Threads and Memory Models

Lecture 15

CS 501
5/20/2013
Preliminaries

- Papers
  - *Foundations of the C++ Concurrency Memory Model*
    Boehm and Adve
  - *The Java Memory Model*
    Manson, Pugh, and Adve
Threads and shared memory

- Multithreaded programs allow multiple threads to run concurrently.
- Each thread has its own local variables (stack and registers), but...
- All threads share a common view of memory (globals / statics)
- Commonly used to multiple cores in hardware
Safety of optimization

• As defined in Lecture 1:

*If, in their actual program context, the result of evaluating e’ cannot be distinguished from the result of evaluating e, the compiler can substitute e’ for e.*

• What does this mean in a multi-threaded setting?
Register promotion

// x is global, initially 0

void foo(int* a, int n) {
    for (int i = 0; i < n; ++i)
        x += i;
}

// x is global, initially 0
void foo(int* a, int n) {
    for (int i = 0; i < n; ++i)
        x += i;
}

// Optimized
void foo(int* a, int n) {
    int reg = x;
    for (int i = 0; i < n; ++i)
        reg += i;
    x = reg;
}
Before optimization

// Thread 1

void foo(int* a, int n) {
    for (int i = 0; i < n; ++i)
        x += i;
}

// Thread 2

void bar() {
    x = 10;
    ...
}
Before optimization

// Thread 1
void foo(int* a, int n) {
    for (int i = 0; i < n; ++i)
        x += i;
}

// Thread 2
void bar() {
    x = 10;
    ...
}

What happens when n == 0?
After optimization

// Thread 1

void foo(int* a, int n) {
    int reg = x;
    for (int i = 0; i < n; ++i)
        reg += i;
    x = reg;
}

// Thread 2

void bar() {
    x = 10;
    ...
}

What happens when n == 0?
What happened?

- In executions where \( n == 0 \), the compiler optimization creates a value out of thin air.
  - Original code: \( x == 10 \) is guaranteed
  - Optimized code: new write of \( x = 0 \) creates new result
- Safety is no longer maintained
How did we get here?

- C & C++ originally defined as single-threaded languages
- Compilers didn’t consider threads
- Threads were provided by externally libraries (e.g. pthreads) that defined their own semantics
- This is a broken model!
- New spec explicitly deals with threads (Boehm, et al)
Dekker’s example

Thread 1
x = 1; \hspace{1cm} (a)
r1 = y; \hspace{1cm} (b)

Thread 2
y = 1; \hspace{1cm} (c)
r2 = x; \hspace{1cm} (d)
Dekker’s example

- Initially, $x == y == 0$

  Thread 1
  
  $x = 1;$ (a)
  
  $r1 = y;$ (b)

  Thread 2
  
  $y = 1;$ (c)
  
  $r2 = x;$ (d)
Dekker's example

- Initially, \( x == y == 0 \)

  Thread 1
  \[
  \begin{align*}
  x &= 1; & (a) \\
  r1 &= y; & (b)
  \end{align*}
  \]

  Thread 2
  \[
  \begin{align*}
  y &= 1; & (c) \\
  r2 &= x; & (d)
  \end{align*}
  \]
Dekker’s example

• Initially, \( x == y == 0 \)

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 1; ) (a)</td>
<td>( y = 1; ) (c)</td>
</tr>
<tr>
<td>( r1 = y; ) (b)</td>
<td>( r2 = x; ) (d)</td>
</tr>
</tbody>
</table>
Dekker’s example

• Initially, $x == y == 0$

  Thread 1
  
  $x = 1$; (a)  
  $r1 = y$; (b)

  Thread 2
  
  $y = 1$; (c)  
  $r2 = x$; (d)

• What are possible executions?
Dekker’s example

• Initially, \( x == y == 0 \)

  Thread 1                                  Thread 2
  \( x = 1; \quad (a) \quad y = 1; \quad (c) \)
  \( r1 = y; \quad (b) \quad r2 = x; \quad (d) \)

• What are possible executions?

  • Consider interleavings of thread 1 & 2:
Dekker’s example

• Initially, \( x == y == 0 \)

  Thread 1                        Thread 2
  \( x = 1; \) (a)                \( y = 1; \) (c)
  \( r1 = y; \) (b)               \( r2 = x; \) (d)

• What are possible executions?
  • Consider interleavings of thread 1 & 2:
    • \( abcd, acbd, acdb, cdab, cadb, cabd \)
Dekker’s example

• Initially, $x == y == 0$

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = 1; \ (a)$</td>
<td>$y = 1; \ (c)$</td>
</tr>
<tr>
<td>$r_1 = y; \ (b)$</td>
<td>$r_2 = x; \ (d)$</td>
</tr>
</tbody>
</table>

• What are possible executions?

• Consider interleavings of thread 1 & 2:
  - $abcd, acbd, acdb, cdab, cadb, cabd$
Dekker’s example

• Initially, $x == y == 0$

  
  Thread 1  
  $x = 1; \ (a)$  
  $r1 = y; \ (b)$
  
  Thread 2  
  $y = 1; \ (c)$  
  $r2 = x; \ (d)$

• What are possible executions?

  • Consider interleavings of thread 1 & 2:

    • $abcd, acbd, acdb, cdab, cadb, cabd$
Dekker’s example

• Initially, x == y == 0
  Thread 1
  x = 1;
  r1 = y;
  Thread 2
  y = 1;
  r2 = x;

• Can r1 == r2 == 0?
  • No interleaving gives this results, but...
  • Most hardware will allow it (store buffers)
  • Many compilers will allow it (instruction sched.)
What is a correct execution?

• Simplest notion: sequential consistency (Lamport ‘79)
  "... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."

• This is essentially the interleaving model

• Too expensive (?)
  • Nobody implements this in practice
Refined notion

• Guarantee sequential consistency only for correctly synchronized programs (Adve)
• Give the programmer rules to follow
• Give simple semantics when rules are obeyed
• Correctly synchronized
  • Must be intuitive to programmer
  • Must not be restrictive for implementer
Data races

- Two operations *conflict* if they both access a memory location and one is a write.

- A execution contains a *data race* if two adjacent operations from two different threads conflict

  ```
  x = 1; y = 1; r1 = y; r2 = x;
  ```

- A program is race-free if, for no sequentially consistent execution (i.e., interleaving) has a data race.
Correct synchronization

• We call a program correctly synchronized if it is data race free.

• Basic contract:
  • If programmers write race free programs, implementers will provide sequentially consistent semantics.
  • This is the fundamental underpinning for Java, C++ memory models.
Another example

- Dekker’s example is not race free.
- What about: (initially, $x == y == 0$)

  Thread 1
  
  $r_1 = x$;
  
  if ($r_1 > 0$)
  
  $y = 1$;

  Thread 2

  $r_2 = y$;

  if ($r_2 > 0$)

  $x = 1$;

- Is $r_1 == r_2 == 1$ legal?
How do we avoid races?

- Mutual exclusion:

  - Thread acquires lock before accessing a shared variable:

    Thread 1
    lock (mutex);
    tmp1 = x;
    tmp2 = tmp1 + 1;
    x = tmp2
    unlock (mutex);

    Thread 2
    lock (mutex);
    tmp3 = x;
    tmp4 = tmp3 + 1;
    x = tmp4
    unlock (mutex);

- Locks disallow problematic interleavings
How do we avoid races?

- Volatile variables (atomic in new C++):
  - Certain variables are declared with stronger ordering semantics (initially, x and flag are 0):

    Thread 1
    x = 1;
    flag = 1;

    Thread 2
    if (flag == 1)
    t = x;

- If flag is declared volatile, then write to x cannot be sunk in T1 and read from x cannot be hoisted in T2 by definition.

- Compiler must respect ordering.
What does this mean for compilers?

- In the absence of synchronization, compilers may *almost* operate as if programs were single-threaded.

- Compilers must respect ordering due to synchronization (and generate necessary hardware instructions).

- Caveat: compiler must not introduce races into correctly synchronized code (e.g. register prom.)
What happens on a race?
What happens on a race?

- In C++, undefined semantics
What happens on a race?

- In C++, undefined semantics

  Thread 1 \( (x == y == 0) \)
  - \( x = 1; \) (a)
  - \( r1 = y; \) (b)

  Thread 2
  - \( y = 1; \) (c)
  - \( r2 = x; \) (d)
What happens on a race?

- In C++, undefined semantics

  Thread 1 (\(x == y == 0\))  
  \(x = 1;\)  \((a)\)  
  \(r1 = y;\)  \((b)\)  

  Thread 2  
  \(y = 1;\)  \((c)\)  
  \(r2 = x;\)  \((d)\)
What happens on a race?

- In C++, undefined semantics

 Thread 1 \quad (x == y == 0) \quad Thread 2
  \begin{align*}
  x &= l; \quad (a) \\
  r1 &= y; \quad (b) \\
  y &= l; \quad (c) \\
  r2 &= x; \quad (d)
  \end{align*}
What happens on a race?

- In C++, undefined semantics

  Thread 1 \((x == y == 0)\)  
  \(x = 1;\)  \((a)\)  
  \(r1 = y;\)  \((b)\)

  Thread 2
  \(y = 1;\)  \((c)\)  
  \(r2 = x;\)  \((d)\)

- Valid results:
What happens on a race?

- In C++, undefined semantics

  Thread 1     (x == y == 0)  Thread 2
  x = 1;       (a)            y = 1;     (c)
  r1 = y;      (b)            r2 = x;    (d)

- Valid results:
  - r1 = 0 and r2 = 0
What happens on a race?

• In C++, undefined semantics

  Thread 1     (x == y == 0)  Thread 2
  x = 1;       (a)           y = 1;      (c)
  r1 = y;      (b)           r2 = x;      (d)

• Valid results:
  • r1 = 0 and r2 = 0
  • r1 = 0 and r2 = 2
What happens on a race?

• In C++, undefined semantics

  Thread 1 \quad (x == y == 0) \quad Thread 2
  \begin{align*}
  x &= 1; \quad & (a) \\
  r1 &= y; \quad & (b) \\
  \end{align*}

  \begin{align*}
  y &= 1; \quad & (c) \\
  r2 &= x; \quad & (d) \\
  \end{align*}

• Valid results:
  • r1 = 0 and r2 = 0
  • r1 = 0 and r2 = 2
  • “format c:\”
What happens on a race?

• In C++, undefined semantics

  Thread 1 (x == y == 0) Thread 2
  x = 1; (a) y = 1; (c)
  r1 = y; (b) r2 = x; (d)

• Valid results:
  • r1 = 0 and r2 = 0
  • r1 = 0 and r2 = 2
  • “format c:\”

• No such thing as a benign race in C++!
Hard to bound effects

unsigned x;

if (x < 3) {
    // x modified by another thread
    switch (x) {
        case 0: ...
        case 1: ...
        case 2: ...
    }
}

• Compiler should be able to generate table
• Assumes x in range after check
• Async change to x causes arbitrary behavior
Type-safety issues

- In Java, data races cannot violate type safety
- Java promises a measure of security
- Synch. errors may be used on purposed by untrusted code to open / exploit holes
- Java memory model must provide some guarantees in the presence of races
- Why is previous example not an issue in Java?
Java ordering

• Java’s memory model defines a partial order overall all actions in a program.
• For each thread, actions must happen in program order.
• Globally, synchronization actions must be totally ordered.
• These two must be consistent.
Synchronization edges

- A *synchronization edge* is defined each release to each matching acquire that follows in synchronization order.

- A volatile write has an edge to all later volatile reads to the same variable.

- An unlock has an edge to all later lock operations to the same monitor.
Happens-before

- Java defines a *happens-before* relationship as the transitive closure over program order and synchronization edges.

- A read $r$ is not allowed to see a write $w$ to the same variable $v$ if
  
  - $r \text{ hb } w$ or
  
  - exists another write $w'$ to $v$ such that $w \text{ hb } w'$ \text{ hb } r

  - otherwise, $r$ may see $w$
• What about: (initially, \(x == y == 0\))

Thread 1
\[
\begin{align*}
  r1 &= x; \\
  \text{if } (r1 > 0) \\
  y &= 1;
\end{align*}
\]

Thread 2
\[
\begin{align*}
  r2 &= y; \\
  \text{if } (r2 > 0) \\
  x &= 1;
\end{align*}
\]

• Is \(r1 == r2 == 1\) legal?
Another example

• What about: (initially, $x == y == 0$)

Thread 1
  $r1 = x$;
  $r2 = x$;
  if ($r1 == r2$)
    $y = 1$;

Thread 2
  $r3 = y$;
  $x = r3$

• Is $r1 == r2 == r3 == 1$ legal?
Races in Java

- Incorrectly synchronization programs in Java must still obey happens-before
- Additional subtle restrictions:
  - final fields
  - causal safety ... not quite right yet
- We’ll revisit with transactional memory....
C++ vs Java

- Data-Race-Free programs guarantee Sequentially Consistent semantics.
- Violations of race freedom
  - C++: All bets are off!
  - Java: Still need type safety, security, ...
• Is sequential consistency *that* expensive?
• Should we avoid programming models with mutable shared memory?