Compiler Testing via a Theory of Sound Optimisations in the C11/C++11 model

Robin Morisset
ENS Paris & INRIA

Pankaj Pawan
IIT Kanpur & INRIA

Francesco Zappa Nardelli
INRIA

Do C/C++ compilers correctly compile concurrent programs?

Shared memory initialisation: `int g1 = 1; int g2 = 0;`

GCC 4.7 -O2

Thread 1

```
void *th_1(void *p) {
  for (int l = 0; (l != 4); l++) {
    if (g1)
      return NULL;
    for (g2 = 0; (g2 >= 26); ++g2)
      ;
  }
}
```

Thread **2**

```
void *th_2(void *p) {
  g2 = 42;
  printf("%d\n",g2);
}
```

Output: 42

- Thread 1 returns before accessing g2
- The program is data-race free
- The C11 standard states that data-race free programs must exhibit only sequentially consistent behaviours
- 42 is the only sequentially consistent behaviour

Thread 1

```
 movl   g1(%rip), %edx   # load g1 into edx
 movl   g2(%rip), %eax   # load g2 into eax
 testl  %edx, %edx       # if g1!=0
 jne    .L2              # jump to .L2
 movl   $0, g2(%rip)
 ret
.L2:
 movl   %eax, g2(%rip)   # store eax into g2
 xorl   %eax, %eax       # store 0 into eax
 ret                     # return
```

Thread **2**

```
void *th_2(void *p) {
  g2 = 42;
  printf("%d\n",g2);
}
```

Output: 42 or 0

- The assembler for Thread 1 *saves* and *restores* g2
- Thread 1 can overwrite the update to g2 of Thread 2
- The compiled code can also print 0
- This is forbidden by the C11 standard

The compiler generated apparently correct sequential code for Thread 1, but did not respect the C11 memory model. The compiled code thus exhibits more behaviours than the source code. This is a *concurrency compiler bug*.

Our goal is to design and implement a technique to hunt for these compiler bugs.

Standard fuzzy testing of compilers cannot detect these because the generated code is correct in a sequential environment, and non-determinism makes naive extensions to fall short.

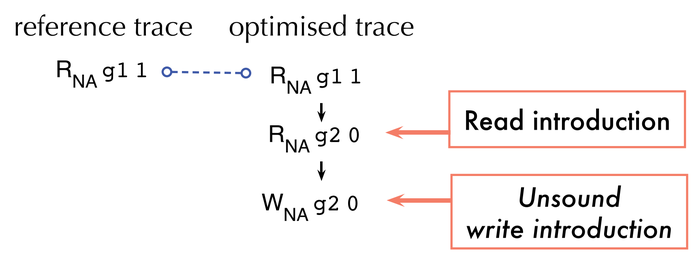
Hunting concurrency compiler bugs

Idea: C / C++ compilers support separate compilation; they can only apply optimisations sound in an arbitrary concurrent context

Contribution 1
Characterisation of optimisations sound in an arbitrary concurrent context

On the code above, the optimised trace exhibits two extra accesses, including one introduced memory write. Introduction of observable memory writes is provably unsound in the C11 memory model: the optimised trace cannot be justified from the reference trace by a sequence of correct transformations. We detect a concurrency compiler bug.

Execution traces for the compiler bug above.
Initially `g1=1` and `g2=0`.



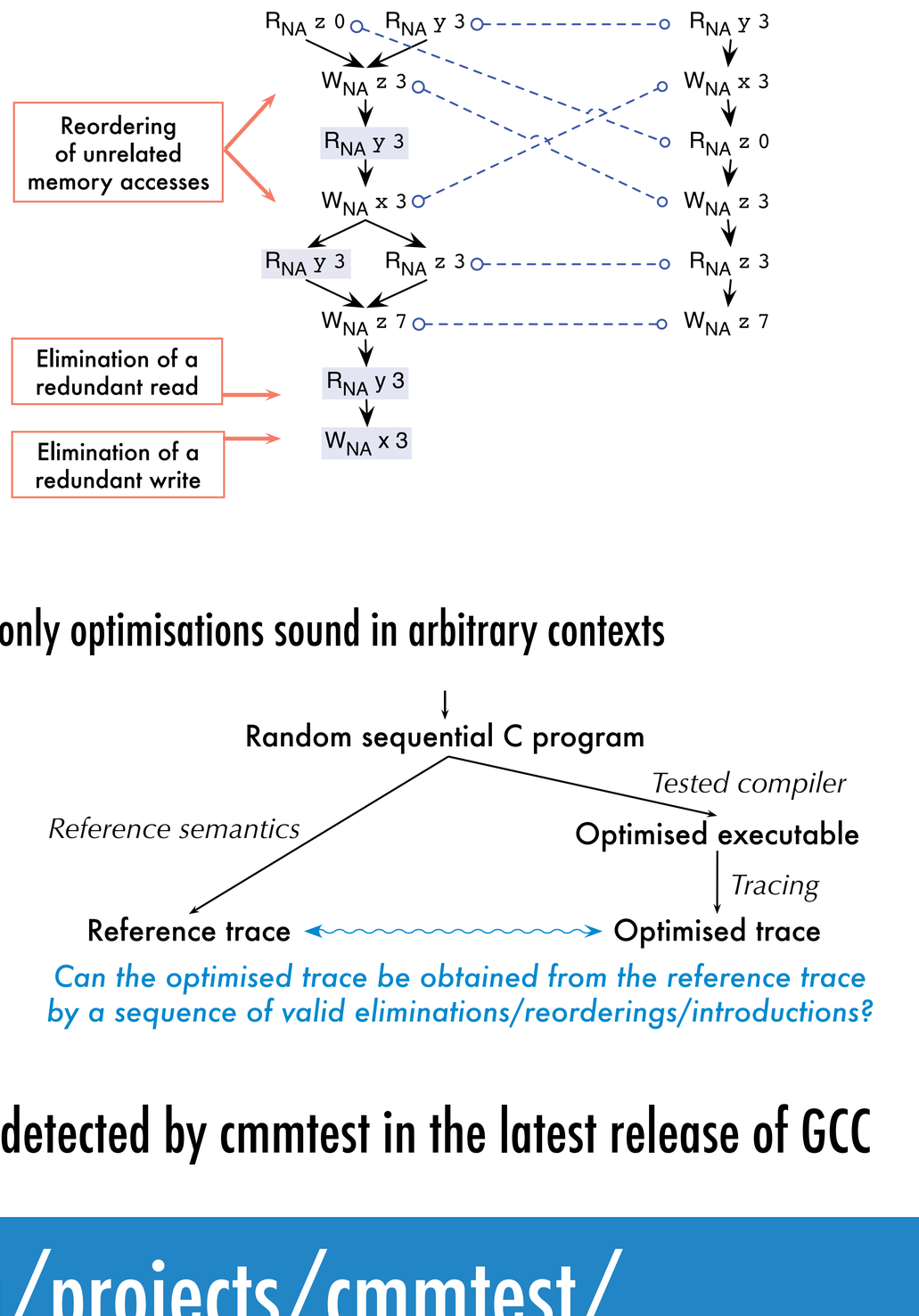
In general, it is possible to characterise which optimisations are correct in a concurrent setting by observing how they eliminate, reorder, or introduce, memory accesses in the traces of the sequential code with respect to a reference trace. As an example, the instance of loop-invariant code motion on the right eliminates *redundant* memory accesses and reorders *independent* accesses: it is provably correct.

```
for (i=0; i<2; i++) {
  z = z + y + i;
  x = y;
}
```

lim

```
t = y;
x = t;
for (i=0; i<2; i++) {
  z = z + t + i;
}
```

Execution traces for the unoptimised and optimised code, supposing an initial global state where `z=x=0` and `y=3` (while `t` and `i` are local variables).



Contribution 2
Design and implementation of a random-testing tool that checks if a compiler applies only optimisations sound in arbitrary contexts

The **cmmtest** tool that we designed and implemented puts this strategy at work:
- it generates a well-defined sequential C program using a modified version of CSmith
- it invokes the optimising compiler under test
- traces the global (potentially shared) memory accesses of the optimised code
- it compares the optimised trace against a reference trace for the source program, building on our theory of sound optimisations for the C11/C++11 memory model
- if the traces cannot be matched, it performs test-case reduction

Several concurrency compiler bugs and unexpected behaviours detected by cmmtest in the latest release of GCC

http://www.di.ens.fr/~zappa/projects/cmmtest/