数論的関数における数値計算の高速化

あざみ野中学校3年　梶田光

概要

数論の研究において、数値計算は重要であるが、計算に数日かかるため検証できるアイデアの数に制限があった。
そこで計算を高速にするライブラリを開発した。またその過程で、素因数分解を効率的に記録できるアルゴリズムを発見した。
これにより、これまで使われてきたSympyの40,000倍の速度で計算が可能になった。

背景

初等整数論の研究では、数論的関数を含む方程式の解を研究する。
ここで重要となる例外解は、しばしば巨大で発見しづらいことが多い。
さらにSympy、Primes.jl等のライブラリを利用しても時間がかかるため、パソコンの性能が制約となる。
そこで予めルックアップテーブルを構築し、家庭用PCでも研究しやすくなるようなライブラリの実装を開始した。

実装

クロスプラットフォームで高速なRustで実装した。
またディスクに構築する際、高速なランダムアクセスが重要なため、メモリマップドファイルを利用した。
さらに、アクセス時にメモリは消費されるため、アルゴリズムは区間篩を選択した。
プログラムの構成は、

1. √Nまでの小さい素数を構成
   (これは大きくないためメモリ上にVec<u64>で保持)
2. RAMに収まる区間(chunk sizeと呼ぶ)でメモリマップドオブジェクトを作成し、その区間内で区間篩を実行
   (ここでRayonによる並列処理を行っている)

となっている。
なお、Rustが提供するMutex、RefCellなどの安全性は今回のパフォーマンス重視の目的と合致しなかったため、内部ではUnsafeCellを利用したり生ポインタをマルチスレッドに共有することで高速化を図っている。

新しいアルゴリズムの発見

今回問題になったのは素因数分解のテーブルについてである。
まずHashMapを用いた素因数と指数のペアの記録はメモリ量が大きくなるため避けたい。
しかし最小素因数だけを記録する方法では、素数で割っていく度にアクセスが増え、またそのアクセス箇所も不規則なため、指定の区間のメモリマップドオブジェクトとは相性が悪い。
そこで64ビットのnについて、√Nまでのnの最小素因数とそれぞれのnについて8バイトの情報を記録するだけでnの素因数分解がO(log n)で求められる(おそらく未発見の)**アルゴリズムの存在を数学的に証明し、実装した**。
これによってランダムアクセスを必要とする区間の長さを小さく保ちながら一つのnに対して8バイトのアクセスのみで素因数分解が高速にできるようになった。

(証明に関しては、学習院大学名誉教授/飯高茂先生のゼミにおいて発表した。論文は投稿準備中。)

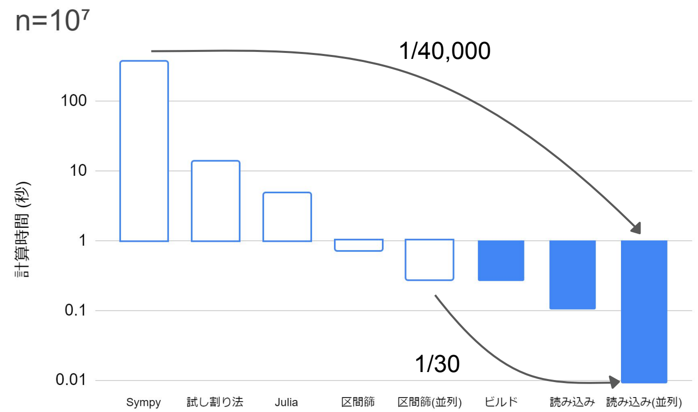
実際の流れ

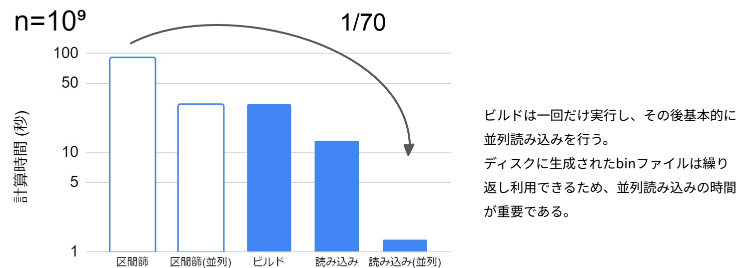
1. ルックアップテーブルをbuildメソッドで構築
2. loadメソッドで読み込み
3. 返されるSieveオブジェクトからfactorize、totientなどのメソッドを利用

ここでSieveオブジェクトからget_segmentでchunk sizeだけ読み込むことができる。
返されるPartialSieveはSend + SyncであるからRayonを利用して読み込みを並列化することも可能になる。

計測結果

n-2φ(n)=-1の解の数値計算にかかった時間を計測した。
(Sympy、Julia以外はすべてRustで実装、区間の長さは5.0*10⁸、**対数グラフ**であることに注意)





利点

- 並列読み込みを利用すれば10億までの走査が1秒強で終わる。この速度は方程式の例外解を見つけるにあたって有用である。
- 様々な最適化によってビルドに必要な時間も通常の並列化された区間篩と同程度まで抑えられた。
- 並列化専用のメソッドを用意しており、簡単に利用できる。
- このライブラリはオイラー関数を想定して作ったが、素因数分解が求められることから他の数論的関数も自然に計算が可能。

```
for i: usize in 0..sieve.segment_count() {
    sieve.load_segment_nth(segment_index: i)?;
    let view: PartialSieve<'_> = sieve.get_segment();
    sieve.nth_segment_range(segment_index: i).into_par_iter().for_each(op: |j: u64| {
        if j + 1 == 2 * view.totient(j).unwrap() {
            println!("{}", j);
        }
    });
}
```

―並列化を利用したコード―

制約と課題

HDDでは特にランダムアクセス時の読み書きが遅いため、SSDの使用が推奨される。

参考文献

https://doc.rust-lang.org/stable/std/cell/struct.UnsafeCell.html
https://docs.rs/rayon/latest/rayon/