i) {
    value = random_string
    key = i;
    Log(string1)
    mmstream32(&key, 1, value);
    mmstream32(&key + i, value);
    array[i] = value
}

// Uncacheable
.L8:
    call rand
    movl %eax, %r12d
    call rand
    movq %rbx, %rdx
    addq Log(%rip), %rdx
    movnti %r12d, (%rdx)
    movq Log(%rip), %rdx
    movnti %eax, 4(%rdx, %rbx)
    sfence

// Cacheable
.L8:
    call rand
    movl %eax, %r12d
    call rand
    movq Log(%rip), %rdx
    movl %r12d, (%rdx, %rbx)
    movl %eax, 4(%rdx, %rbx)
    movq array(%rip), %rax
    addq $8, %rbx
    movl %r12d, (%rax, %rbp)
    addq $4, %rbp
    cmpq %r13, %rbx
    jne .L8
    sfence
More overhead to do type casting, if the type of data written is not integer

```c
void _mm_stream_si32(int *p, int a)
asm("movnti %1, %0" : "=m" (*p) : "r"(v)); // int * p, int v;
```
Issues with Limited WCB Size

Log updates among transactions issued by program

WCB

NVRAM bus
**Inefficiencies of Uncacheable Log**

<table>
<thead>
<tr>
<th>String size (Bytes)</th>
<th>iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2097152</td>
</tr>
<tr>
<td>8</td>
<td>1048576</td>
</tr>
<tr>
<td>16</td>
<td>524288</td>
</tr>
<tr>
<td>32</td>
<td>262144</td>
</tr>
<tr>
<td>64</td>
<td>131072</td>
</tr>
<tr>
<td>128</td>
<td>65536</td>
</tr>
<tr>
<td>256</td>
<td>32768</td>
</tr>
</tbody>
</table>

![Graph showing execution time and speedup for different string sizes](image-url)

- **Vertical Axis**: Execution Time (Billion cycles)
- **Horizontal Axis**: String size (Bytes)

Legend:
- **Blue**: uncacheable
- **Red**: cacheable
- **Black**: speedup

- **Graph Notes**:
  - Partial writes and sfence
  - WCB size limit

- **Speedup**:
  - 1.0
  - 1.2
  - 1.4
  - 1.6

---

---

---

---
Summary

• Tradeoff between cacheable and uncachable log
  o Issues with cacheable log – cache contamination
  o Issues with uncachable log – sub-optimal design in
    • Uncacheable write instructions and programming interface
    • Hardware components, e.g., write-combining buffer design and the way it is used

• More results
  o Sensitivity study on read/write ratio in transactions
  o Sensitivity study on transaction size
  o Other data structures: hash table, rbtree, b+tree, etc.
Logging in Persistent Memory: to Cache, or Not to Cache?

Mengjie Li, Matheus Ogleari, Jishen Zhao