C++ 11 でマルチスレッド・コードを記述する

この記事は、インテル® デベロッパー・ゾーンに公開されている「Writing Multi-Threading Code in C++ 11」の日本語参考訳です。

長年にわたって、開発者は POSIX* スレッド (別名 Boost Threads) を使用してコードをスレッド化してきました。POSIX* は古くからありますが、問題の多いツールです。例えば、POSIX* にはスレッドのキャンセルがありますが、これは std::thread で問題を生じます。特に、RAII (Resource Acquisition Is Initialization: リソースの確保は初期化時に) を行う場合、オブジェクト生成中のリソースの取得とデストラクターによる割り当て解除で問題になります。しかし、代替手段があります。この記事では、POSIX* ライブラリーを使用して C++ 11 でコードをスレッド化する方法を示します。

ここでは、5 枚のカードから成るポーカーハンドを評価する単純なプログラムを使用します。各カードには、KC (クラブのキング) や 2H (ハートの 2) のように、ランク (番号) とスート (マーク) を示す 2 つの文字があります。これは良く定義された問題であり、非常に効率良く、高速なプログラムが出回っています。ここでは、シングルスレッド・バージョンから派生したマルチスレッド・バージョンが大幅にスピードアップするかどうか見ていきます。

上がり役を評価する
上がり役は、ロイヤルフラッシュが最も強く、ハイカードが最も弱く、その間に、ワンペア、ツーペア、スリー・オブ・ア・カインド、ストレート (5-6-7-8-9 のようにランクが連続している場合)、フラッシュ (すべてのカードが同じスートの場合)、フルハウス (同じランクが 3 枚とワンペアの場合)、フォー・オブ・ア・カインド (同じランクのカードが 4 枚ある場合)、ストレートフラッシュ (同じスートでランクが連続している場合) があります。

サンプルプログラムでは、ランダムに生成された 100 万件の上がり役を使用します。このデータは、テキストファイルに保存されており、各行に 1 つの役を示す RSRSRSRSRS 形式の 10 文字があります。"R" はランク、"S" はスートを表します。例えば、AS7D8S7HJD は、スペードのエース、ダイヤモンドの 7、スペードの 8、ハートの 7、ダイヤモンドのジャックを表し、7 が 2 つあるため、ワンペアと評価されます。

最初の 5 つの役は以下のとおりです。

```
7S2H7DTH5C -- 7 のワンペア
THQC2SAC3H -- ハイカード: クラブのエース
7SKC6H8H9S -- ハイカード: クラブのキング
9DKD4HQS8S -- ハイカード: ダイヤモンドのキング
3DAC3H2S5H -- 3 のワンペア
```

サンプルプログラムは、できるだけ速く 100 万行を読み取り、評価して、上記のような結果を出力する必要があります。評価が正しく行われることを確認するため、以下の役で再度実行してみたところ、次のような結果になりました。

```
6S2H7DTH5C -- ハイカード
THTC2SAC3H -- 10 のワンペア
7S9C6H6C9S -- ツーペア
9DKDKHKS8S -- スリー・オブ・ア・カインド
2D4C3H6S5H -- ストレート
4S3HAC2D5S -- ストレート (5-ハイ)
KCQHJDAHTS -- ストレート (A-ハイ)
2D9D3D6D5D -- フラッシュ
```

```
7S9H9S7C9C -- フルハウス
9DKDKHKSKC -- フォー・オブ・ア・カインド
2D4D3D6D5D -- ストレートフラッシュ
JCQCKCACTC -- ロイヤルフラッシュ
```

次に、すべてのハンドを昇順に並べ替えた結果を示します。ストレートの場合は、コードで特別な処理が必要になります。エースから順にキング方向へチェックしていきます。

```
2H5C6S7DTH -- ハイカード
2S3HTHTCAC -- ワンペア
6H6C7S9C9S -- ツーペア
8S9DKDKHKS -- スリー・オブ・ア・カインド
2D3H4C5H6S -- ストレート
AC2D3H4S5S -- ストレート
TSJDQHKCAH -- ストレート
2D3D5D6D9D -- フラッシュ
7S7C9H9S9C -- フルハウス
9DKDKHKSKC -- フォー・オブ・ア・カインド
2D3D4D5D6D -- ストレートフラッシュ
TCJCQCKCAC -- ロイヤルフラッシュ
```

C++ 11
サンプルコードでは、いくつかの C++ 11 機能を使用しています。例えば、配列の代わりに、新しいコレクション・クラスに std::array を使用したり、列挙型に新しい enum 型を使用しています。新しい enum 型では、static_casts により a size_t (整数型) から列挙型への変換が必要です。また、コードを単純化するため、"for (auto variable : collection)" ループも使用しています。オブジェクトを変更する場合、"auto& " を使用して、役の各カードの選択済みフラグを false に設定します。

```
for (auto& card : Ph.hand) {
    card.selected = false;
}
```

評価の実装
カードを格納する PokerCard クラスを作成し、次に std::array<PokerCard,5> を持ち、文字列からカードを 1 枚ずつ処理してハンドを生成する PokerHand オブジェクトを作成します。そして、評価を行うため、PokerHand クラスにフレンドメソッドを追加します。さらに、役を読み取った後、ランク順 (A、2...K) にカードを並べ替えるカスタムソートを使用しています。これにより、役の評価を容易に行うことができます。

ツーペアとフルハウス (同じランクが 3 枚とワンペア) をチェックするため、最初にワンペア、スリー・オブ・ア・カインド、フォー・オブ・ア・カインドを探し、各カードのフラグを true に設定します。ワンペアまたはスリー・オブ・ア・カインドが見つかった場合、フラグが true に設定されていないカードの中からワンペアまたはスリー・オブ・ア・カインドがないか探します。前のカードへのポインターを使用することで、ワンペアが見つかった場合、両方のカードの選択済みフラグを true に設定することができます。

シングルスレッド・コードのリリースバージョンは、100 万件のハンドの処理に 3.35 秒 (あるいは、1 役の評価に 3.35 マイクロ秒) かかりました。12MB 近いファイルを 1 行ずつ読み取っていることを考えれば、悪くない結果と言えます。

これを上回ることはできるでしょうか? もちろんできます。以下は、main() のコードです。このコードを変更する必要があります。

```
  while (std::getline(filein, str))
      {
          PokerHand pokerhand(str);
          auto result=EvaluateHand(pokerhand);
          pokerhand.WriteResult(fileout,result);
      }
```

マルチスレッド化する
完全に独立したフレンド関数 EvaluateHand() を作成します。この関数はモノリシックで、160 行ほどあり、プログラムの約半分を占めます。この方法により、簡単にマルチスレッド化できます。必要なのは PokerHand のインスタンスの参照だけです。

最も簡単な方法 (そして、成功する確率の高い方法) は、std::async を使用して、一度に 11 個 (例えば、各スレッドで 1 個) を評価することです。async では、プログラマーは使用するスレッドを選べません。しかし、内部でスレッドプールを使用しているようです。以下が変更後のコードです。シンボル Multi は、コンパイラーの #define で定義しています。#define 行をコメントアウトしてファイルを保存すると、シングルスレッド・バージョンになります。マルチスレッド・バージョンではコメントのままにします。

評価結果は、futures に格納されます。future は、非同期実行されるコードの完了を待機して結果を格納するオブジェクトです。非同期なので、評価の完了を待機するため、各 future に対して get() を呼び出す必要があります。そのため、理論上は、一度に 12 個を評価します。これは、futures[count++] = の右辺のコードで行われます。ラムダ式 (匿名関数) として定義されており、[str] で文字列 str を渡します。MaxThreads は定数で 12 に定義されているため、11 個のスレッド (1 つはメインスレッドに必要なため) を使用します。

```
std::array<std::future,MaxThreads> futures;
auto count = 0;
while (count < MaxThreads-1) {
    if (filein.eof()) break;
    std::getline(filein, str);

    futures[count++] = std::async([str]{
        PokerHand pokerhand(str);
        auto result = EvaluateHand(pokerhand);
        return pokerhand.GetResult(result);
    });
    if (count == MaxThreads-1) {
        for (auto & e : futures) {
            fileout << e.get() << std::endl;
        }
        count = 0;
    }
}
```

MaxThreads の値は、以下を呼び出すことで取得できます。

```
auto NumThreads = std::thread::hardware_concurrency();
```

スピードアップできたのでしょうか?
これだけの労力をかけたにもかかわらず、実行に 4.33 秒かかりました。つまり、シングルスレッド・バージョンよりも 33% も実行時間が長くなりました。評価の実行時間が短すぎて、タスクを非同期に呼び出すオーバーヘッドが原因であることが考えられます。

この仮説を検証するため、EvaluateHand() 関数に 1 ミリ秒の遅延を生じさせる Sleep(1) を追加してみました。また、評価する役の数を 2 万件に減らしました。そして、両方のバージョンを実行したところ、マルチスレッド・バージョンの結果は 3.25 秒、シングルスレッド・バージョンの結果は 31 秒になりました。

つまり、スピードアップの理論は正しく、評価関数が速すぎたのです。

VC++ 15 プロジェクト、ソースファイル、および 100 万件のハンドを含む、ここで使用したテストファイルは GitHub* (英語) からダウンロードできます。