DIY7

Test generator

diy7 tool

- It is used for generating tests.It can be configured to test various memory models and architectures.
- **Basic command syntax:**
  - **diy7 -arch ARCH**
- **-arch ARCH**: Specify the target architecture (e.g., X86, ARM, PowerPC).

Candidate relaxations

- **Relaxation** refers to the process of weakening the constraints of a more restrictive model (like Sequential Consistency) to allow for greater concurrency and potentially improve performance
- A **candidate relaxation** is a specific type of relaxation applied to memory operations between threads. It is a way to describe and analyze how certain memory operations can be reordered or observed differently from what Sequential Consistency (SC) would enforce.
- **Types of candidate relaxations:**
  - Communication candidate relaxation
  - Program order candidate relaxation
  - Fence candidate relaxations

Communication candidate relaxation

These refer to specific types of relaxations that involve two events communicating via memory. These events, which can occur either on the same processor or across different processors, interact by reading from or writing to the same memory location.

diy7 syntax	Source	Target	Processor	Additional property
`Rfi`	W	R	Same	Target reads its value from source
`Rfe`	W	R	Different	Target reads its value from source
`Coi`	W	W	Same	Source precedes target in coherence order
`Coe`	W	W	Different	Source precedes target in coherence order
`Fri`	R	W	Same	Source reads a value from a write that precedes target in coherence order
`Fre`	R	W	Different	Source reads a value from a write that precedes target in coherence order

Program order Candidate relaxations

Program order candidate relaxations refer to the possible relaxations between two events that occur in program order, which means they occur on the same processor.

The syntax for program order candidate relaxations is:

Po(s|d)(R|W)(R|W)

- **Po**: Refers to *program order*.
- **s** or **d**:
  - **s**: Indicates that the two events are operating on the *same memory location*.
  - **d**: Indicates that the two events are operating on *different memory locations*.
- **R** or **W**:
  - **R**: Represents a *read* event.
  - **W**: Represents a *write* event

diy7 syntax	Source	Target	Location
PosRR	R	R	Same
PodRR	R	R	Diff
PosRW	R	W	Same
PodRW	R	W	Diff
PosWW	W	W	Same
PodWW	W	W	Diff
PosWR	W	R	Same
PodWR	W	R	Diff

Fence candidate relaxations

- Relaxed architectures provide specific instructions, namely barriers or fences, to enforce order of memory accesses.
- In diy7 the presence of a fence instruction is specified with fence candidate relaxations, similar to program order candidate relaxations, except that a fence instruction is inserted.
- The inserted fence is the strongest fence that varies based on the provided architecture, that is **mfence for x86 dmb for ARM , fence for riscv and sync for Power.**
- Basic command to list the fence candidate relaxation of particular architecture
    - **diy7 -arch ARCH -show fences**
    - **ARCH =** <C|CPP|LISA|AArch64|ARM|MIPS|PPC|X86|RISCV|X86_64>

Arch	Fence prefixes
X86	`MFence`
PPC	`Sync LwSync Eieio ISync`
ARM	`DSB DMB DMB.ST DSB.ST ISB`
AArch64[a]	`DMB.SY DMB.ST DMB.LD ISB`
MIPS	`Fence`
RISCV	`Fence.rw.rw Fence.rw.r Fence.rw.w Fence.r.rw Fence.r.r Fence.r.w Fence.w.rw Fence.w.r Fence.w.w`
C	`FenceSc FenceAR FenceAcq FenceRel FenceCons FenceRlx`

Usage of candidate relaxations with diy7

The options **-relax** and **-safe** in **diy7** allow users to define sets of memory ordering relaxations or constraints that the tool will use when generating and analyzing memory model behaviors through litmus tests

- **-relax**: Specifies which memory edges (like reads and writes) can be **relaxed** (reordered). It allows you to model behaviors where some operations do not follow strict program order.
- **-safe**: Specifies which memory edges are **safe** and must respect program order, meaning no relaxation is allowed for these edges.
- **Syntax :**
  - **diy7 -relax r -safe S**
  - r is the list of candidate relaxations to be relaxed(reordered)
  - s is the list of candidate relaxation sin safe set (cannot be reordered)
- **Safe Relaxations**: Expected to always hold (it must follow the memory ordering rules), and tests should **not** give "Ok."
- **Relaxed Relaxations**: Can be broken(may or may not follow memory reordering rules), and tests can **sometimes** give "Ok."

Use of conf file to generate the litmus tests

- Repeating command line options is painful and error prone. Besides, configuration parameters may get lost. Thus, we regroup those in configuration files that simply list the options to be passed to diy7, one option per line
- Pass that configuration file as the argument to option **“-conf”** used with diy7 command
- **Syntax:**
  - **diy7 -conf “conf-file”**

#sample conf-file

-arch X86
-nprocs 4
-size 6
-name rfi
-safe PosR* PodR* PodWW PosWW Rfe Fre
-relax Rfi
#At most three "instructions" per thread.
-ins 3

Additional relaxations

Intra processor dependencies

Data Dependencies

Data Dependencies occur when a memory access instruction relies on the value of a previous memory access instruction. They are specified in diy7 as:

- **Dp(s|d)(R|W)**: Indicates a data dependency where s (source) and d (destination) can be the same or different locations, and R (read) or W (write) specifies the target event type.
  - DpAddr(s|d)(R|W): Address dependency, meaning that the dependency is based on memory addresses.
  - DpData(s|d)W: Data dependency, meaning that the dependency is based on the data values.

Control Dependencies

Control Dependencies occur when the execution of a memory access depends on the outcome of a previous conditional operation. They are specified with a Ctrl tag:

- **Ctrl(s|d)(R|W)**: Indicates control dependencies where s (source) and d (destination) can be the same or different locations, and R (read) or W (write) specifies the type of the target event.
  - DpCtrl(s|d)W: Conditional execution of a write.
  - DpCtrlFenceI(s|d)R: Conditional execution of a read with synchronization (isync).

Identifying coherence orders with observers

- **Observers** are additional threads introduced into a test to observe and validate the ordering of memory operations.
- The core idea is to monitor how loads from the same memory location are handled by the system.
- Observers are threads that help test and validate memory consistency models by performing loads from shared memory locations.
- By executing multiple loads in sequence from the same location, observers can help determine the order in which memory operations are visible, which is essential for understanding coherence and memory consistency models.

-obs Option and Its Arguments

The -obs option in the diy7 tool controls the use of observer threads in the test generation process.

- **accept**: This setting allows the use of observers if they are beneficial for the test generation process. Observers will be included if they are useful for achieving the desired test cases.
- **avoid**: This is the default setting. It avoids using observers in the test generation process. The tool will generate tests without adding observer threads.
- **force**: This setting forces the inclusion of observers in all generated tests. It ensures that observers are used to validate coherence orders and memory operations, regardless of whether they are deemed necessary.
- **local**: This setting enables local observers. Local observers are used to test memory consistency at a more localized level and are not intended for broader coherence order testing.

-obstype Option and Its Arguments

- The -obstype option in diy7 specifies the style of observers used in the test generation process.
- Each style affects how observers are used and how they interact with the main threads.

argument	description	example

fenced	observers are placed with memory fences or barriers around their operations.	P0
		lwz r1,0(r2)
		lwz r3,0(r2)

loop	observers that perform their operations in a loop, continuously accessing memory locations	P0
		lwz r1,0(r2)
		sync
		lwz r3,0(r2)

straight	observers perform their memory accesses in a straightforward, linear fashion without additional synchronization or looping	P0
		L00:
		lwz r1,0(r2)
		cmpwi r1,0
		beq L00
		li r4,200
		L01:
		lwz r3,0(r2)
		cmpw r3,r1
		bne L02
		addi r4,r4,-1
		cmpwi r4,0
		bne L01
		L02:

Usage of -obs and -obstype option with diy7 tools

diyone7 : generates one litmus test from the specification of a violation of the sequential consistency memory model as a cycle

$ diyone7 -name 2+2WObs -obs force -obstype straight -arch PPC PodWW Coe PodWW Coe

diycross7 : The tool diycross7 has an interface similar to diyone7, except it accepts list of candidate relaxations where diyone7 accepts single candidate relaxations.

$ diycross7 -arch PPC -obs force -obstype fenced -name iriw Rfe DpdR,LwSyncdRR Fre Rfe DpdR,LwSyncdRR Fre

diy7 : generates several tests, aimed at confirming that candidate relaxations are relaxed or safe

$ diy7 -arch PPC -obs force -obstype fenced -name iriw -relax LwSyncdRR -safe Fre,Rfe,DpdR

Family Names

- **Family names** represent the structure of tests by summarizing the types of operations (reads and writes) and their sequence in the test cycles.
- These names are important because they help group similar tests together, which is useful for understanding and analyzing the behavior of memory models.

- **Example :** For instance, consider the cycle **PodWW Rfe PodRR Fre**

  There are two threads in the corresponding test (as there are two external communication candidate relaxations), one thread starts and ends with a write (written WW), while the other thread starts and ends with a read (written RR). The family name is thus WW+RR, (or RR+WW, but we choose the former)

Common Nicknames and Family Names

2+2W	WW+WW	PodWW Coe PodWW Coe
LB	RW+RW	PodRW Rfe PodRW Rfe
MP	WW+RR	PodWW Rfe PodRR Fre
R	WW+WR	PodWW Coe PodWR Fre
S	WW+RW	PodWW Rfe PodRW Coe
SB	WR+WR	PodWR Fre PodWR Fre

Additional tools: extracting cycles and classification

When non-standard family names or numeric names are used, it proves convenient to rename tests with the standard naming scheme. We provide two tools to do so:

- **mcycles7** that extracts cycles from litmus source files
- **classify7** that normalises and renames cycles

Usage :

- Create one configuratio file to generate litmus tests using diy7

  $ cat X.conf
  -arch X86
  -name X
  -nprocs 3
  -size 6
  -safe Pod**,Fre,Rfe,Wse
  -mode critical

$ diy7 -conf X.conf

Generator produced 23 tests

$ ls

@all X004.litmus X009.litmus X014.litmus X019.litmus X000.litmus X005.litmus X010.litmus X015.litmus X020.litmus X001.litmus X006.litmus X011.litmus X016.litmus X021.litmus X002.litmus X007.litmus X012.litmus X017.litmus X022.litmus X003.litmus X008.litmus X013.litmus X018.litmus X.conf

- use mcycles7 on a litmus test to extract the memory access pattern

$ mcycles7 @all

X000: Rfe PodRR Fre PodWR Fre

X001: Rfe PodRW Coe PodWR Fre

X002: Rfe PodRW Rfe PodRR Fre

.

.

- The output of mcycle7 can be piped into classify7 for family classication

  $ mcycles7 @all | classify7 -arch X86

  2+2W
  X021 -> 2+2W: PodWW Coe PodWW Coe
  3.2W
  X022 -> 3.2W: PodWW Coe PodWW Coe PodWW Coe
  3.LB
  X015 -> 3.LB: PodRW Rfe PodRW Rfe PodRW Rfe

To directly generate tests with family names based on the cycles used we have two methods

1. one can normalise tests, using normalised names by piping mcycle7 output into diyone7 with options -norm -num false

   $ mcycles7 @all | diyone7 -arch X86 -norm -num false -o normalized_op

1. Alternatively, one may instruct classify7 to produce output for diyone7.

   $ mcycles7 @all | classify7 -arch X86 -diyone | diyone7 -arch X86 -o normalized_op/

THANK YOU

S.Lochani Vilehya

43317