A. Fahmy A. Jendo W. Golab
{a7fahmy, ajendo, wgolab}@uwaterloo.ca

Faculty of Engineering

UNIVERSITY OF WATERLOO

The Illusive Failure-Atomic Double-Width Compare-And-Swap

ApPLIED Workshop, Nantes, France, 2024

Outline

Table of Contents

Intel Optane persistent memory

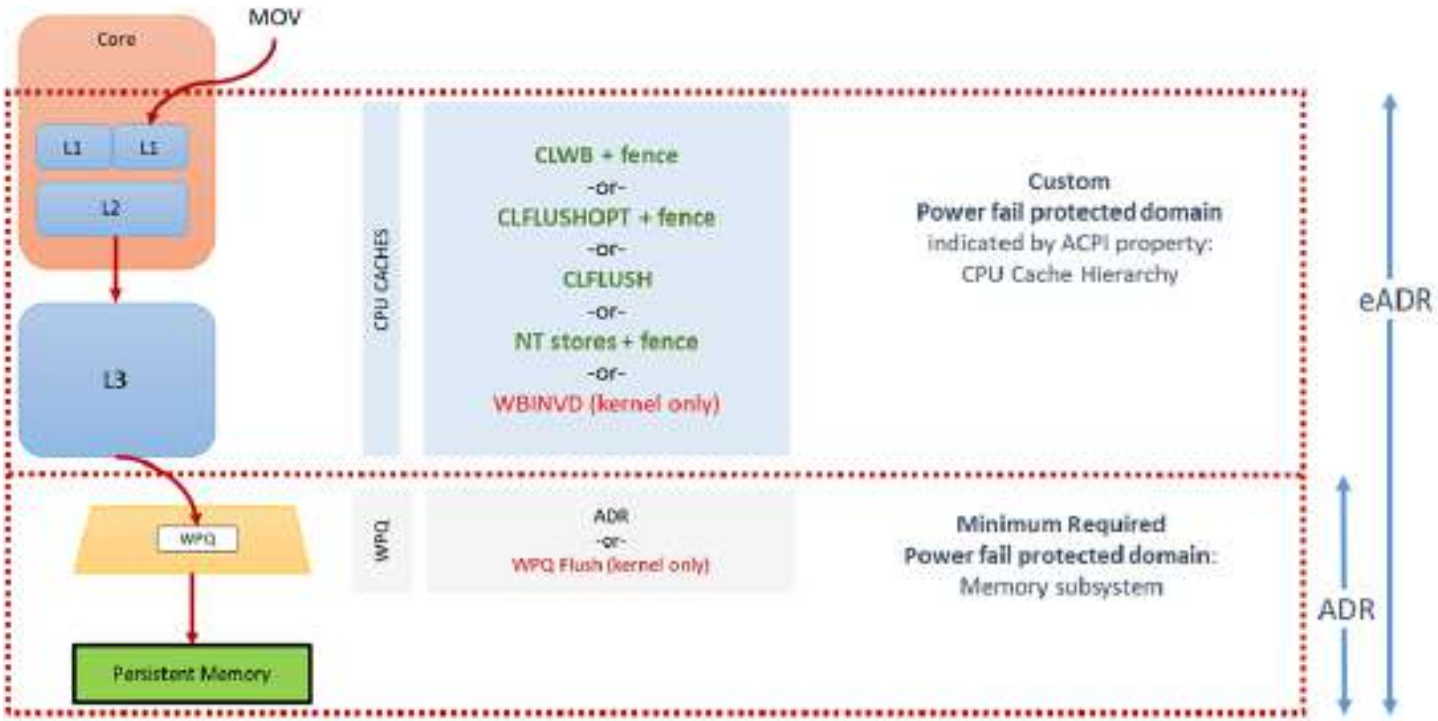

Figure: Power-fail protection domain on Intel platforms.

Image courtesy of Intel:
`https://www.intel.com/content/dam/developer/articles/training/pmem-learn-more-series-part-2/PMEM_LearnMore2_2.png`

Table of Contents

Intel's DWCAS instruction: CMPXCHG8B/CMPXCHG16B

Description:

"Compares the 64-bit value in EDX:EAX (or 128-bit value in RDX:RAX if operand size is 128 bits) with the operand (destination operand). If the values are equal, the 64-bit value in ECX:EBX (or 128-bit value in RCX:RBX) is stored in the destination operand. Otherwise, the value in the destination operand is loaded into EDX:EAX (or RDX:RAX). The destination operand is an 8-byte memory location (or 16-byte memory location if operand size is 128 bits)."

"This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically."

`https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2a-manual.pdf`

64-bit mode (128-bit width) pseudo-code

DEST: destination address
RDX:RAX: comparison value
RCX:RBX: new value
ZF: zero flag (indicates equality)

```
TEMP128 ← DEST
IF (RDX:RAX = TEMP128) THEN
    ZF ← 1;
    DEST ← RCX:RBX;
ELSE
    ZF ← 0;
    RDX:RAX ← TEMP128;
    DEST ← TEMP128;
FI;
```

https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2a-manual.pdf

Comments from Andy Rudoff [5]

PROGRAMMING

Persistent Memory Programming

ANDY RUDOFF

Andy Rudoff is a Senior Principal Engineer at Intel Corporation, focusing on non-volatile memory programming. He is a contributor to the SNIA NVM Programming Technical Work Group. His more than 30 years' industry experience includes design and development work in operating systems, file systems, networking, and fault management at companies large and small, including Sun Microsystems and VMware. Andy has taught various operating systems classes over the years and is a co-author of the popular UNIX Network Programming textbook. andy.rudoff@intel.com

In the June 2013 issue of *;login:*, I wrote about future interfaces for non-volatile memory (NVM) [1]. In it, I described an NVM programming model specification [2] under development in the SNIA NVM Programming Technical Work Group (TWG). In the four years that have passed, the spec has been published, and, as predicted, one of the programming models contained in the spec has become the focus of considerable follow-up work. That programming model, described in the spec as NVM.PM.FILE, states that persistent memory (PM) should be exposed by operating systems as memory-mapped files. In this article, I'll describe how the intended persistent memory programming model turned out in actual OS implementations, what work has been done to build on it, and what challenges are still ahead of us.

The Essential Background on Persistent Memory

Comments from Andy Rudoff [5]

"On Intel, only an eight-byte store, aligned on an eight-byte boundary, is guaranteed to be failure atomic."

"Anything larger than eight bytes can be torn by power failure and must be handled by software."

"... there's no single instruction that will solve that ..."

Recent theory papers that need power-fail atomic DWCAS

Publication	Algorithm
Attiya, Ben-Baruch, and Hendler [1]	Compare-And-Swap
Ben-David, Blelloch, Friedman, and Wei [3]	Compare-And-Swap
Ben-Baruch, Hendler, and Rusanovsky [2]	Compare-And-Swap
Jayanti, Jayanti, and Jayanti [4]	Compare-And-Swap and Load-Linked/Store-Conditional

Closer look at Attiya, Ben-Baruch, and Hendler [1]

Model: persistent shared memory, individual process crash failures

Code snippet: `C.CAS(<id,val>, <p,new>)`

"We assume that CAS is never invoked with old = new and that values written to C by the same process are distinct. (This assumption can be easily satisfied by augmenting each written value with a per-process sequence number.)"

Shared state:

process ID	per-process sequence number	value

Closer look at Ben-David, Blelloch, Friedman, and Wei [3]

Model: persistent shared memory, individual process crash failures

Code snippet: `CAS(x, <a, pid , seq'>, <b, i, seq>)`

Shared state:

value	process ID	per-process sequence number

Closer look at Ben-Baruch, Hendler, and Rusanovsky [2]

Model: persistent shared memory, system-wide crash failures

Code snippet: `C.CAS(<val, vec>, <new, newvec>)`

Shared state (for N processes):

value	bit vector of length N

Closer look at Jayanti, Jayanti, and Jayanti [4]

Model: persistent shared memory, individual process crash failures

Code snippet: `CAS(X, (ĥ, s), (h, ŝ))`

Shared state:

pointer	sequence number

Summary

- Four recent algorithms perform CAS operations on variables in persistent memory that in practice span multiple memory words [1, 3, 2, 4].
- Two of the publications [3, 4] refer to a DWCAS hardware instruction.
- All four algorithms can break if the atomic operation they rely on can be torn by a power failure.

Table of Contents

Is DWCAS truly necessary?

Yes, it is ...

- 64-bit Intel processors use 48-bit pointers, 40 bits are required if address is 256-byte aligned
- Linux process IDs are 16 bits, at least 8 bits are required to label 256 processes
- unique sequence numbers require at least 48 bits
  (2^{48} ops until overflow $\div$ millions of ops/s $\approx$ years until overflow)

Does a modern Intel processor flush a cache line atomically?

Andy Rudoff on the behaviour of cache line write-backs [6]:

"It is important to distinguish between what is *likely* to happen and what is *architecturally guaranteed* to happen."

Intel's architectural guarantees:

- "an 8-byte store is failure atomic"
- "a younger store won't pass an older store to the same cache line" (total store order)

Does a modern Intel processor flush a cache line atomically?

Andy Rudoff on the behaviour of cache line write-backs [6]:

```
/* assume the cache line starts off containing zeros */
buff[0] = 1
buff[8] = 1
CLWB buff
SFENCE
```

"Assuming you have taken steps to prevent the compiler from reordering the stores, then a younger store will not pass an older store to the same cache line. If my example code gets interrupted by a crash, on recovery either buff is all zeros, only buff[0] is 1, or both buff[0] is 1 and buff[1] are 1. It is not possible for buff[8] to be 1 and buff[0] to be zero."

"Of course, on a quiet system it is very likely that you can change a full cache line and flush it and the entire cache line travels to the persistent memory in a single chunk. It just isn't guaranteed."

Table of Contents

Workaround?

How about emulating power-fail atomic DWCAS using PMwCAS [7]?

- correct, but ...
- only lock-free
- much slower than hardware DWCAS

Opportunity!

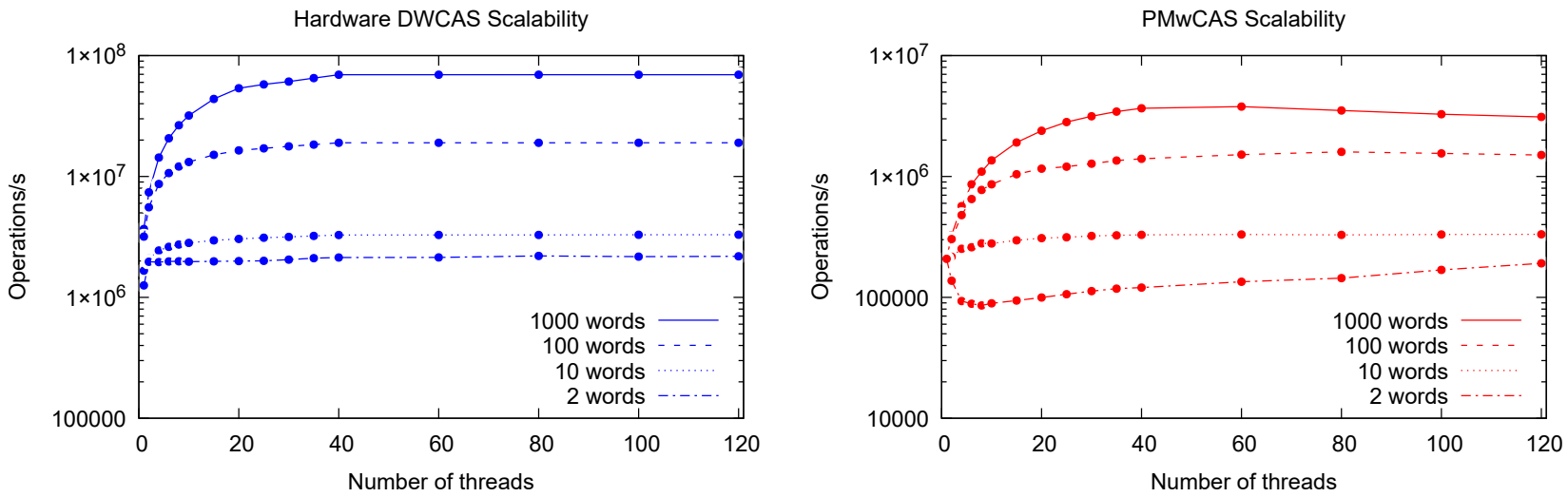

Figure: Performance comparison of the hardware DWCAS instruction against PMwCAS [7] on a 20-core Intel processor with Optane persistent memory.

Research questions

- How can we optimize PMwCAS for pairs of consecutive memory words?
- Can we do better if we start from scratch?

Table of Contents

Takeaways

- atomic $\neq$ power-fail atomic
- need for software implementations of power-fail atomic DWCAS

Bibliography I

[1] Hagit Attiya, Ohad Ben-Baruch, and Danny Hendler. Nesting-safe recoverable linearizability: Modular constructions for non-volatile memory. In *Proc. of the ACM Symposium on Principles of Distributed Computing*, PODC, page 7–16, 2018.

[2] Ohad Ben-Baruch, Danny Hendler, and Matan Rusanovsky. Upper and lower bounds on the space complexity of detectable objects. In *Proc. of the 39th Symposium on Principles of Distributed Computing*, PODC, page 11–20, 2020.

[3] Naama Ben-David, Guy E. Blelloch, Michal Friedman, and Yuanhao Wei. Delay-free concurrency on faulty persistent memory. In *Proc. of the 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, page 253–264, 2019.

[4] Prasad Jayanti, Siddhartha Jayanti, and Sucharita Jayanti. Durable Algorithms for Writable LL/SC and CAS with Dynamic Joining. In *Proc of the 37th International Symposium on Distributed Computing*, DISC, pages 25:1–25:20, 2023.

Bibliography II

[5] Andy Rudoff. Persistent memory programming. *login Usenix Mag.*, 42(2), 2017.

[6] Andy Rudoff. How to use clwb instruction. `https://groups.google.com/g/pmem/c/R8H3sKq9sLQ/m/ltL7Kng4BAAJ`, 2019. [Accessed 25-05-2024].

[7] Tianzheng Wang, Justin J. Levandoski, and Per-Åke Larson. Easy lock-free indexing in non-volatile memory. In *Proc. of the 34th IEEE International Conference on Data Engineering*, ICDE, pages 461–472, 2018.