Is the Problem-Based Benchmark Suite Fearless with Rust?

Javad Abdi, Guowei Zhang, Mark C. Jeffrey

SPAA 2023
Rust is gaining popularity because of its safety guarantees
Rust is gaining popularity because of its safety guarantees.

Rust is on its seventh year as the most loved language...

Is the Problem-Based Benchmark Suite Fearless with Rust?
Rust is gaining popularity because of its safety guarantees.

Rust is gaining popularity because of its safety guarantees.

Rust is gaining popularity because of its safety guarantees.

Rust catches all type and memory safety errors.

Rust claims to provide “fearless concurrency”

**Fear**: Anticipation of concurrency errors that manifest at run time.

**Our RQ**: How does fearless concurrency translate to parallelism?

![Diagram](image)

Is the Problem-Based Benchmark Suite Fearless with Rust?

4
Rust claims to provide “fearless concurrency”

**Fear**: Anticipation of concurrency errors that manifest at run time.

**Our RQ**: How does fearless concurrency translate to parallelism?

Are all parallel patterns fearless in Rust?
Contribution: Interrogate fearless concurrency by expressing (ir)regular parallelism

**Rusty-PBBS:**
- A port of PBBS [Anderson et al., PPoPP’22] in Rust with both regular and irregular patterns.

**Our Case Study:**
- Classification of parallel expression patterns in Rusty-PBBS.
- Evaluating Rust support and fearlessness for each pattern.
Contribution: Interrogate fearless concurrency by expressing (ir)regular parallelism

**Rusty-PBBS:**
- A port of PBBS\cite{Anderson et al.,PPoPP'22} in Rust with both regular and irregular patterns.

**Our Case Study:**
- Classification of parallel expression patterns in Rusty-PBBS.
- Evaluating Rust support and fearlessness for each pattern.
Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
  - Known set of tasks
  - Known dependences
Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
Known set of tasks
Known dependences

```rust
fn par_increment(v: &mut [u32]) {
    v.par_iter_mut().for_each(|vi| *vi += 1);
}
```

Stride pattern on `v`
Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
- Known set of tasks
- Known dependences

```rust
fn par_increment(v: &mut [u32]) {
    v.par_iter_mut().for_each(|vi| *vi+=1);
}
```

Stride pattern on `v`

Task
Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
Known set of tasks
Known dependences

```rust
fn par_increment(v: &mut [u32]) {
    v.par_iter_mut().for_each(|vi| *vi+=1);
}
```

(task: Cannot access `v` ⇒ No data races)
Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
Known set of tasks
Known dependences

```rust
fn par_increment(v: &mut [u32]) {
    v.par_iter_mut().for_each(|vi| *vi+=1);
}
```

(task: Cannot access v) ➞ No data races
Is the Problem-Based Benchmark Suite Fearless with Rust?

Fearless regular parallelism with Rust(+Rayon)

Regular parallelism:
- Known set of tasks
- Known dependences

Rust statically rules out data races for regular parallelism

```rust
fn par_increment(v: &mut [u32]) {
    v.par_iter_mut().for_each(|vi| *vi+=1);
}
```

Stride pattern on `v`:
The task cannot access `v`:

⇒ No data races

(SndInd)

(RngInd)
Irregular parallelism remains scary

```rust
def indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter()
    .for_each(|i|
        v[offsets[i]] += 1
    );
}
```
Irregular parallelism remains scary

```rust
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter().for_each(|i|
        v[offsets[i]] += 1)
}
```

Dangerous
Irregular parallelism remains scary

```rust
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter().
        for_each(|i| v[offsets[i]] += 1);
}
```

Parallel loop

Dangerous

Compile error
Irregular parallelism remains scary

```rust
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter()
        .for_each(|i| v[offssets[i]] += 1);
}
```

Duplicates: Synchronization

Compile error

Dangerous

Offsets
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter().
    .for_each(|i|
        v[offsets[i]] += 1
    );
}
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter()
        .for_each(|i|
            v[offssets[i]] += 1
        );
}
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter()
        .for_each(|i|
            v[offsets[i]] += 1);
}

offsets

Duplicates: Synchronization 🤦‍♂️

Unique

Unsafe without checks 🤨

Unsafe with checks 🎉

Synchronization

Is the Problem-Based Benchmark Suite Fearless with Rust?
Irregular parallelism remains scary

```rust
fn indirect_increment(v: &mut [u32], offsets: &[usize]) {
    (0..v.len()).into_par_iter().for_each(
        | i |
        v[offsets[i]] += 1;
    );
}
```

Rust solutions for irregular parallelism are not fearless

- Duplicates: Synchronization
- Unique: Unsafe without checks, Unsafe with checks, Synchronization

Is the Problem-Based Benchmark Suite Fearless with Rust?
Does this matter?
Irregular parallelism is common in PBBS!

Regular parallelism ✔
Irregular parallelism ❌
Does this matter?
Irregular parallelism is common in PBBS!

Regular parallelism ✔
Irregular parallelism ✘

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Regular</th>
<th>Irregular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RO</td>
<td>Stride</td>
</tr>
<tr>
<td>bwid</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>dedup</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>dr</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>hist</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>isort</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>lrs</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>mis</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>mm</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>msf</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sa</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sf</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sort</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Irregular parallelism is common in PBBS!

Expressing PBBS in Rust is not fearless

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Regular</th>
<th>Irregular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RO</td>
<td>Stride</td>
</tr>
<tr>
<td>hist</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>isort</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>lrs</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>mis</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>mm</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>msf</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sa</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sf</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sort</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Does this matter?

Does this matter?

Irregular parallelism is common in PBBS!
Conclusions

Regular parallelism

Irregular parallelism

Easy parallelism is fearless!

Hard parallelism is still scary...

github.com/mcj-group/rusty-pbbs