Memory Consistency Models

Raghesh A (CS12D015)

PACE Lab, Department of CSE, IIT Madras

January 7, 2015

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.
  - One common approach is to apply synchronization.
  - But the degree of synchronization must be minimal.
  - Basically synchronization controls the order of read and write to shared memory locations.
  - Without sychronization the order of read and writes may be altered either by compiler (optmizations) or by hardware for better performance.

Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.
  - One common approach is to apply synchronization.
  - But the degree of synchronization must be minimal.
  - Basically synchronization controls the order of read and write to shared memory locations.
  - Without sychronization the order of read and writes may be altered either by compiler (optmizations) or by hardware for better performance.
- The objective.
  - Less compromise on performance - With less degree of synchronization shall we get acceptable output ????
  - Arguing about the correctness of the program.

Memory Model

- A formal specification of how the memory system will appear to the programmer.
- Eliminating gap between expected behaviour (by the programmer) and the actual behaviour.
- Essentially, a **Memory Model** defines the legitimate orderings of read and write to memory locations.

Memory Model

- A formal specification of how the memory system will appear to the programmer.
- Eliminating gap between expected behaviour (by the programmer) and the actual behaviour.
- Essentially, a **Memory Model** defines the legitimate orderings of read and write to memory locations.
  - A strict memory model - Easy to argue about correctness, but restricts optimizations (both at compiler and hardware level).
  - A relaxed memory model - More opportunities for optimizations, but not easy to argue correctness.

Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...

Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.

Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.
- An Example [1]

Initially $X = Y = 0$

Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.
- An Example [1]

Initially $X = Y = 0$	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

- Some possible interleavings.

Execution 1	Execution 2	Execution 3
X = 1;	Y = 1;	X = 1;
r1 = Y;	r2 = X;	Y = 1;
Y = 1;	X = 1;	r1 = Y;
r2 = X;	r1 = Y;	r2 = X;
//r1 == 0;	//r1 == 1;	//r1 == 1;
//r2 == 1;	//r2 == 0;	//r2 == 1;

Relaxed Consistency (RC)

- SC is hard to realize.
- Too much restriction for hardware and compiler optmizations
- Even SC may not always be produce correct output.
- So we can live with **Relaxed Consistency Models**.

Relaxed Consistency (RC)

- SC is hard to realize.
- Too much restriction for hardware and compiler optmizations
- Even SC may not always be produce correct output.
- So we can live with **Relaxed Consistency Models**.
- Potential relaxations [2]
  - Program order: (Refers to different memory locations)
    - Relax $W \rightarrow R$ program order.
    - Relax $W \rightarrow W$ program order.
    - Relax $R \rightarrow R$ and $R \rightarrow W$ program order.

Relaxed Consistency (RC)

- SC is hard to realize.
- Too much restriction for hardware and compiler optmizations
- Even SC may not always be produce correct output.
- So we can live with **Relaxed Consistency Models**.
- Potential relaxations [2]
  - Program order: (Refers to different memory locations)
    - Relax $W \rightarrow R$ program order.
    - Relax $W \rightarrow W$ program order.
    - Relax $R \rightarrow R$ and $R \rightarrow W$ program order.
  - Write atomicity: (Refers to same memory location)
    - Read others' write early.

Relaxed Consistency (RC)

- SC is hard to realize.
- Too much restriction for hardware and compiler optmizations
- Even SC may not always be produce correct output.
- So we can live with **Relaxed Consistency Models**.
- Potential relaxations [2]
  - Program order: (Refers to different memory locations)
    - Relax $W \rightarrow R$ program order.
    - Relax $W \rightarrow W$ program order.
    - Relax $R \rightarrow R$ and $R \rightarrow W$ program order.
  - Write atomicity: (Refers to same memory location)
    - Read others' write early.
  - Write atomicity and program order.
    - Read own write early

Total Store Order (TSO)

Definition [2]:

1. A read can complete before an earlier write to a different address.
2. A read can not return the value of a write by another processor unless all processors have seen the write.
3. A read can return the value of own write before others see it (Makes use of store buffers).

Total Store Order (TSO)

Definition [2]:

1. A read can complete before an earlier write to a different address.
2. A read can not return the value of a write by another processor unless all processors have seen the write.
3. A read can return the value of own write before others see it (Makes use of store buffers).

Model	$W \rightarrow R$ order	$W \rightarrow W$ order	$R \rightarrow RW$ order	R Others' W Early	R Own W Early
TSO	$\surd$				$\surd$

Total Store Order (TSO)

Definition [2]:

1. A read can complete before an earlier write to a different address.
2. A read can not return the value of a write by another processor unless all processors have seen the write.
3. A read can return the value of own write before others see it (Makes use of store buffers).

Model	$W \rightarrow R$ order	$W \rightarrow W$ order	$R \rightarrow RW$ order	R Others' W Early	R Own W Early
TSO	$\surd$				$\surd$

- An Example

Initially $X = Y = 0$

Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

Total Store Order (TSO)

Definition [2]:

1. A read can complete before an earlier write to a different address.
2. A read can not return the value of a write by another processor unless all processors have seen the write.
3. A read can return the value of own write before others see it (Makes use of store buffers).

Model	$W \rightarrow R$ order	$W \rightarrow W$ order	$R \rightarrow RW$ order	R Others' W Early	R Own W Early
TSO	$\surd$				$\surd$

- An Example

Initially $X = Y = 0$	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

- Some possible interleavings (in addition to that of SC).

Time	Execution 1
t_1	r1 = Y;
$t_2 = t_1 + \delta$	X = 1;
$t_2 = t_1 + \delta$	Y = 1;
$t_3 = t_2 + \delta$	r2 = X;
	//r1 == 0;
	//r2 == 1;

- Effect of 1: It is possible to reorder read to Y before write to X.
- Effect of 2: It is impossible to get value of r2 as 0, by reading 0 from X, because an earlier write to X by Red thread should have seen by the Blue thread.

Total Store Order (TSO)

Definition [2]:

1. A read can complete before an earlier write to a different address.
2. A read can not return the value of a write by another processor unless all processors have seen the write.
3. A read can return the value of own write before others see it (Makes use of store buffers).

Model	$W \rightarrow R$ order	$W \rightarrow W$ order	$R \rightarrow RW$ order	R Others' W Early	R Own W Early
TSO	$\surd$				$\surd$

- An Example

Initially $X = Y = 0$

Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

- Some possible interleavings (in addition to that of SC).

Time	Execution 1
t_1	r1 = Y;
$t_2 = t_1 + \delta$	X = 1;
$t_2 = t_1 + \delta$	Y = 1;
$t_3 = t_2 + \delta$	r2 = X;
	//r1 == 0;
	//r2 == 1;

  - Effect of 1: It is possible to reorder read to Y before write to X.
  - Effect of 2: It is impossible to get value of r2 as 0, by reading 0 from X, because an earlier write to X by Red thread should have seen by the Blue thread.

- r1 = 0 and r2 = 0 is possible in TSO, but not in SC !!!

Correctness of RC [3]

- Same program may exhibit more executions on a relaxed model than SC [3].
- Let T_{Π}^{Y} be the set of executions of program Π on memory model Y. Then $T_{\Pi}^{Y} \subset T_{\Pi}^{SC}$ [3].
- To verify relaxed executions T_{Π}^{Y}, verify following two problems.
  - Use standard verification methodology for concurrent programs to show that the executions in T_{Π}^{SC} are correct.
  - Use specialized methodology for *memory model safety* verification showing that $T_{\Pi}^{Y} = T_{\Pi}^{SC}$.
  - The program is $Y - safe$ if $T_{\Pi}^{Y} = T_{\Pi}^{SC}$.

Linearisability

Sarita V. Adve and Hans-J. Boehm.

Memory models: A case for rethinking parallel languages and hardware.

Commun. ACM, 53(8):90–101, August 2010.

Memory Consistency Models.

`http://www.cs.utah.edu/~rajeev/cs7820/pres/7820-12.pdf`.

Sebastian Burckhardt and Madanlal Musuvathi.

Effective program verification for relaxed memory models.

In *Proceedings of the 20th International Conference on Computer Aided Verification*, CAV '08, pages 107–120, Berlin, Heidelberg, 2008. Springer-Verlag.