Mechanised industrial concurrency specification: C/C++ and GPUs

Mark Batty
University of Kent
It is time for mechanised industrial standards

Specifications are written in English prose: this is insufficient

Write mechanised specs instead (formal, machine-readable, executable)

This enables verification, and can identify important research questions

Writing mechanised specifications is practical now
A case study: 
industrial concurrency specification
Shared memory concurrency

Multiple threads communicate through a shared memory
Shared memory concurrency

Multiple threads communicate through a shared memory

Most systems use a form of shared memory concurrency:
An example programming idiom

\[
data, \text{flag}, r \text{ initially zero}
\]

Thread 1:
\[
data = 1;
\]
\[
\text{flag} = 1;
\]

Thread 2:
\[
\text{while (flag==0)} 
\{
\}
\]
\[
r = \text{data};
\]

In the end \( r==1 \)

Sequential consistency: simple interleaving of concurrent accesses

Reality: more complex
An example programming idiom

data, flag, r initially zero

Thread 1:
data = 1;
flag = 1;

Thread 2:
while (flag==0)
{
};
r = data;

In the end r==1

Sequential consistency: simple interleaving of concurrent accesses

Reality: more complex
Relaxed concurrency

Memory is slow, so it is optimised (buffers, caches, reordering…)
e.g. IBM’s machines allow reordering of unrelated writes
(so do compilers, ARM, Nvidia…)

data, flag, r initially zero

Thread 1: Thread 2:
data = 1; while (flag==0)
flag = 1; {};
r = data;

In the end r==1

Sometimes, in the end r==0, a relaxed behaviour

Many other behaviours like this, some far more subtle, leading to trouble
Relaxed concurrency

Memory is slow, so it is optimised (buffers, caches, reordering…)

e.g. IBM’s machines allow reordering of unrelated writes

(so do compilers, ARM, Nvidia…)

data, flag, r initially zero

Thread 1: Thread 2:

flag = 1; while (flag==0)
data = 1; {};
r = data;

In the end r==1

Sometimes, in the end r==0, a relaxed behaviour

Many other behaviours like this, some far more subtle, leading to trouble
Relaxed behaviour leads to problems

Bugs in deployed processors
Many bugs in compilers
Bugs in language specifications
Bugs in operating systems

Power/ARM processors: unintended relaxed behaviour observable on shipped machines

[AMSS10]
Relaxed behaviour leads to problems

Bugs in deployed processors
Many bugs in compilers
Bugs in language specifications
Bugs in operating systems

Errors in key compilers (GCC, LLVM): compiled programs could behave outside of spec.

[MPZN13, CV16]
Relaxed behaviour leads to problems

Bugs in deployed processors
Many bugs in compilers
Bugs in language specifications
Bugs in operating systems

The C and C++ standards had bugs that made unintended behaviour allowed.

More on this later.

[BOS+11, BMN+15]
Relaxed behaviour leads to problems

- Bugs in deployed processors
- Many bugs in compilers
- Bugs in language specifications
- Bugs in operating systems

Confusion among operating system engineers leads to bugs in the Linux kernel

[McK11, SMO+12]
Relaxed behaviour leads to problems

Bugs in deployed processors
Many bugs in compilers
Bugs in language specifications
Bugs in operating systems

Current engineering practice is severely lacking!
Vague specifications are at fault

Relaxed behaviours are subtle, difficult to test for and often unexpected, yet allowed for performance.

Specifications try to define what is allowed, but English prose is untestable, ambiguous, and hides errors.
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Build mechanised executable formal models of specifications

[AFI+09,BOS+11,BDW16]
[FGP+16,LDGK08,OSP09]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Provide tools to simulate the formal models, to explain their behaviours to non-experts

Provide reasoning principles to help in the verification of code

[BOS+11, SSP+, BDG13]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Run a battery of tests to understand the observable behaviour of the system and check it against the model

[AMSS’11]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Explicitly stated design goals should be proved to hold

[BMN+15]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Test to find the relaxed behaviours introduced by compilers and verify that optimisations are correct

[MPZN13, CV16]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

Specifications should be fixed when problems are found

Test suites can ensure conformance to formal models

[B11]
A diverse and continuing effort

Modelling of hardware and languages
Simulation tools and reasoning principles
Empirical testing of current hardware
Verification of language design goals
Test and verify compilers
Feedback to industry: specs and test suites

I will describe my part:
The C and C++ memory model
Acknowledgements

M. Dodds  A. Gotsman  K. Memarian  K. Nienhuis  S. Owens

J. Pichon-Pharabod  S. Sarkar  P. Sewell  T. Weber
C and C++

The medium for system implementation

Defined by WG14 and WG21 of the International Standards Organisation

The ’11 and ’14 revisions of the standards define relaxed memory behaviour
C and C++

The medium for system implementation

Defined by WG14 and WG21 of the International Standards Organisation

The '11 and '14 revisions of the standards define relaxed memory behaviour

We worked with the ISO, formalising and improving their concurrency design
C++11 concurrency design

A contract with the programmer: they must avoid data races, two threads competing for simultaneous access to a single variable

\[ \text{data initially zero} \]

Thread 1: \[ \text{data} = 1; \]
Thread 2: \[ r = \text{data}; \]

Beware:
Violate the contract and the compiler is free to allow anything: catch fire!
C++11 concurrency design

A contract with the programmer: they must avoid **data races**, two threads competing for simultaneous access to a single variable

**data** initially zero

Thread 1: Thread 2:

\[ \text{data} = 1; \quad r = \text{data}; \]

**Beware:**

Violate the contract and the compiler is free to allow anything: **catch fire!**
C++11 concurrency design

A contract with the programmer: they must avoid data races, two threads competing for simultaneous access to a single variable.

Data initially zero

Thread 1: Thread 2:

```
data = 1;
r = data;
```

Beware:
Violate the contract and the compiler is free to allow anything: catch fire!

Atomics are excluded from the requirement, and can order non-atomics, preventing simultaneous access and races.
C++11 concurrency design

A contract with the programmer: they must avoid data races, two threads competing for simultaneous access to a single variable

\[
data, r, \text{atomic flag}, \text{initially zero}\]

Thread 1:
\[
data = 1; \\
flag = 1;
\]

Thread 2:
\[
\text{while (flag==0)} \\
\{\}; \\
r = data;
\]

Beware:
Violate the contract and the compiler is free to allow anything: catch fire!

**Atomics** are excluded from the requirement, and can order non-atomics, preventing simultaneous access and races
The design is complex but the standard claims a powerful simplification:

C++11/14: §1.10p21
It can be shown that programs that correctly use mutexes and
memory_order_seq_cst operations to prevent all data races and use no
other synchronization operations behave [according to] “sequential
consistency”.

This is the central design goal of the model, called DRF-SC
Implicit design goals

Compilers like GCC, LLVM map C/C++ to pieces of machine code

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Power</th>
<th>ARM</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load acquire</td>
<td>ld; cmp; bc; isync</td>
<td>ldr; dmb</td>
<td>MOV (from memory)</td>
</tr>
</tbody>
</table>

Each mapping should preserve the behaviour of the original program
We formalised a draft of the standard

A mechanised formal model, close to the standard text

**C++11 standard §1.10p12:**
An evaluation A happens before an evaluation B if:

• A is sequenced before B, or
• A inter-thread happens before B.

The implementation shall ensure that no program execution demonstrates a cycle in the “happens before” relation.

**The corresponding formalisation:**

```plaintext
let happens_before sb ithb = sb ∪ ithb

let consistent hb hb =
  isIrreflexive (transitiveClosure hb)
```
Communication with WG21 and WG14

Issues were discussed in N-papers and Defect Reports
Major problems fixed, key properties verified

DRF-SC:
The central design goal, was false, the standard permitted too much
Fixed the model and then proved (in HOL4) that the goal is now true
Fixes were incorporated, pre-ratification, and are in C++11/14

Compilation mappings:
Efficient x86, Power mappings are sound [BOS+11,BMO+12,SMO+12]

Reasoning:
Developed a reasoning principle for proving programs correct [BDO13]
A fundamental problem uncovered

x, y, r1, r2 initially zero

// Thread 1 // Thread 2
r1 = x; r2 = y;
if(r1==1) y = 1; if(r2==1) x = 1;

Can we observe r1==1, r2==1 at the end?
A fundamental problem uncovered

x, y, r1, r2 initially zero

// Thread 1
r1 = x;
if(r1==1) y = 1;

// Thread 2
r2 = y;
if(r2==1) x = 1;

Can we observe r1==1, r2==1 at the end?

The write of y is dependent on the read of x
A fundamental problem uncovered

x, y, r1, r2 initially zero

// Thread 1                      // Thread 2
r1 = x;                        r2 = y;
if(r1==1) y = 1;               if(r2==1) x = 1;

Can we observe r1==1, r2==1 at the end?

The write of y is dependent on the read of x

The write of x is dependent on the read of y
A fundamental problem uncovered

x, y, r1, r2 initially zero

// Thread 1           // Thread 2
r1 = x;              r2 = y;
if(r1==1) y = 1;     if(r2==1) x = 1;

Can we observe r1==1, r2==1 at the end?

The write of y is dependent on the read of x

The write of x is dependent on the read of y

This will never occur in compiled code, and ought to be forbidden
A fundamental problem uncovered

\[ x, y, r1, r2 \text{ initially zero} \]

// Thread 1 // Thread 2
r1 = x; \hspace{1cm} \text{r2 = y;}
if(r1==1) y = 1; \hspace{1cm} \text{if(r2==1) x = 1;}

Can we observe r1==1, r2==1 at the end?

The write of y is dependent on the read of x

The write of x is dependent on the read of y

This will never occur in compiled code, and ought to be forbidden

"[Note: [...] However, implementations should not allow such behavior. — end note]"

ISO: notes carry no force, and “should” imposes no constraint, so yes!
A fundamental problem uncovered

The write of \( y \) is dependent on the read of \( x \)

The write of \( x \) is dependent on the read of \( y \)

This will never occur in compiled code, and ought to be forbidden

“[Note: […] However, implementations should not allow such behaviour — end note]”

ISO: notes carry no force, and “should” imposes no constraint, so yes!

Why? Dependencies are ignored to allow dependency-removing optimisations

Should respect the left-over dependencies

We have proved that no fix exists in the structure of the current specification

This identifies a difficult research problem
Timing was everything

Achieved direct impact on the standard

C++11 was a major revision, so the ISO was receptive to change

Making this work was partly a social problem
GPU concurrency
Acknowledgements

J. Alglave  B. Beckmann  A. Donaldson  G. Gopalakrishnan

J. Ketema  D. Poetzl  T. Sorensen  J. Wickerson
Alternate design path: throughput over latency, thousands of threads

Forecast for use in critical applications: AUDI-Nvidia Drive Partnership

Hardware and specs under rapid development (computing only 10 years old)

An opportunity for lightweight verification at the design phase
Many fronts of progress

Empirical testing of GPU behaviour
Refinement of an AMD GPU design
Formalisation of OpenCL concurrency
Direct engagement with Nvidia

Observed ‘surprising’ relaxed behaviours that break algorithms in the literature

e.g. Cederman and Tsigas queue

Same for programming idioms in vendor-supported tutorials

[ABD+15]
Many fronts of progress

Empirical testing of GPU behaviour
Refinement of an AMD GPU design
Formalisation of OpenCL concurrency
Direct engagement with Nvidia

Direct collaboration with AMD
Modelled a prototype GPU design
Found bugs, refined the design
Early concept, so change is cheap

[WBDB15]
Many fronts of progress

Empirical testing of GPU behaviour
Refinement of an AMD GPU design
Formalisation of OpenCL concurrency
Direct engagement with Nvidia

OpenCL is an extension of C11 to CPU-GPU systems
Extended C11 model to OpenCL
Verified AMD compiler mapping
[BDW16,WBDB15]
Many fronts of progress

<table>
<thead>
<tr>
<th>Empirical testing of GPU behaviour</th>
<th>Helping to develop internal specification for next-gem architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refinement of an AMD GPU design</td>
<td>Verifying compilation mapping in HOL4 theorem prover</td>
</tr>
<tr>
<td>Formalisation of OpenCL concurrency</td>
<td></td>
</tr>
<tr>
<td>Direct engagement with Nvidia</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Mechanised industrial specification is practical and can have major impact

It can guide us to future research questions

This is a necessary step in formal verification

Formalisation can inform good hardware and language specifications
Bibliography


[LDGK08] G. Li, M. Delisi, G. Gopalakrishnan, and R. M. Kirby. Formal specification of the mpi-2.0 standard in tla+. PPoPP’08


