Brief Announcement: Preserving Happens-Before in Persistent Memory

Joseph Izraelevitz   Hammurabi Mendes   Michael L. Scott
{jhi1,hmendes,scott}@cs.rochester.edu
University of Rochester

28th ACM Symposium on Parallelism in Algorithms and Architectures
July 11, 2016
Motivation

• Persistent non-volatile main memory (NVM) is coming
  • Byte addressable
  • Access times comparable to DRAM
  • Survives power outages

• Opportunity for fast durable storage
Motivation

- Persistent non-volatile main memory (NVM) is coming
  - Byte addressable
  - Access times comparable to DRAM
  - Survives power outages
- Opportunity for fast durable storage
- *But caches are expected to remain transient*
A Problem

• How to ensure consistency in the wake of a crash?
Goal

• How to ensure consistency in the wake of a crash?
  • Easily applicable transform to take a conventional program and make it persistence aware
  • Preserve happens-before ordering into persistence
The tricky case

T1

\[ x . s t (42) \]
\[ f l a g . s t _ { r e l } (1) \]

T2

\[ f l a g . l d _ { a c q } () \]
\[ y . s t (56) \]

\[ x , y \text{ synchronized by flag} \]
The tricky case: a naïve (incorrect) solution

T1

\begin{verbatim}
.x.st(42)
.x.pwb()
pfence()
flag.st_rel(1)
flag.pwb()
\end{verbatim}

T2

\begin{verbatim}
flag.ld_acq()
y.st(56)
y.pwb()
\end{verbatim}
The tricky case: a naïve (incorrect) solution

T1

x.st(42)
x.pwb()
pfence()
flag.st.rel(1)
flag.pwb()

T2

flag.ld_acq()
y.st(56)
y.pwb()
The tricky case: a naïve (incorrect) solution

T1

\[ x.st(42) \]
\[ x.pwb() \]
\[ pfence() \]
\[ flag.st.rel(1) \]
\[ flag.pwb() \]

What if we synchronize before T1 writes-back \textit{flag}?

T2

\[ flag.ld_acq() \]
\[ y.st(56) \]
\[ y.pwb() \]
The tricky case: a naïve (incorrect) solution

What if we synchronize before T1 writes-back flag?

T1

```plaintext
x.st(42)
x.pwb()
pfence()
flag.st_rel(1)
flag.pwb()
```

T2

```plaintext
flag.ld_acq()
y.st(56)
y.pwb()
```
The tricky case

**T1**

\[ \text{x.} \text{st(42)} \]
\[ \text{x.} \text{pwb()} \]
\[ \text{pfence()} \]
\[ \text{flag.} \text{st\_rel(1)} \]
\[ \text{flag.} \text{pwb()} \]

**T2**

\[ \text{flag.} \text{ld\_acq()} \]
\[ \text{flag.} \text{pwb()} \]
\[ \text{pfence()} \]
\[ \text{y.} \text{st(56)} \]
\[ \text{y.} \text{pwb()} \]

**Solution:** reader must also write-back *flag* and *fence.*
Our contributions

• **Explicit epoch persistency**
  • Very relaxed persistency model
  • Explicit write-back
  • Buffering
  • Release consistency
  • Sufficiently relaxed to easily translate to real hardware ISAs (x86, ARM v8) and other theoretical persistency models (strict\(^1\), epoch\(^2\), buffered epoch\(^3\))

• An **automatic transform**
  • Preserves the transient happens-before order into persistence
  • Nonblocking structures are always consistent

\(^1\)Pelley, ISCA ’11; \(^2\)Condit, ISCA ’09, \(^3\)Joshi, MICRO ’15
Explicit Epoch Persistency

- **pwb(void* addr)**
  - Initiates write-back of a specified location to persistent memory, but does not block

- **pfence()**
  - Enforces an ordering between previous and subsequent pwb’s in the calling thread

- **psync()**
  - Blocks until all preceding pfence’s in the calling thread have become persistent
## Translation to ISAs

<table>
<thead>
<tr>
<th>Explicit Epoch Persistency</th>
<th>Intel x86</th>
<th>ARM v8</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwb (addr)</code></td>
<td><code>CLWB addr</code></td>
<td><code>DC CVAC addr</code></td>
</tr>
<tr>
<td><code>pfence</code></td>
<td><code>SFENCE</code></td>
<td><code>DSB†</code></td>
</tr>
<tr>
<td></td>
<td><code>PCOMMIT*</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>SFENCE*</code></td>
<td></td>
</tr>
<tr>
<td><code>psync</code></td>
<td><code>\</code></td>
<td><code>\</code></td>
</tr>
</tbody>
</table>

*Required if memory controller is transient
†Requires persistent memory controller; Transient memory controllers are unsupported on ARMv8 at this time.
Informal proof in the paper
Conclusion

• **Explicit epoch persistency**
  • Explicit write-back
  • Buffering
  • Release consistency

• **An automatic transform**
  • Preserves the transient happens-before order into persistence
  • Nonblocking structures are always consistent

• **(Near) Future Work:**
  • Linearization of Persistent Memory Objects under a Full-System-Crash Failure Model (to appear DISC 2016)
    • Safety and liveness, analogues to linearization points, formal definitions of persistency model, proofs, ...