情報科学プロジェクト実験レポート課題

S142063 佐藤涼亮

平成 29 年 1 月 25 日

スレッドの同期

1　課題の内容

スレッドの同期による素数のカウント

1.1　要点

1. マルチスレッドプログラミングによる並列処理を行う
2. スレッド数は CPU 数に応じて設定する
3. 複数のスレッドがメモリ等の計算資源を操作しているときに競合が発生しないようにする

2　プログラムの説明

逐次処理で計算していたプログラムを、マルチスレッドプログラムに変更し並列処理を行い、実行時間の短縮を図る。今回、変更するプログラムは指定した数以下の自然数にいくつ素数があるかを数えるプログラム。

並列処理には、std::thread を使い、自然数を分割し素数判定を行う。スレッドの数は CPU の数に合わせる。

2.1　目的

マルチスレッドプログラムのスレッドの同期による、プログラムの整合性を保つ

2.2　方法

シングルスレッドプログラムをマルチスレッドプログラムに変更し、スレッドの同期を用いた素数のカウント

2.3 結果

```
ryousuke@HP-Spectre:~/project/15sync$ time ./prime 3333333
# of prime (<=3333333) is 239119

real  0m0.531s
user  0m0.528s
sys 0m0.000s
ryousuke@HP-Spectre:~/project/15sync$ time ./prime_thread 3333333
# of prime (<=3333333) is 239119
It took 0.289744 seconds

real  0m0.293s
user  0m1.132s
sys 0m0.000s
ryousuke@HP-Spectre:~/project/15sync$ time ./prime 33333333
# of prime (<=33333333) is 2050943

real  0m13.164s
user  0m13.164s
sys 0m0.000s
ryousuke@HP-Spectre:~/project/15sync$ time ./prime_thread 33333333
# of prime (<=33333333) is 2050943
It took 7.28054 seconds

real  0m7.284s
user  0m28.984s
sys 0m0.000s
ryousuke@HP-Spectre:~/project/15sync$ time ./prime 333333333
# of prime (<=333333333) is 17955279

real  5m48.356s
user  5m48.368s
sys 0m0.008s
ryousuke@HP-Spectre:~/project/15sync$ time ./prime_thread 333333333
# of prime (<=333333333) is 17955279
It took 198.546 seconds

real  3m18.550s
user  13m13.264s
sys 0m0.000s
ryousuke@HP-Spectre:~/project/15sync$
```

指定した数	逐次処理	並列処理	結果
3333333	0m0.531s	0m0.293s	239119
33333333	0m13.164s	0m7.284s	2050943
333333333	5m48.356s	3m18.550s	17955279

CPU コア 4 個の環境で実行し、時間測定には time コマンドを用いた。

2.4 考察

結果を見ると、シングルスレッドのプログラムに対して、std::thread の場合は約 1.8 倍、処理効率が向上したと言える。また、結果も一致していることからプログラムの整合性も保たれたと言える。

3 感想

今回の実験においても、マルチスレッド化することでプログラムの整合性を保ちつつ効率は上げることができた。今回も real 時間と user 時間を見比べてみると、CPU の数だけしっかり活用できている。プログラムを並列化するとき、どの部分をどのように並列化するかが難しいと感じた。今回の課題の場合、100 を 1 25,26 50...,76 100 のように単純に分けてしまうと、数が大きくなるに連れて調べる回数が増えるため、最初に割り当てられたスレッドは短く、最後に割り当てられたスレッドは長くなってしまう。指定された数が大きいほどその差は大きくなってしまう。差が大きくなると、早く終わってしまったスレッドはその分無駄になってしまうので、平均的に割り当てられるようにしたい。そこで今回は、`[1,9,...],[3,11,...],[5,13,...],[7,15,...]` のように、判定を奇数のみに絞り、数値の大小が偏らないように割り当てることで対応した。このような対応力を経験を積むことで鍛えていきたいと思う。

4 プログラム

ソースコード 1: prime_number.cpp

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;
bool is_prime(int x) {
        if (x == 2) return true;
        if (x % 2 == 0) return false;
        for (int i = 3; i*i <= x; i+=2 )
                if (x % i == 0) return false;
        return true;
}
int main(int argc, char *argv[]) {
        if (argc < 2) {
                cerr << "usage: " <<argv[0]<<" max\n";
                return 1;
        }
        int max = atoi(argv[1]);
        if (max <= 0) return 1;
        int count = 0;
```

```
19          for (int i = 2; i <=max; i++) // prime number is greater than 1
20                  if (is_prime(i)) ++ count;
21          cout << "# of prime (<="<<max<<") is "<<count<<"\n";
22          return 0;
23  }
```

ソースコード 2: prime_number_thread.cpp

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <vector>
4   #include <thread>
5   #include <chrono>
6   #include <mutex>
7   #include <condition_variable>
8   using namespace std;
9
10  int worker = 0;
11  int count = 0;
12  mutex f;
13  bool is_prime(int x) {
14    for (int i = 3; i*i <= x; i+=2 )
15      if (x % i == 0) return false;
16    return true;
17  }
18
19
20  void prime_count(int n,const int p,int max,condition_variable &cv){
21          int fcount = 0;
22    if(n==1){ // 1を省き 2の分加算
23      n+=p;
24      if(max>=2) fcount++;
25    }
26          for (int i = n; i <= max; i+=p) // 素数は奇数のみ判定
27      if (is_prime(i)) fcount++;
28          unique_lock<mutex> l(f);
29          count += fcount;
30          if(--worker == 0) cv.notify_one();
31  }
32
33  int main(int argc, char *argv[]) {
34          if (argc < 2) {
35                  cerr << "usage: "<<argv[0]<<" max\n";
36                  return 1;
37          }
38          int max = atoi(argv[1]);
39          if (max <= 0) return 1;
40          const int num = thread::hardware_concurrency();
41          worker = num;
42          mutex mx;
43          condition_variable cv;
44          auto start = std::chrono::high_resolution_clock::now();
45          for (int i=0; i<num; i++){
46                  thread(prime_count,i*2+1,num*2,max,ref(cv)).detach();
47          }
48          unique_lock<mutex> lock(mx);
49          while(worker > 0) cv.wait(lock);
50          auto stop = std::chrono::high_resolution_clock::now();
51          cout << "# of prime (<="<<max<<") is "<<count<<"\n";
52          std::cout << "It took "
53                  << std::chrono::duration<double>(stop-start).count()
54                  << " seconds\n";
55          return 0;
56  }
```