Adventures in Porting Rust

Sarah Harris, Simon Cooksey, Michael Vollmer, Mark Batty

University of Kent

April 2024
Rust for Morello

We have a port of the Rust compiler for Morello!

- Rust 1.56.0 (update to 1.72.1 coming)
- Target: Morello CHERI BSD
- Working compiler and standard library
- Builds real code (108k lines of benchmarks)

Website:
https://www.cs.kent.ac.uk/people/staff/mjb211/rust/index.htm
What I’ll be talking about:

- Rust
- usize
- const
- Sizes
- Vtables
- panic!()
Why Rust?

Rust and CHERI fit together well

- Safety (especially memory safety) focused language
- Widely used
- CHERI protects `unsafe` sections (mostly)

```rust
extern "C" {
    fn iffy_c_library(data: *const u8);
}

fn main() {
    unsafe { iffy_c_library("Hello C\0".as_ptr()); }
}
```
The Problem: `usize`

`usize` is “The pointer-sized integer type.”¹

- Assumption: pointers are just an address
- Multiple solutions for CHERI
- Ours: `usize` is 64-bit
- It’s Complicated

¹Rust standard library documentation
**const Expressions**

**const** expressions happen at compile time

- Allows large subset of Rust
- Memory layout must match run time
- What about capability metadata?
- Our answer: leave uninitialised gaps

```
const POINTER: *const u8 = "a string".as_ptr();
```

```
struct Data { a: u64, b: 64, c: *const (), d: u32 }
const DATA: Data = Data{ a: 1, b: 2, c: POINTER, d: 3 };
```

**Morello**

<table>
<thead>
<tr>
<th>a: u64</th>
<th>b: u64</th>
<th>c: *const ()</th>
<th>c: u32</th>
</tr>
</thead>
</table>

**Compiler (x86)**

<table>
<thead>
<tr>
<th>a: u64</th>
<th>b: u64</th>
<th>c: *const ()</th>
<th>c: u32</th>
</tr>
</thead>
</table>
Data Size

Values in compiler need size information

- We now need size of data \textit{and} size in memory
- size becomes data\_size and memory\_size
- Memory layout logic needs to record both sizes
- Propagate change through compiler (~230 files)
  Scalar, ScalarInt, AllocRange, CodegenContext, TargetDataLayout, Layout...

![Memory Layout Diagram](image-url)
Vtables

References to types known only at run-time use vtables

```rust
trait DataTrait { fn reticulate(); }
fn dynamic_method_call(data: &dyn DataTrait) {
    data.reticulate();
}
```

▶ Stored as array of slots, some numbers, some pointers
▶ Assumes `usize` and pointers have same sizes
▶ Current solution: make every slot pointer sized

**Vtable (x86)**
- drop in place
- size (usize)
- align (usize)
- method info

**Vtable (Morello)**
- drop in place (function)
- size (usize)
- align (usize)
- method info

always present

type-specific
Don’t (always) **panic!()**

`panic!()` emits error and terminates, but can be caught

```rust
fn fatal() { panic!("Oh no!") }
std::panic::catch_unwind(fatal);
```

- Borrows plumbing for exceptions
- Relies on reading tables of handlers
- Morello uses a custom encoding
- Extend Rust’s reader to avoid messy crashes

<table>
<thead>
<tr>
<th>Stack</th>
<th>_gcc_except_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>panic!()</td>
<td>0x11000-0x11523</td>
</tr>
<tr>
<td>fatal()</td>
<td>0x11524-0x1153f</td>
</tr>
<tr>
<td>std::panic::catch_unwind()</td>
<td>0x11540-0x11567</td>
</tr>
<tr>
<td>main()</td>
<td>0x11568-0x1157f</td>
</tr>
<tr>
<td>std::rt::lang_start()</td>
<td>...</td>
</tr>
</tbody>
</table>
Questions, Perhaps Even Answers...

That’s (some of) how you port Rust to Morello!

- Rust 1.56.0 compiler (1.72.1 on the way)
- Targets CHERI BSD on Morello
- Includes standard libraries

Compiler source, binaries:
https://www.cs.kent.ac.uk/people/staff/mjb211/rust/index.htm

ECOOP paper: