iDO: Compiler-Directed Failure Atomicity for Nonvolatile Memory

Qingrui Liu¹  Joseph Izraelevitz²  Sekwon Lee³
Michael L. Scott²  Sam H. Noh³  Changhee Jung¹

¹Virginia Tech  ²University of Rochester  ³UNIST

Non-Volatile Memories Workshop
San Diego, CA, March 2019
Work originally presented at MICRO 2018
How To Use Byte-Addressable NVM?

• PCM, ReRAM, STT-MRAM being developed for density and low power
• Likely to displace some uses of DRAM
  • Envision machines with volatile registers and (for now) caches + byte-addressable NVM
• Could stick with traditional model: transient memory + persistent block storage
• Tempting to leave long-lived data “in memory” across program executions and even system crashes

• Failure model: non-corrupting errors not due to bugs in NVM-accessing code (power fail, kernel crash, ...)

Storage Model

- Traditional
- Failure-atomic msync
  - Still doesn’t leverage byte addressability
  - Reads and writes still occur at block granularity
- Direct access (DAX) with CLWB and SFENCE

Programming Model

- Nonblocking data structures
- Transactions
- Lock-based Failure-Atomic Sections (FASEs)
The Problem: Crash (In)Consistency

```c
int data;
bool valid;
STORE data = 0x1111
STORE valid = true
```
Partial Solution: Ordering Writes

(Intel ISA)

STORE data = 0x1111
CLWB data
SFENCE
STORE valid = true
CLWB valid
SFENCE
But Ordering is Not Enough

Suppose x must always equal y

LOCK L
store x = 3
WB x
fence
store y = 3
WB y
fence
UNLOCK L

Need failure atomicity!
We assume lock-based source code

“FASE” (Failure-Atomic SEction)

[Chakraborti et al., OOPSLA’14]
**Undo Logging**

- log old value of x
- WB & fence
- store x; WB
- log old value of y
- WB & fence
- store y; WB
- ...
- fence
- mark log finished
- WB & fence

Must track dependences across FASEs

**Redo Logging**

- log new value of x
- WB & fence
- log new value of y
- WB & fence
- ...
- mark log complete
- WB & fence
- store x; WB
- store y; WB
- ...
- mark log finished
- WB & fence

Must arrange to read our own writes
JUSTDO Logging  [Izraelevitz et al., ASPLOS’16]

log new value of x, &x, PC  
WB & fence
store x  
WB & fence
log new value of y, &y, PC  
WB & fence
store y  
WB & fence
...

On recovery, pick up at the most recent store: use code of original program to execute from logged PC through end of FASE; release all locks.

• Log size is $O(T+L)$ for $T$ threads and $L$ locks
• Must treat all data as “volatile” in FASEs
• WB & fence operations can be elided if caches are nonvolatile; expensive otherwise — i.e., on conventional machines
Key Observation for iDO

A region of code is **idempotent** iff its prefixes can be re-executed multiple times and it will still produce the same result.

\[
\begin{align*}
x &= 1 \\
y &= x \\
z &= 3
\end{align*}
\]

Output: \(x = y = 1; z = 3\)

Don’t have to log at every store!
iDO Logging ≈ JUSTDO + Idempotence

FASE

region

log recently-written still-live registers, PC
store; WB & fence
store; WB
... fence
log recently-written still-live registers, PC
store; WB & fence
store; WB
... fence
...
On recovery, resume FASE at the beginning of the interrupted idempotent region

- No need for happens-before FASE tracking (unlike UNDO)
- No need to take care to read own writes (unlike REDO)
- Small bounded log per thread
Idempotent Regions

• Leverage analysis of deKruif et al. [PLDI’12]
• Break at antidependences
• Typical region is just a few stores
• Can be very large:
  
  L.acquire()
  for (int i = 0; i < len; ++i)
      array[i] = i
  L.release()

• Could be extended with better alias analysis or code restructuring
Comparison

Compare iDO with:

- **ATLAS** [OOPSLA’14]: FASE + undo logging
- **JUSTDO** [ASPLOS’16]: FASE + resumption
- **NVThreads** [EuroSys’17]: FASE + copy-on-write
- **Mnemosyne** [ASPLOS’11]: Txns + redo logging
- **NVML** [FAST’15]: Txns + undo logging

Run on 4-socket, 64-core AMD Opteron 6276 server

Assume CLFLUSH+SFENCE over DRAM ≈ CLWB+SFENCE over NVM; MICRO paper includes sensitivity analysis
Performance

Redis throughput for databases with 10K, 100K, and 1M-element key ranges (single threaded)
Scalability

Hash map

Throughput (Ops/s) vs Threads

- NVTHREADS
- MNEMOSYNE
- ATLAS
- JUSTDO
- IDO

$10^7$
Ongoing Work

• Persistent nonblocking malloc/free, transactions (OO and word-based)
• Testing methodology
• Systems support for persistent segments
• Protected user-space libraries for safe sharing among untrusting apps
• Recovery from individual process failures
iDO Conclusion

• Compiler-directed failure atomicity for data in nonvolatile memory
• Makes resumption-based recovery practical on machines w/ volatile caches
• Better performance than FASE-based undo and redo
• Excellent scalability
• Fast recovery
MICRO paper available at:
www.cs.rochester.edu/research/synchronization/
www.cs.rochester.edu/u/scott/