Rust for the Jungle

Seo Sanghyeon
Samsung Electronics

2013-07-25
Disclaimers

- Opinions here are my own and not that of my employer
- Not all code is running code
Speaker

- PyPy
- IronPython
- Clang
<table>
<thead>
<tr>
<th>Device</th>
<th>Release Date</th>
<th>Processor Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPad 2</td>
<td>2011-03-02</td>
<td>dual-core</td>
</tr>
<tr>
<td>iPhone 4S</td>
<td>2011-10-04</td>
<td>dual-core</td>
</tr>
<tr>
<td>Galaxy S III</td>
<td>2012-05-03</td>
<td>quad-core</td>
</tr>
<tr>
<td>Galaxy S4</td>
<td>2013-03-14</td>
<td>octa-core (big.LITTLE)</td>
</tr>
</tbody>
</table>
Intel MIC
Three layer cake

- Message passing
- Fork join
- SIMD
## Three layer cake

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message passing</td>
<td>Non-deterministic</td>
<td>Thread</td>
</tr>
<tr>
<td>Fork join</td>
<td>Deterministic</td>
<td>Thread</td>
</tr>
<tr>
<td>SIMD</td>
<td>Deterministic</td>
<td>Vector</td>
</tr>
</tbody>
</table>
Minimal Rust

```rust
default
let name: Type = value;
```
let name: Type = value;

let name = value;
let name;
name = value;
Minimal Rust

```rust
let name;
name = value;

let mut name = value;
name = another_value;
```
let i: int = 1;
let pi: &int = &i;
*pi;
let i: int = 1;
let pi: &int = &i;
*pi;

let mut i: int = 1;
let mpi: &mut int = &mut i;
*mpi = 2;
Rust

- Safe manual memory management
- Data race freedom
Safe manual memory management

If reference counts are always 0 or 1, you don’t need reference counts

- ref = allocate
- unref = free
Data race freedom

If writer counts are always 0 or 1, you don’t need mutual exclusion

- Purely functional: 0 writers
- Safe and mutable: 1 writer
- Data race: more than 2 writers
Aliasing control

- No alias analysis
- No false positives
- False negatives: how to make this usable
Nested aliasing

```rust
{
    let a: ~T;
    *a;
    {
        let b: &T = a; // borrow
        *a;
        *b;
    }
    *a;
}
```
Non-overlapping aliasing

```plaintext
{
    let a: ~T;
    *a;
    let b: ~T = a; // move
    *b;
}
```
Multiple readers

```rust
{
    let a: ~T;
    {
        let r1: &T = a;
        let r2: &T = a;
    }
}
```
Single writer

```rust
{
    let mut a: ~T;
    {
        let w1: &mut T = a;
        let w2: &mut T = a; // error
    }
}
```
Writing is exclusive

```rust
{
    let mut a: ~T;
    {
        let r: &T = a;
        let w: &mut T = a; // error
    }
}
```
## Compatibility table

<table>
<thead>
<tr>
<th></th>
<th>&amp;</th>
<th>&amp;mut</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>&amp;mut</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
References

- Welcome to the Jungle
  http://herbsutter.com/welcome-to-the-jungle/
- Three layer cake for shared memory programming
  http://dl.acm.org/citation.cfm?id=1953616
References

- On the connection between memory management and data race freedom
- Data parallelism in Rust
- http://smallcultfollowing.com/babysteps/