神奈川大学理学部　情報科学科　馬谷研究室

Java の仮想スレッドを用いた Lox 言語への並行機能の追加

片山翔太　201902961

1　はじめに

本研究では Java 19 の新機能である仮想スレッドを使って Java 言語で開発した Lox 言語に並行機能を追加した。Lox 言語で扱えるコードは式と文に分かれている。式には四則演算や比較などがある。文には print 文、ループ、if 文、変数宣言などがすでに実装済みである。しかし、並行処理機能はまだ実装されていない。そこで、Java 19 で新しく追加された仮想スレッドを使って、Lox 言語に並行処理機能を追加することにした。

2　仮想スレッド

従来の Java のスレッドは、1 つのスレッドを生成するとネイティブ側でも同じように 1 対 1 で対応するスレッドが生成されていた。OS で管理するネイティブスレッドは、負荷が大きくかかる。これに比べて仮想スレッドは、Java のバーチャルマシンの中でスレッドを作成、管理できる。これにより、メモリの消費とスレッド生成のコストを軽減することができる。大量のスレッドを生成するような場面で、従来のスレッドと比べて大きな違いを生むことができる。

下記に Java 19 で仮想スレッドを作成し、スタートするコードの例を示す。

```
Runnable task = () -> {
  System.out.println("test");
};

var thread = Thread.ofVirtual().
                        start(task);
```

`Thread.ofVirtual()` で仮想スレッドを生成し、`start(task)` で `Runnable` 型の `task` を仮想スレッドでスタートしている。`Thread.ofPlatform` とすると、従来のスレッドが生成される。

3　設計

Lox 言語は、インタプリタとして実装されている。インタプリタには字句解析の Scanner クラス、構文解析器の Parser クラス、構文実行器の Interpreter クラスの三つの構成がある。新しく機能を追加する場合、それぞれに手を加える必要がある。Lox 言語の開発には Visitor パターンを採用しているため機能の追加が容易になっている。Visitor パターンは Visitor を引数として受け取る `accept()` メソッドを持ったクラスを複数、要素として持つ構造のことである。

Lox に追加した仮想スレッドの使用例を以下に示す

```
var VThread = fork { print 1; };
```

上記コードは、仮想スレッドで数字の 1 を表示するコードである。仮想スレッドは式として扱っているため、変数に格納できる。これにより複数のスレッドを区別することができる。まず `var` で仮想スレッドを変数に格納し、`fork{};` でブロックの中を仮想スレッドで処理している。このコードを実行できるように実装を行っていく。

4　実装

設計で示した三つの構成に従い順番に実装していく。まず Scanner クラスで字句解析をする。`Map` を使い String 型の`"fork"`と TokenType 型の `FORK` をペアにして格納する。これにより `fork` というキーワードをトークンとして認識できるようになる。

```
private static final Map<String,
            TokenType> keywords;
static {
    keywords = new HashMap<>();
    keywords.put("fork",  FORK);
}
```

次に Parser クラスで構文解析をする。まずトークン `FORK` にマッチするかを調べる。マッチした場合、空のリストを作成する。次に`"{"`とマッチするかを調べる。マッチした場合`"}"`とマッチするか最後の文字までの文をリストに追加する。`"}"`がなかった場合はエラーを報告する。最後に変数 `expr` を visitor の Expr クラスの Fork メソッドにインス

タンス化する。

```
if (match(FORK)) {
    List<Stmt> statements =
            new ArrayList<>();
    if (match(LEFT_BRACE)){
        while (!check(RIGHT_BRACE)
                    && !isAtEnd()) {
            statements.add
                    (declaration());
        }
    }

    consume(RIGHT_BRACE, "Expect '}'
                        after block.");
    Expr expr = new Expr.Fork
                        (statements);
    return expr;
}
```

最後に Interpreter クラスで、構文解析をする。Runnable 型の task を VirtualThread で起動する。task には構文解析したリストが入っている。最後にそのスレッドを返す。

```
public Thread visitForkExpr
            (Expr.Fork expr) {
    Runnable task = (() -> {
        executeFork(expr.statements,
            new Environment
                    (environment));
    });
    var vThread = Thread.ofVirtual().
                          start(task);
    return vThread;
}
```

Visitor クラスである Expr クラスは下記のようになっている。

```
static class Fork extends Expr {
    Fork(List<Stmt> statements)
        { this.statements = statements; }
    @Override
    <R> R accept(Visitor<R> visitor) {
        return visitor.
                    visitForkExpr(this);
    }
    final List<Stmt> statements;
}
```

5 性能比較

従来のスレッドと仮想スレッドの性能を比較する。そのために従来のスレッドを生成する `start` という文を Lox に追加した。実装方法は、Interpreter クラス以外は仮想スレッドと同様である。

```
public Thread visitStartExpr
            (Expr.Start expr){
    Runnable task = (() -> {
        executeFork(expr.statements,
        new Environment(environment));
    });
    var pThread = Thread.ofPlatform().
                            start(task);
    return pThread;
}
```

使用方法も同じである。二つの性能を比較するために 10 万個のスレッドをそれぞれ生成する。

下記は仮想スレッドを 10 万個生成するコードである。

```
for(var i=0; i<100000; i=i+1)
            { var vthread = fork{}; }
```

下記は従来のスレッドを 10 万個生成するコードである。

```
for(var i=0; i<100000; i=i+1)
            { var pthread = start{}; }
```

上記二つのコードをそれぞれ 10 回ずつ行い、処理にかかった平均時間を比較する。処理時間の計算には `System.currentTimeMillis()` を使用する。

結果は、仮想スレッドが平均 1885ms で従来のスレッドが平均 26200ms であった。仮想スレッドは従来のスレッドの 1/13 の速度で処理が終わった。

6 今後の課題

今後の課題は、`join()` メソッドを追加することである。`join()` メソッドとは、当該スレッドが終了するまで呼び出し元のスレッドを待機させるメソッドである。Lox 言語では、スレッドを変数に格納することでスレッドを区別できるようにしたので、`join()` メソッドがあれば特定のスレッドの終了を待機することができる。。これにより、仮想スレッドがより便利なものになると考える。

7 まとめ

本研究では、Lox 言語に仮想スレッドを使って並行機能を追加した。そして、従来のスレッドとの性能を比較することで仮想スレッドの有用性を証明することができた。

参考文献

[1] JEP 425: Virtual Threads (Preview)
`https://openjdk.org/jeps/425`
[2] Crafting Interpreters
`http://craftinginterpreters.com`