Persistency Semantics of the Intel-x86 Architecture

Azalea Raad\textsuperscript{1,2} John Wickerson\textsuperscript{2} Gil Neiger\textsuperscript{3} Viktor Vafeiadis\textsuperscript{1}

\textsuperscript{1} Max Planck Institute for Software Systems (MPI-SWS)
\textsuperscript{2} Imperial College London
\textsuperscript{3} Intel Corporation

\texttt{azalea@mpi-sws.org} \hspace{1cm} \texttt{SoundAndComplete.org} \hspace{1cm} \texttt{@azalearaad}
Computer Storage

✅ fast ✗ slow

✓ volatile ✓ persistent
What is Non-Volatile Memory (NVM)?

NVM: Hybrid Storage + Memory
Best of both worlds:
✓ **persistent** (like HDD)
✓ **fast, random access** (like RAM)
Q: Why *Formal* NVM Semantics?

**Volatile** memory

```plaintext
// x = 0
x := 1
// x = 1
// no recovery
// x = 0
```

**Non-Volatile** memory

```plaintext
// x = 0
x := 1
// x = 1
// recovery routine
// x = 1
```
Q: Why **Formal** NVM Semantics?

**A: Program Verification**
Q: Why **Formal** NVM Semantics?

What about **Concurrency**?

\[
\begin{align*}
// & \ x = y = \ldots = 0 \\
&& C_1 \ || \ C_2 \ || \ \ldots \ || \ C_n \\
&& // \ ???\\n&& // \ recovery \ routine\\n&& // \ ???
\end{align*}
\]
Formal Semantic Models

- Sequential (1940s)
- SC (1979)
- WMC (1990s)
Weak Memory Consistency (WMC)

**No** total execution order \((to)\) \(\Rightarrow\)

*weak* behaviour absent under SC, caused by:

- instruction *reordering* by compiler
- write propagation across *cache hierarchy*
Weak Memory Consistency (WMC)

No total execution order (to) ⇒

weak behavior absent under SC, caused by:

• instruction reordering by compiler
• write propagation across cache hierarchy

**Consistency Model**

the *order* in which writes are made visible to other threads

e.g. x86 (TSO), ARMv8, C11, Java
Formal Semantic Models

Difficulty

Sequential (1940s) SC (1979) WMC (1990s) WNVMC (2017)

This Talk
What Can Go Wrong?

// x=y=0
x := 1;
y := 1;

// recovery routine
// x=y=1 OR x=y=0 OR x=1; y=0 OR x=0; y=1

!! Execution continues *ahead of persistence*
   — *asynchronous* persists

!! Writes may persist *out of order*
   — *relaxed* persists
What Can Go Wrong?

**Consistency Model**

the *order* in which writes are *made visible* to other threads

**Persistency Model**

the *order* in which writes are *persisted* to NVM

**NVM Semantics**

Consistency + Persistency Model
Px86

(Persistent x86):

NVM Semantics

of the

x86 Architecture
Warmup: Sequential Px86
x86: (Sequential) Persistent Hardware Model

\[ x := 1 \quad : \quad \text{adds } x := 1 \quad \text{to } p\text{-buffer} \]
x86: (Sequential) Persistent Hardware Model

Unbuffered at **non-deterministic** points in time!

\[ x := 1 : \text{ adds } x := 1 \text{ to p-buffer} \]

\[ \text{unbuffer}^* : \text{ p-buffer to memory} \]

\* at non-deterministic times
x86: (Sequential) Persistent Hardware Model

Unbuffered at *non-deterministic* points in time!

Buffering & unbuffering orders may disagree!

\[ x := 1 \text{ : adds } x := 1 \text{ to } p\text{-buffer} \]

\[ \text{unbuffer}^* : p\text{-buffer to memory} \]

* at non-deterministic times
x86: (Sequential) Persistent Hardware Model

Unbuffered at **non-deterministic** points in time!

Buffering & unbuffering orders may disagree!

\[
x := 1 \quad : \quad \text{adds } x := 1 \text{ to } p\text{-buffer}
\]

\[
\text{unbuffer}^* \quad : \quad p\text{-buffer to memory}
\]

\[
a := x \quad : \quad \text{if } p\text{-buffer contains } x, \text{ reads latest entry}
\quad \text{else reads from memory}
\]

* at non-deterministic times
x86: (Sequential) Persistent Hardware Model

**CPU**

- **x:=1**: adds $x:=1$ to p-buffer

**Persistence Buffer**

- **unbuffer**: p-buffer to memory
- **a:=x**: if p-buffer contains $x$, reads latest entry else reads from memory
- **p-buffer lost; memory retained**

**Unbuffered at non-deterministic points in time!**

**Buffering & unbuffering orders may disagree!**

* at non-deterministic times
Fixing *Relaxed* Persists: Attempt #1

```plaintext
// x=0; y=0
x := 1;
y := 1;

// recovery routine
// x=1; y=1  OR  x=0; y=0  OR  x=1; y=0  OR  x=0; y=1
```

!! *out of order* persists

.persist barriers?
Persist Barriers: Desiderata

// x=0; y=0
x := 1;
// recovery routine
y := 1;

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

!! out of order persists

persist barriers?
Persist Barriers: **Desiderata**

```plaintext
// x=0; y=0

// x=1; y=0

!! out of order persists

Persist Barriers:

does not provide persist barriers!

x86 memory barriers (e.g. sfence, mfence) do not enforce persist ordering!
```
Fixing *Relaxed* Persists: Attempt #2

```
x := 1;
```  

// x=0; y=0

```
x := 1;

y := 1;
```

// recovery routine

```
// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1
```

!! *out of order* persists

漹 explicit persists?
Explicit Persists: *Desiderata*

```plaintext
// x=0; y=0
x := 1;
persist x;
y := 1;

// recovery routine
// x=1; y=1  OR x=0; y=0  OR x=1; y=0  OR x=0; y=1
```

!! *out of order* persists

_excepted persist?_
Explicit Persists: *Reality on x86*

```plaintext
// x=0; y=0
x := 1;
clwb x;
y := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

!! out of order persists

❖ explicit persists?
clwb x/clflushopt x/clflush x:
    asynchronously persist cache line containing x
```
Explicit Persists: *Reality on x86*

```
// x=0; y=0
x := 1;
clwb x;
y := 1;
```

x86 explicit persists are **asynchronous** and can themselves **persist out of order**!

```
!! out of order persists

遏 explicit persists?
clwb x/clflushopt x/clflush x:
asynchronously persist cache line containing x
```
Solution: **Persist Sequence**

```plaintext
// x=0; y=0
x := 1;
clwb x;
sfence;
y := 1;

// recovery routine
// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1
```

- **Waits** until earlier writes on `x` are persisted
- **Disallows reordering**

- ✓ **synchronous** persists
- ✓ **no out of order** persists
x86 Persists: **clwb, clflushopt, clflush**

- **clwb** and **clflushopt**: same ordering constraints
- **clwb** does not invalidate cache line
- **clflushopt** invalidates cache line
- **clflush**: strongest ordering constraints; invalidates cache line
Concurrent Px86
x86: (Volatile) Concurrent Hardware Model (TSO)

\[ x := 1 \] : adds \( x := 1 \) to buffer

unbuffer* : buffer to memory

\[ a := x \] : if buffer contains \( x \), reads latest entry else reads from memory

\[ \text{buffer and memory lost} \]

* at non-deterministic times
Px86: Persistent & Concurrent x86

Sequential, Persistent x86

Concurrent, Volatile x86
Persistent x86 (Px86)

buffer/unbuffer order: *consistency* model
Persistent x86 (Px86)

buffer/unbuffer order: *consistency* model

buffer/unbuffer order: *persistency* model
Persistent x86 (Px86)

buffer/unbuffer order: *consistency* model

buffer/unbuffer order: *persistence* model

*NVM Semantics (Px86)*
“Executions of the `clwb` instruction are ordered with respect to fence instructions …”

“ They are not ordered with respect to other executions of `clwb`, to executions of `clflush` and `clflushopt` …”

**Ambiguities in text!**

↓

**Two** Px86 models
"Executions of the `clwb` instruction are ordered with respect to fence instructions …”

“They are not ordered with respect to other executions of `clwb`, to executions of `clflush` and `clflushopt` …”

**Ambiguities in text!**

↓

**Two Px86 models**
Summary

✓ Formalised Intel-x86 NVM semantics:
  ✬ $Px86_{man}$: equivalent operational & declarative models
  ✬ $Px86_{sim}$: equivalent operational & declarative models

✓ More in the paper
  ✬ *Persistent transactional* library implemented in Px86
  ✬ *Persistent queue* library implemented in Px86

❓ Future Work:
  ✬ program logics
  ✬ model checking algorithms
  ✬ litmus testing
Summary

✓ Formalised Intel-x86 NVM semantics:
  ✧ $\textit{Px86_{man}}$: equivalent operational & declarative models
  ✧ $\textit{Px86_{sim}}$: equivalent operational & declarative models

✓ More in the paper
  ✧ *Persistent transactional* library implemented in Px86
  ✧ *Persistent queue* library implemented in Px86

❓ Future Work:
  ✧ program logics
  ✧ model checking algorithms
  ✧ litmus testing

Thank You for Listening!

azalea@mpi-sws.org  SoundAndComplete.org  @azalearaad