Risotto

A Dynamic Binary Translator for Weak Memory Model Architectures

Redha Gouicem*, Dennis Sprokholt*, Jasper Ruehl,
Rodrigo C. O. Rocha, Tom Spink, Soham Chakraborty, Pramod Bhatotia

Technical University of Munich

TU Delft

The University of Edinburgh

University of St Andrews

Towards a heterogeneous CPU landscape

New emerging architectures challenge the x86 dominance

ARM

RISC-V

Major vendors build custom Arm CPUs

Azure

Google Cloud

aws

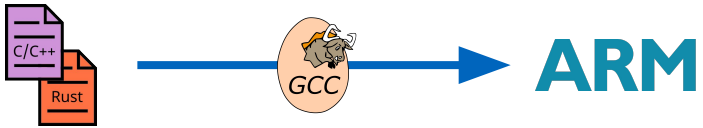
Major cloud providers offer Arm-based instances

What does this mean for legacy applications?

Legacy applications on new architectures

Source code available? ✔

No x86-specific assembly code? ✔



Just compile and run! ✅

You only have pre-built binaries?

Your codebase contains x86 assembly,

requiring complicated porting?

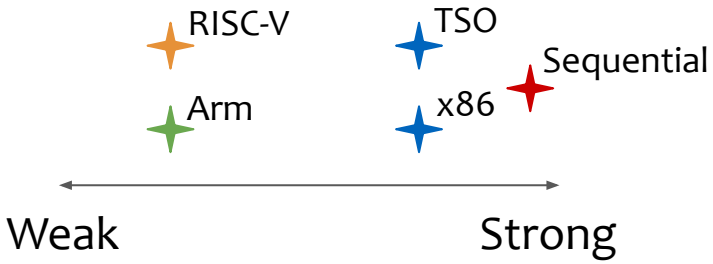
Binary translation is the way to go!

Binary translation and memory models

Translated binaries **must** have the **same** behaviour as on the source architecture

What about the architectural differences between source and target architectures?

Memory ordering model: reordering rules for memory accesses



X = Y = 0

(*thread 1*)	(*thread 2*)
X = 1 Y = 1	if (Y == 1) **assert(X == 1)**

On x86, assert **always succeeds**

On Arm, assert **can fail** if instructions are re-ordered

The source memory model must be enforced

The cost of correctness…

Enforcing a memory model ⇒ inserting *fences* next to memory operations

QEMU, a state-of-the-art binary translator, tries to *enforce the x86 memory model on Arm* by inserting ***strong fences*** before every memory operation:

e.g., *store* → *full-fence ; store*

load → *load-fence ; load*

But fences have a performance cost:

48% of the execution time of PARSEC and Phoenix benchmarks is spent executing memory fences!

…without the correctness

We still find correctness bugs in QEMU with atomic operations

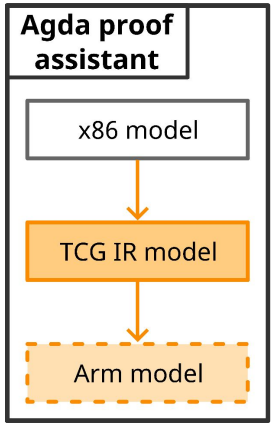
(see the paper for full details)

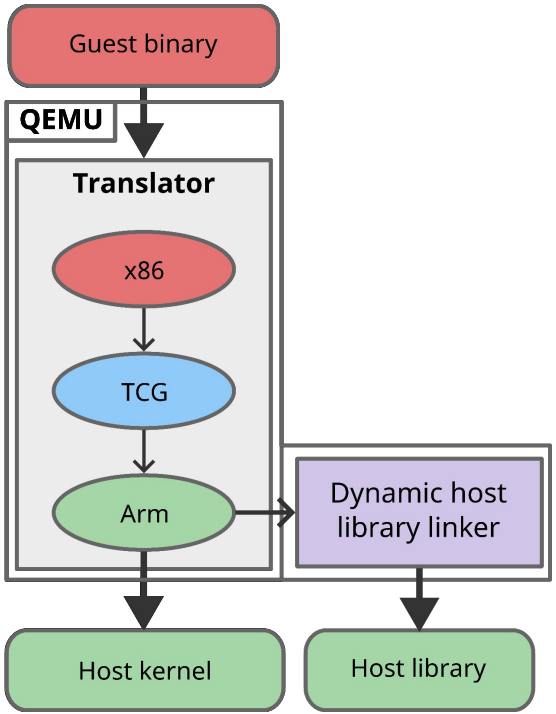
How can we maximise the performance of binary translation while ensuring the execution correctness?

Risotto: A Dynamic Binary Translator for Weak Memory Model Architectures

We base our design on QEMU and its intermediate representation, *TCG IR*

1. **Correctness**
   - ➢ Precise memory mappings
   - ➢ Formal correctness proofs
2. **Performance**
   - ➢ Optimised code generation
   - ➢ Maximising native code use with shared libraries

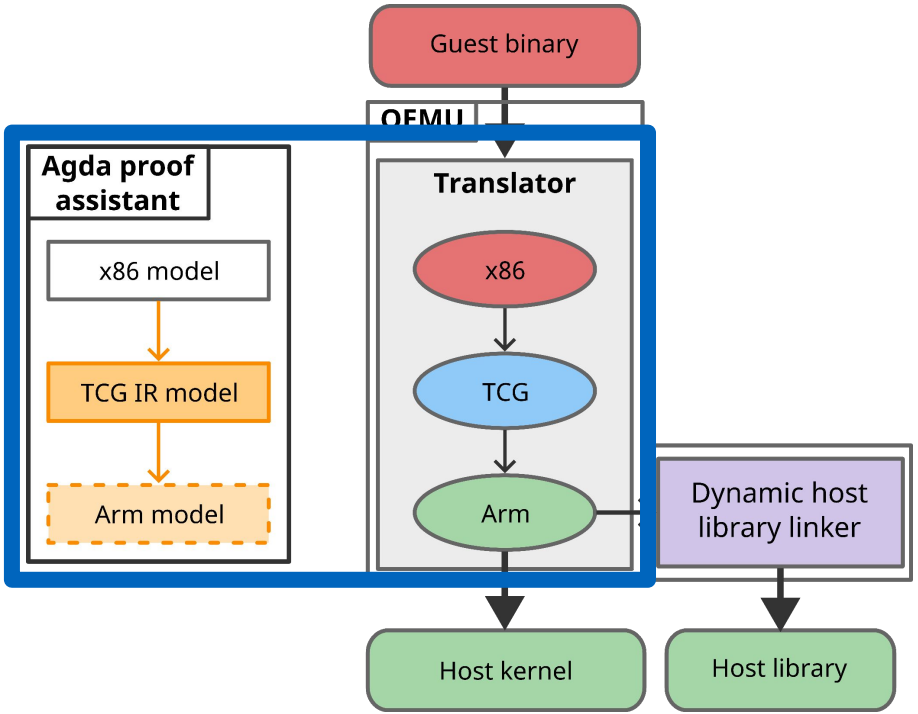




Risotto: A Dynamic Binary Translator for Weak Memory Model Architectures

We base our design on QEMU and its intermediate representation, *TCG IR*

1. **Correctness**
   - ➢ *Precise memory mappings*
   - ➢ *Formal correctness proofs*
2. **Performance**
   - ➢ Optimised code generation
   - ➢ Maximising native code use with shared libraries



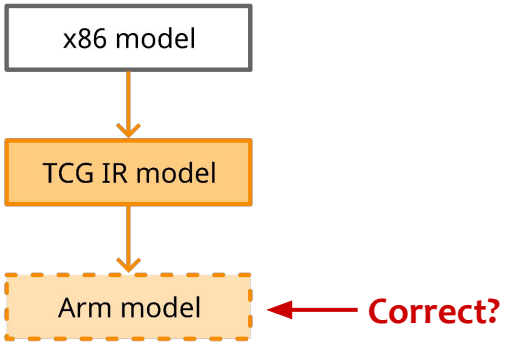
Precise memory mappings

We propose **precise** memory mappings to enforce the x86 memory model

*i.e., every fence is necessary **and** sufficient*

store	→	full-fence	;	store		store-fence	;	store
load	→	load-fence	;	load		load	;	load-fence

We formally verify the correctness of our mappings with the Agda theorem prover



Fixing the Arm model

`casal`: Compare-And-Swap Acquire Release

Our proofs show that the *official* Arm memory model
does not enforce the full fence!

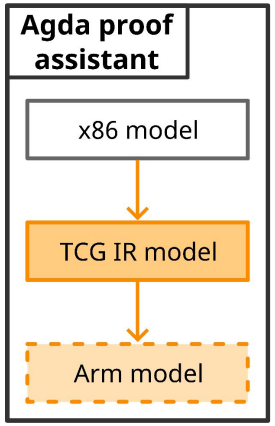
We fix the official Arm model and submit the issue

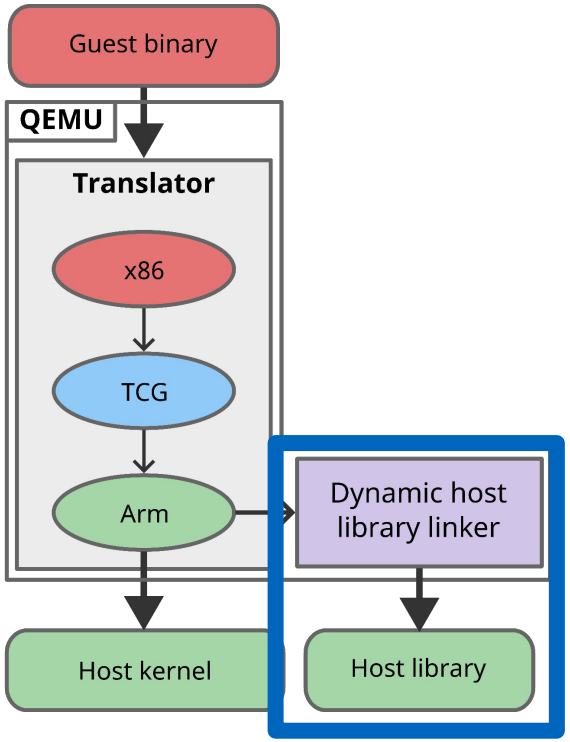
The official model was fixed!

Risotto: A Dynamic Binary Translator for Weak Memory Model Architectures

We base our design on QEMU and its intermediate representation, *TCG IR*

1. **Correctness**
   - ➢ Precise memory mappings
   - ➢ Formal correctness proofs
2. **Performance**
   - ➢ Optimised code generation
   - ➢ *Maximising native code use with shared libraries*

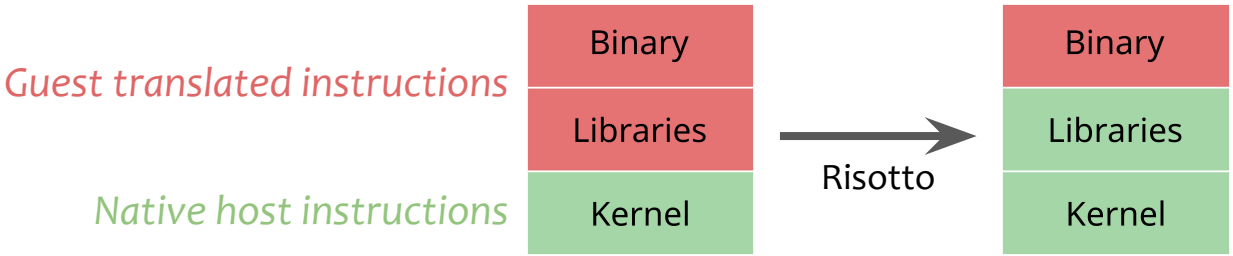




Maximising native code usage

Native binaries *(from source code to host)* are **faster** than translated binaries

→ maximising the native binary base is a good target for optimisation



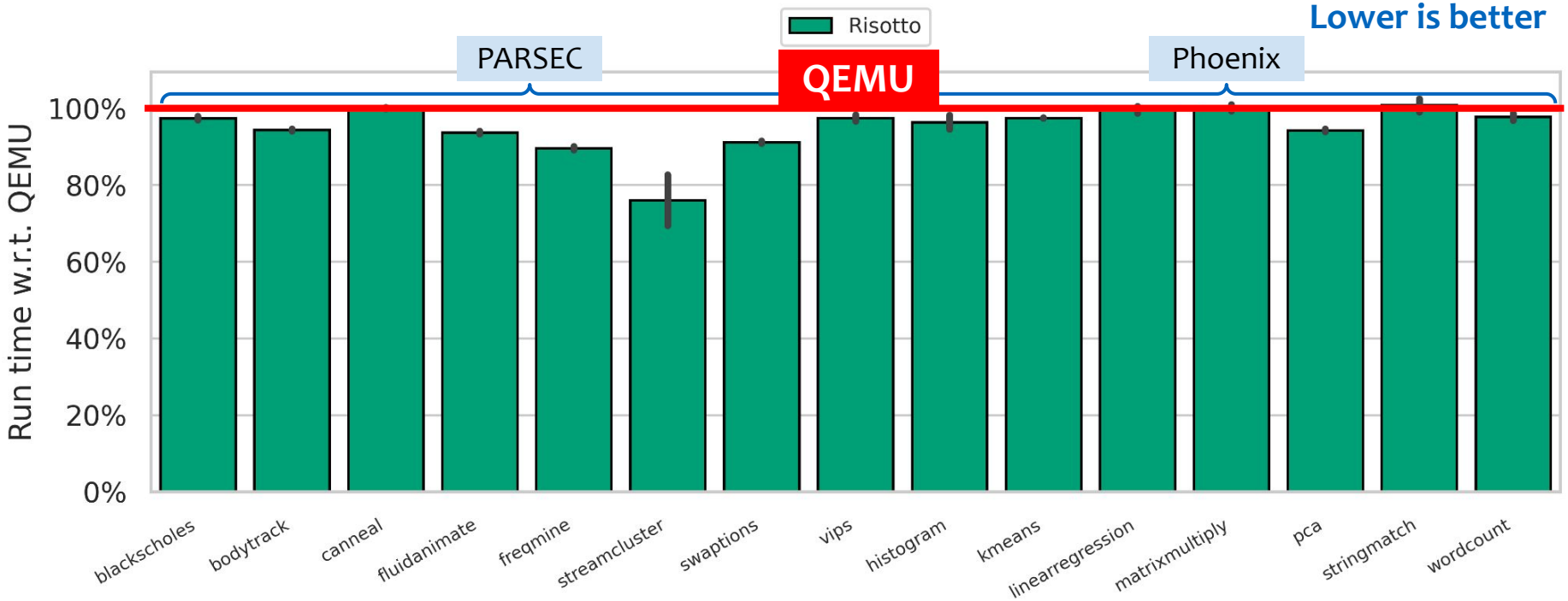
We introduce a **dynamic host library linker** that:

- detects calls to shared library functions
- patches the target jump address to invoke the native host library

Evaluation

1. Do Risotto mappings outperform QEMU mappings?
   → Risotto vs QEMU (incorrect)
   → Benchmarks: PARSEC and Phoenix
2. Does our host library linker achieve native performance for shared libraries?
   → Risotto vs QEMU
   → Benchmarks: openssl, math and sqlite

Evaluation: Memory Mappings

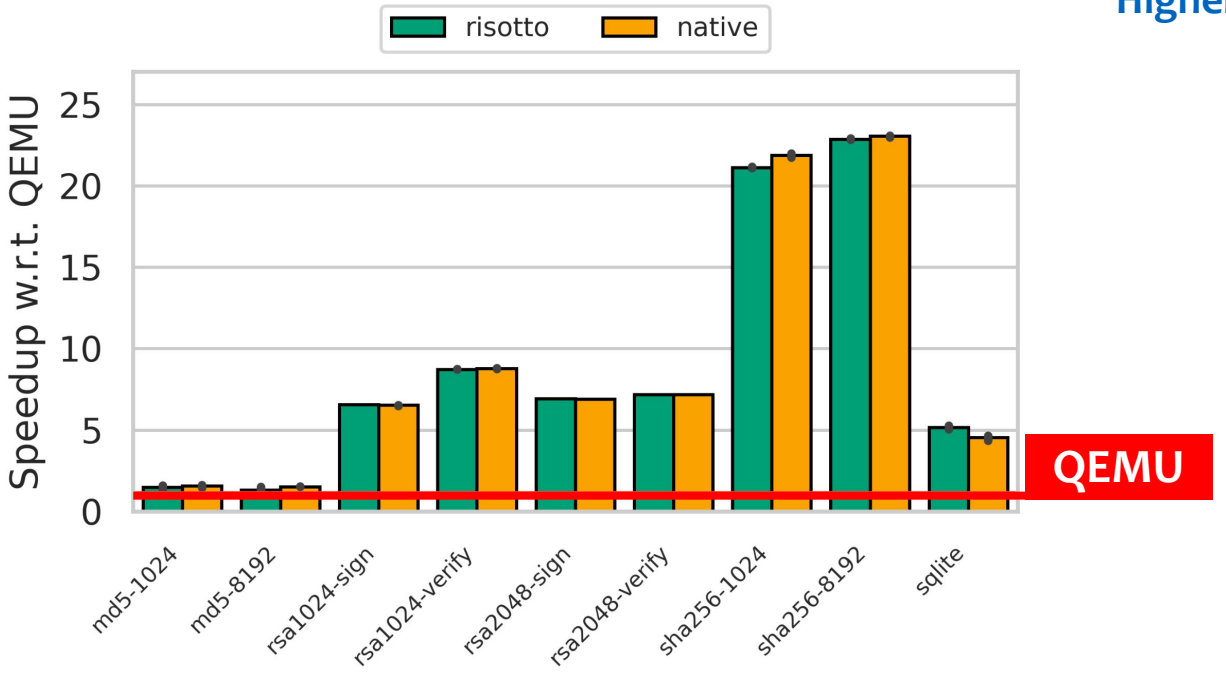


Risotto vs QEMU: 6.7% faster on average

Evaluation: Dynamic Host Library Linker

Openssl and sqlite speedtests

Higher is better



Risotto matches native execution

Conclusion

We propose **Risotto**, a dynamic binary translator for weak memory model architectures

- **Correct** enforcement of the x86 memory model on Arm hosts
  - *Formal model of QEMU's intermediate representation, TCG*
  - *Precise memory mappings from x86 to Arm via TCG*
  - *Proof of correctness in Agda*
  - *Fix to the official Arm memory model*
- **Improved performance** while being correct
  - *Optimised generated code → 6.7% faster than QEMU on average*
  - *Dynamic host library linker → matches native library performance*

Check out the paper here!