Improving Performance of Provable Computations Using Rust

How we reimplemented the Cairo VM using Rust

Federica Moletta, Herman Obst
LambdaClass
Who are we?
Context
What is StarkNet?

StarkNet is a ZK-Rollup

Transactions -> App contracts

Proof + State update -> StarkNet Contract

App contracts -> Applications

Ethereum

Developers

Users
What is **StarkNet**?

Cost of verification $\sim \log n$
What is **StarkNet**?

Cost of verification $\sim \log n$

$n \rightarrow \infty \Rightarrow \text{Tsx fee} \rightarrow 0$
What is a **STARK**?

*Scalable Transparent Argument of Knowledge*

- STARKS are a specific type of Zero Knowledge Proofs
- ZKP allow us to prove the veracity of a statement without revealing any information beyond the fact that the statement is true
What is **Cairo**?

- Programming language for writing provable programs.
- Running a program produces a trace.
- The trace can be sent to a **prover** to generate a STARK proof.
Cairo VM

Source code

Cairo Compiler

Compiled Program

Cairo VM

Trace
Characteristics of Cairo VM Architecture: Memory Model

- Program Segment
- Execution Segment
- Builtin Segments
- User segments

- Program Segment
- Execution Segment
- Builtin Segments
- User segments
## Relocation Process: Computing each Segment Size

<table>
<thead>
<tr>
<th>Segment Index</th>
<th>0:0</th>
<th>0:1</th>
<th>0:2</th>
<th>0:3</th>
<th>0:4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.41686662805676031</td>
<td>2</td>
<td>5.189976364521848832</td>
<td>4</td>
<td>2.345108766317314046</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Index</th>
<th>1:0</th>
<th>1:1</th>
<th>1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2:0</td>
<td>3:0</td>
<td>4</td>
</tr>
</tbody>
</table>
Relocation Process: Calculating each Segment Base

<table>
<thead>
<tr>
<th>Prev Segment Base</th>
<th>Prev Segment Size</th>
<th>Segment Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:0 74168662805676031</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0:1 5189976364521848832</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>0:2 2345108766317314046</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

1 + 5 = 6
6 + 3 = 9
9 + 0 = 9
Relocation Process: Relocating each Address

Segment Base + Offset = Relocated Address

<table>
<thead>
<tr>
<th>Segment Base</th>
<th>Offset</th>
<th>Relocated Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:0</td>
<td>74168662805676031</td>
<td></td>
</tr>
<tr>
<td>0:1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0:2</td>
<td>5189976364521848832</td>
<td>2</td>
</tr>
<tr>
<td>0:3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>0:4</td>
<td>2345108766317314046</td>
<td>4</td>
</tr>
<tr>
<td>1:0</td>
<td>2:0</td>
<td>6</td>
</tr>
<tr>
<td>1:1</td>
<td>3:0</td>
<td>7</td>
</tr>
<tr>
<td>1:2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
### Relocation Process: Relocating each Address

Segment Base + Offset = Relocated Address

<table>
<thead>
<tr>
<th>Segment Base</th>
<th>Offset</th>
<th>Relocated Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0:0</td>
<td>74168662805676031</td>
</tr>
<tr>
<td></td>
<td>0:1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0:2</td>
<td>518997636452184832</td>
</tr>
<tr>
<td></td>
<td>0:3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0:4</td>
<td>2345108766317314046</td>
</tr>
<tr>
<td>1</td>
<td>1:0</td>
<td>2:0</td>
</tr>
<tr>
<td></td>
<td>1:1</td>
<td>3:0</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Relocation Process: Relocating each Address

<table>
<thead>
<tr>
<th>Segment Base + Offset</th>
<th>Relocated Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:0 74168662805676031</td>
<td>1:0 74168662805676031</td>
</tr>
<tr>
<td>0:1 2</td>
<td>2</td>
</tr>
<tr>
<td>0:2 5189976364521848832</td>
<td>3:0 5189976364521848832</td>
</tr>
<tr>
<td>0:3 4</td>
<td>4</td>
</tr>
<tr>
<td>0:4 2345108766317314046</td>
<td>5:0 2345108766317314046</td>
</tr>
<tr>
<td>1:0 74168662805676031</td>
<td>6:0 9</td>
</tr>
<tr>
<td>1:1 3:0</td>
<td>7:0 9</td>
</tr>
<tr>
<td>1:2 4</td>
<td>8:0 4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
Characteristics of Cairo VM Architecture: **Registers**

- Program Segment
- Execution Segment
- Built-in Segments
- User segments
Characteristics of Cairo VM Architecture: *Registers*

Program Segment

- "0x40780017fff7fff",
- "0x1",
- "0x208b7fff7fff7ffe",
- "0x400380007ffc7ffd",
- "0x480680017fff8000",
- "0xff00ff00ff00ff00ff00ff",
- "0x400280017ffc7fff",
- "0x400280017ffc7fff",
- "0x480680017fff8000"

Execution Segment
Characteristics of Cairo VM Architecture: **Registers**

<table>
<thead>
<tr>
<th>Program Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Segment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FP</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:5</td>
<td>0:94</td>
</tr>
<tr>
<td>7:0</td>
<td>34623634663146736</td>
</tr>
<tr>
<td>598249824422424658356</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1:5</td>
</tr>
<tr>
<td>0:106</td>
<td>0</td>
</tr>
<tr>
<td>4:0</td>
<td>598249824422424658356</td>
</tr>
<tr>
<td>34623634663146736</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Main execution loop: Step
Trace Generation

```plaintext
TraceEntry {
    pc: Relocatable {
        segment_index: 0, 
        offset: 0 
    },
    ap: Relocatable {
        segment_index: 1, 
        offset: 2 
    },
    fp: Relocatable {
        segment_index: 1, 
        offset: 2 
    }
}

TraceEntry {
    ap: 4,  
    fp: 4, 
    pc: 1
}
```
Features of Cairo: **Builtins**

- Low level optimizations
- Integrated into the core loop of the VM
- Allow otherwise expensive computations to be performed
Main execution loop: **Step with Builtins**

1. **Compute Operands**
2. **Deduce Operands**
3. Add current register values to Trace
4. Update Registers
5. Fetch next instruction
6. Decode instruction
7. **Step with Builtins**
Main execution loop: **Step with Builtins**

```rust
def deduce_memory_cell(
    &mut self,
    addr: &Relocatable,
    memory: &Memory,
) -> Result<MaybeRelocatable> {
    let x = memory[addr - 1]
    let y = memory[addr - 2]
    return pedersen_hash(x, y)
}
```
func hash2{hash_ptr: HashBuiltin*}(x, y) -> (result: felt) {
    hash_ptr.x = x;
    hash_ptr.y = y;
    let result = hash_ptr.result;
    let hash_ptr = hash_ptr + HashBuiltin.SIZE;
    return (result=result);
}
Features of Cairo: Hints

- Python code embedded into a Cairo program
- Can access and modify the VM’s state
- Can also interact with each other through execution scopes

```c
// Allocates a new memory segment.
func alloc() -> (ptr: felt*) {
    %{ memory[ap] = segments.add() %}
    ap += 1;
    return (ptr=cast([ap - 1], felt*));
}
```
Features of Cairo: **Hints**

**Execution Scopes:**

- Stack of dictionaries which hold variables created inside hints.
- Hints can pop and push scopes (enter & exit).
- Multiple hints can access the same scope

```c
// Copies len field elements from src to dst.
func memcpy(dst: felt*, src: felt*, len) {
  struct LoopFrame {
    dst: felt*,
    src: felt*,
  }

  if (len == 0) {
    return ();
  }

  %{ vm_enter_scope({'n': ids.len}) %}
  tempvar frame = LoopFrame(dst=dst, src=src);

  loop:
  let frame = [cast(ap - LoopFrame.SIZE, LoopFrame*)];
  assert [frame.dst] = [frame.src];

  let continue_copying = [ap];
  // Reserve space for continue_copying.
  let next_frame = cast(ap + 1, LoopFrame*);
  next_frame.dst = frame.dst + 1, ap++;
  next_frame.src = frame.src + 1, ap++;
  %{ n -= 1
    ids.continue_copying = 1 if n > 0 else 0
  %}
  static_assert next_frame + LoopFrame.SIZE == ap + 1;
  jmp loop if continue_copying != 0, ap++;
  // Assert that the loop executed len times.
  len = cast(next_frame.src, felt) - cast(src, felt);

  %{ vm_exit_scope(src, felt) - cast(src, felt) %}
  return ();
```
Section 2

Hints in Cairo-rs
Why Rust?

- Performance
- Memory Safety
- Great Community
Hints in Cairo-rs:
How we began implementing hints in Rust

```rust
fn execute_hint(
    vm: &mut VM,
    exec_scopes &mut ExecutionScopes
    hint_data: HintProcessorData,
){
    match hint_data.code {
        ADD_SEGMENT => add_segment(vm),
        IS_NN => is_nn(vm, &hint_data),
        IS_LE_FELT => is_le_felt(vm, &hint_data),
        ASSERT_LE_FELT => assert_le_felt(vm, &hint_data),
        ASSERT_250_BITS => assert_250_bit(vm, &hint_data),
        IS_POSITIVE => is_positive(vm, &hint_data),
    }
}
```
Hints in Cairo-rs:
How we began implementing hints in Rust

"memory[ap] = segments.add()"

```rust
pub fn add_segment(vm: &mut VirtualMachine) {
    vm.memory.insert(vm.ap, vm.segments.add())
}
```
Hints in Cairo-rs:
How we began implementing hints in Rust

**Pros**
- Easy to integrate as no new tools were needed
- Better performance

**Cons**
- Need to watch out and modify our implementation if hints change
- Not extensible, as any new hints need to be implemented separately
Hints in Cairo-rs:
How we began integrating python hints with PyO3

Why PyO3?

- Provides Rust bindings for Python
- Allows sharing the VM state with a python context
- Allows python to modify the VM state
- Allow us to define a strict interface through `pyclasses` & `pymethods`
Python Hints: Modifying VM Memory through Hints

```rust
#[pyclass(unsendable)]
pub struct PyMemory {
    vm: Rc<RefCell<VirtualMachine>>,
}

#[pymethods]
impl PyMemory {
    #[getter]
    pub fn __getitem__(&self, key: &PyRelocatable, py: Python) -> PyResult<PyObject> {
        self.vm.memory.get(key).to_object(py))
    }

    #[setter]
    pub fn __setitem__(&self, key: &PyRelocatable, value: PyMaybeRelocatable) -> PyResult<()> {
        self.vm.memory.insert(&key, value)
    }

```
Python Hints:
Modifying Cairo Variables through Hints

```rust
#[pyclass(unsendable)]
pub struct PyIds {
    vm: Rc<RefCell<VirtualMachine>>,
    references: HashMap<String, HintReference>,
    ap_tracking: ApTracking,
}

#[pymethods]
impl PyIds {
    pub fn __getattr__(&self, name: String, py: Python) -> PyResult<PyObject> {
        let hint_ref = self.references.get(&name);
        get_value_from_reference(&self.vm, hint_ref, &self.ap_tracking)? .to_object(py)
    }

    pub fn __setattr__(&self, name: String, val: PyMaybeRelocatable) -> PyResult<()> {
        let hint_ref = self.references.get(&name);
        let var_addr = compute_addr_from_reference(hint_ref, &self.vm, &self.ap_tracking);
        self.vm.memory.insert(&var_addr, &val)
    }
}
```
Python Hints FFI:
Interaction between hints through scopes

```rust
def get_scope_locals(
    exec_scopes: &ExecutionScopes,
    py: Python,
) -> PyDict {
    let locals = PyDict::new(py);
    for (name, elem) in exec_scopes.get_local_variables() {
        if let Some(pyobj) = elem.downcast_ref::<PyObject>() {
            locals.set_item(name, pyobj);
        }
    }
    locals
}

def update_scope_locals(
    exec_scopes: &mut ExecutionScopes,
    locals: &PyDict,
    py: Python,
) {
    for (name, elem) in locals {
        exec_scopes.assign_or_update_variable(&name, any_box!(elem.to_object(py)));
    }
}
```
```rust
pub(crate) fn execute_hint(
    &self,
    hint_data: &HintProcessorData,
    exec_scopes: &mut ExecutionScopes,
){
    Python::with_gil(|py| {
        let locals = get_scope_locals(exec_scopes, py)?;
        let globals = PyDict::new(py);

        globals.set_item("memory", PyMemory::new(&self));
        globals.set_item("segments", PySegmentManager::new(&self));
        globals.set_item("ap", PyRelocatable::from(self.vm.ap));
        globals.set_item("fp", PyRelocatable::from(self.vm.fp));
        globals.set_item("ids", PyIds::new(&self, &hint_data.ids_data, &hint_data.ap_tracking));

        py.run(&hint_data.code, Some(globals), Some(locals))
        update_scope_locals(exec_scopes, locals, py);
    });
}
```
cairo-rs-py
## Benchmarks

### Linear Search

<table>
<thead>
<tr>
<th>VM</th>
<th>Mean [s]</th>
<th>Min [s]</th>
<th>Max [s]</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cairo VM (CPython)</td>
<td>11.6 ± 0.2</td>
<td>11.1</td>
<td>11.9</td>
<td>105 ± 3</td>
</tr>
<tr>
<td>Cairo VM (PyPy)</td>
<td>3.51 ± 0.09</td>
<td>3.33</td>
<td>3.66</td>
<td>31.9 ± 1.1</td>
</tr>
<tr>
<td>Cairo-rs (Rust)</td>
<td>0.11 ± 0.01</td>
<td>0.11</td>
<td>0.12</td>
<td>1.0</td>
</tr>
</tbody>
</table>

### Common Lib Math Functions

<table>
<thead>
<tr>
<th>VM</th>
<th>Mean [s]</th>
<th>Min [s]</th>
<th>Max [s]</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cairo VM (CPython)</td>
<td>63.7 ± 1.0</td>
<td>61.3</td>
<td>65.8</td>
<td>130 ± 2</td>
</tr>
<tr>
<td>Cairo VM (PyPy)</td>
<td>12.1 ± 0.3</td>
<td>11.6</td>
<td>12.9</td>
<td>24.7 ± 0.7</td>
</tr>
<tr>
<td>Cairo-rs (Rust)</td>
<td>0.49 ± 0.01</td>
<td>0.49</td>
<td>0.50</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Thank you!

Federica Moletta, Herman Obst
federica.moletta@lambdaclass.com
herman.obst@lambdaclass.com
@classlambda
@herman_obst
lambdaclass