Weak Memory Models

Weak memory models are everywhere for performance reasons

- Hardware:
  - Intel x86-TSO
  - Power / ARM
  - GPUs

- Programming languages:
  - Java (covered in previous lectures)
  - C/C++
  - Compilers

- Software systems:
  - Databases
  - Operating Systems, e.g., Linux
Section 1

Hardware Programmer Memory Models
x86-TSO Memory model

References

  - A nice very thorough course on weak memory models
  - We are using several examples from this course

Consider the following x86 assembly program with initial shared memory values $[x] = 0$ and $[y] = 0$

```
Thread 0
MOV [x] <- 1
MOV EAX <- [y]

Thread 1
MOV [y] <- 1
MOV EBX <- [x]
```

What are the possible final values for EAX and EBX?
What are the possible final values of EAX and EBX?

- EAX = 1, EBX = 1
- EAX = 0, EBX = 1
- EAX = 1, EBX = 0

When we run the code we also observe EAX = 0, EBX = 0!

This is not something that is possible in any sequentially consistent execution!

The x86 memory model is not sequentially consistent but a weaker memory model

A formalization of the programmer memory model is called x86-TSO
x86-TSO memory subsystem model:

- H/W thread
- Write Buffer
- Lock
- Shared Memory
- H/W thread

...
x86-TSO Abstract Machine Semantics

Write buffers

- Each hardware thread has its own write buffer, which is a FIFO of unbounded size storing writes made by the hardware thread together with their store addresses.
- Read operations must read their data from the latest write to a memory location in the write buffer, if there is one matching the address of the read; otherwise reads are satisfied from the shared memory.
- When a write operation is performed, the write is appended to the store buffer.
- Each one of the writes stored in the store buffer will be eventually flushed to the shared memory in the background.
- There is a MFENCE instruction to flush the store buffer of a hardware thread before continuing with the next instruction.
Implementing LOCK prefix

- The INC increment command consists of a read operation followed by a write operation. Thus it is not atomic.
- The LOCK prefix can be added to an instruction to make it atomic. In addition to increment, there is e.g., atomic compare-exchange that can be implemented this way.
- In particular, the locked increment LOCK INC command will do an atomic increment.
- To implement the LOCK prefix, a global lock needs to be grabbed, and this is to be reflected in the semantics, see next slide.
Modifications to implement the LOCK prefix

- Read operations are only allowed when the global lock is not held by another hardware thread
- Write operations appending to the store buffer FIFO are always allowed
- Flushing writes from the store buffer to the shared memory is only allowed if the global lock is not held by another hardware thread
- If the global lock is free, a hardware thread can acquire the global lock and start an instruction with the LOCK prefix
- When the instruction with a LOCK prefix finishes execution, it frees the global lock
Goals of the x86-TSO definition

- Always allow more behaviours than any hardware implementation (e.g., FIFOs are of unbounded size, while any real HW will have bounded FIFOs)
- Formalize the Intel and AMD technical specifications written in natural language (finding specification bugs in the process)
- Act as a model to prove how to compile e.g., Java into x86 assembly in a provably correct fashion
- Act as a model to prove which compiler optimizations are correct for concurrent programs that contain data races
- Enable to show that data race free programs under x86-TSO are sequentially consistent
Power and ARM Memory Model

Power and ARM have a much more complicated weak memory model

- The following program can terminate with $r = 0$, even when initially $x = y = 0$

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = 1$</td>
<td>while($y == 0$) {};</td>
</tr>
<tr>
<td>$y = 1$</td>
<td>$r = x$</td>
</tr>
</tbody>
</table>

- Can reorder writes + much much more, for gory details & ARM HW bugs, see:
Power and ARM allow among other things:

- Reads to different locations can be scheduled out-of-order.
- Writes to different locations can be scheduled out-of-order.
- Read operations can be performed speculatively, i.e., even before branches before them are resolved to be taken or not.
- Writes can come visible to different hardware threads at different times.
- The formal model is quite complex.
- In addition some ARM hardware has bugs (see paper on previous slide) further complicating things.
Power and ARM Abstract Machine Model

From ARM and Power memory model tutorial:
https://www.cl.cam.ac.uk/~pes20/ppc-supplemental/test7.pdf
The GPU memory models are not really well documented by either NVIDIA nor AMD

- Documentation on the memory models are missing and some programmer guides are just plainly wrong
- Researchers are trying to reverse-engineer the memory model using so called litmus tests, and creating a model of the programmer guarantees
- For details, see e.g.:
Section 2

Weak Memory Model in C/C++
C11/C++11 Memory model

The C and C++ programming languages were standardized in 2011

- ISO/IEC 14882:2011 - Information technology - Programming languages - C++
- The C++ language standard is over 1300 pages long
- The idea is the same as for Java: Programs without race conditions have sequentially consistent behaviors
- Allows out-of-thin air executions, wording in C++14 has changed to disallow them: “Implementations should ensure that no "out-of-thin-air" values are computed that circularly depend on their own computation.”
- Problem: Checking whether a program has a race in an undecidable problem
- Also provides a semantics for programs with data races, and expert concurrency features including explicit weak memory model features called “low level atomics”
C/C++ Low level atomics

- The standard includes various atomic datatypes with sequential consistency helping write race free programs (please use these with default memory order).
  http://en.cppreference.com/w/cpp/atomic

- For memory order, see:
  http://en.cppreference.com/w/cpp/atomic/memory_order

- Problem: Sequentially consistent atomics are sometimes not performant enough for performance critical code.

- C/C++ Low level atomics provide a way to write programs with weaker memory semantics than sequential consistency in a well defined way (only for programmers who really know what they are doing!)
C11/C++11 Low level atomics

C/C++ Low level atomics

- For performance critical code (with races) the load and store operations can be given a memory order: `memory_order_relaxed`, `memory_order_consume`, `memory_order_acquire`, `memory_order_release`, `memory_order_acq_rel`, `memory_order_seq_cst`

- The details of these low level atomics work are left out of the course scope

- For a formalization of the low level atomics, see:

- Mapping low level atomics to x86 and ARM assembly:
  - https://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html

- See tutorials for demo on the use of low level atomics
Compiler Optimization for C/C++

References

- Many C/C++ compiler optimizations are safe once data race freedom is assumed:

- Many (most?) compiler optimizations are unsafe for C/C++ if data races are present, optimizing compilers for racy C/C++ code are very dangerous. C/C++ standard looks quite broken for optimizing programs with low level atomics:
  - Viktor Vafeiadis, Thibaut Balabonski, Soham Chakraborty, Robin Morisset, Francesco Zappa Nardelli: Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it. POPL 2015: 209-220
C/C++ Memory Model

References

- All you want to know about the C/C++ memory model, including improvement suggestions in Chapter 5:
    https://www.cs.kent.ac.uk/people/staff/mjb211/docs/toc.pdf

- Discussion of out-of-thin air problem for C++:
  http://doi.acm.org/10.1145/2618128.2618134

- Classical paper motivating the work on C/C++ memory model:
  - Hans-Juergen Boehm: Threads cannot be implemented as a library. PLDI 2005: 261-268
Section 3

Weak Memory Models in Software Systems
Databases use weak memory models

Relational databases also use weak memory models for transactions

- Databases are employing weak memory models to improve their performance:
Databases use weak memory models

Relational databases also use weak memory models for transactions

- Instead of Serializability (Database version of sequential consistency), weaker models are used, see HAT paper for definitions:
  - Read Committed
  - Repeatable Read
  - Snapshot Isolation
  - Cursor Stability
  - Consistent Read
Example: Snapshot Isolation

Snapshot isolation

**Intuition:**

- A transaction obtains one globally unique snapshot timestamp when it starts.
- All reads inside the transaction are done at the time of the snapshot timestamp.
- When a transaction wants to commit, it obtains a commit timestamp, which is also a globally unique timestamp.
- For each data item the transaction wants to write, it needs to check the data item has not been modified by a transaction with a commit timestamp between the snapshot and our commit timestamp.
- If this check succeeds for all data items to be written, the transaction commits, otherwise it aborts.
Example: Snapshot Isolation (cnt.)

Snapshot isolation (cnt.)

- There are outcomes of concurrent transactions which are not possible in any interleaving of the transactions
- Snapshot isolation allows read-only transactions to always commit without locking
- Almost all relational databases are using a weaker notion of transactions than serializability
- Quite often serializability is not even a configuration option!
Databases use weak memory models

Semantics for transactions from the HAT paper:

<table>
<thead>
<tr>
<th>Database</th>
<th>Default</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actian Ingres 10.0/10S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Aerospike</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>Akiban Persistit</td>
<td>SI</td>
<td>SI</td>
</tr>
<tr>
<td>Clustrix CLX 4100</td>
<td>RR</td>
<td>RR</td>
</tr>
<tr>
<td>Greenplum 4.1</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>IBM DB2 10 for z/OS</td>
<td>CS</td>
<td>S</td>
</tr>
<tr>
<td>IBM Informix 11.50</td>
<td>Depends</td>
<td>S</td>
</tr>
<tr>
<td>MySQL 5.6</td>
<td>RR</td>
<td>S</td>
</tr>
<tr>
<td>MemSQL 1b</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>MS SQL Server 2012</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>NuoDB</td>
<td>CR</td>
<td>CR</td>
</tr>
<tr>
<td>Oracle 11g</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>Oracle Berkeley DB</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Oracle Berkeley DB JE</td>
<td>RR</td>
<td>S</td>
</tr>
<tr>
<td>Postgres 9.2.2</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>SAP HANA</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>ScaleDB 1.02</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>VoltDB</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>


Table 2: Default and maximum isolation levels for ACID and NewSQL databases as of January 2013 (from [9]).
Linux memory model

Linux does not have a formal memory model

- Just a set of coding rules and hints for porting low level primitives to different architectures
- Depends on the interplay of volatile variable accesses not being reordered by the compiler combined with memory barriers
- Some initial work on formalizing the Linux memory model by the C++ standardization people: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4374.html