Rust による円周率計算プログラム

著者：関 勝寿　公開日：2024 年 3 月 17 日　ソース：https://sekika.github.io/2024/03/17/Rust-pi/

円周率πを任意桁数計算するプログラム compute-pi を Rust で作成した。16GB のメモリを持つ PC では、円周率を 3 億 2000 万桁まで計算できる。

1. 計算方法

ガウス・ルジャンドルのアルゴリズム、すなわち初期値を
$a_0 = 1, b_0 = \frac{1}{\sqrt{2}}, t_0 = \frac{1}{4}, p_0 = 1$
反復式を

$$
\begin{aligned}
a_{n+1} &= \frac{a_n + b_n}{2} \\
b_{n+1} &= \sqrt{a_n b_n} \\
t_{n+1} &= t_n - p_n(a_n - a_{n+1})^2 \\
p_{n+1} &= 2p_n
\end{aligned}
$$

とすると、円周率πは
$\pi \approx \frac{(a+b)^2}{4t}$
と近似される、というアルゴリズムにより円周率を任意桁数計算する。多倍長計算の rug crate を使った。

MacBook (Apple M1, 16 GB) では、100 万桁を 1.5 秒で、3 億 2000 万桁を 24 分で計算できて、これがこのプログラムとマシン能力の限界桁数となる。このプログラムによる円周率 3 億 2000 万桁の計算結果は、y-cruncher と Chudnovsky アルゴリズムによる計算結果と完全に一致することが確認された。

高橋大介と金田康正は、1998 年にガウス・ルジャンドル法によって分散メモリ型並列計算機による円周率の 515 億桁計算をした。さらに高橋は 2009 年にガウス・ルジャンドル法で 2 兆 5769 億 8037 万桁を計算した。以下に、いくつかの参考サイトを示す。

- 世界記録は 31 兆桁！ 日本人も活躍する円周率「π」計算の最先端 - 柳谷晃，2021/7/17
- 円周率を求める公式・アルゴリズム - 円周率 .jp
- Chudnovsky の公式を用いた円周率の計算用メモ - Peria Peria, 2015/12/9
- パソコンによる円周率小数点以下５兆桁の計算 - 近藤茂，2011/6/30
- Even more pi in the sky: Calculating 100 trillion digits of pi on Google Cloud - Emma Haruka Iwao, 2022/6/9
- キオクシアと Linus Media Group が「最も正確な円周率の値」のギネス世界記録™を樹立 (300 兆桁) - キオクシア株式会社，2025/5/19
- 円周率の歴史 - Wikipedia

2. 使い方

Rust をインストールして cargo が使えるようになったら、次のように compute-pi crate をインストールする。

```
cargo install compute-pi
```

すると

```
compute-pi < 桁数 >
```

で、その桁数の円周率が表示される。たとえば、

```
compute-pi 10
```

とすると

```
Pi to 10 decimal places: 3.1415926535
```

と表示され、

```
compute-pi 1000
```

とすると

```
Pi to 1000 decimal places:
3.14159265358979323846264338327950288419716939937510582097494
45923078164062862089986280348253421170679821480865132823066647
09384460955058223172535940812848111745028410270193852110555596
44622948954930381964428810975665933446128475648233786783165271
20190914564856692346034861045432664821339360726024914127372458
70066063155881748815209209628292540917153643678925903600113
30530548820466521384146951941511609433057270365759591953092186
11738193261179310511854807446237996274956735188575272489122793
81830119491298336733624406566430860213949463952247371907021798
60943702770539217176293176752384674818467669405132000568127145
26356082778577134275778960917363717872146844090122495343014654
95853710507922796892589235420199561121290219608640344181598136
29774771309960518707211349999998372978049951059737328160963185
95024459455346908302642522308253344685035261931188171010003137
83875288658753320838142061717766914730359825349042875546873115
95628638823537875937519577818577805321712268066130019278766111
959092164201989
```

と表示される。クレートを読み込んで Rust プログラムの中から関数を使うこともできる。ドキュメント参照。

3. 計算時間

MacBook Air (Apple M1, 16 GB) での計算時間を time コマンドで計測した。

- 1 万桁：15.85 ミリ秒
- 10 万桁：0.12 秒
- 100 万桁：1.51 秒
- 1000 万桁：25.30 秒
- 1 億桁：403.05 秒 (6.7 分)
- 2 億桁：14.67 分
- 3 億桁：22.65 分
- 3 億 2000 万桁：23.52 分
- 3 億 3000 万桁：10 時間で計算停止せず

Mac mini (Apple M1, 16 GB) ではこのようになった。

- 100 万桁：1.76 秒
- 1000 万桁：34.2 秒
- 1 億桁：498.80 秒 (8.3 分)
- 3 億 2000 万桁：28.48 分
- 3 億 3000 万桁：10 時間で計算停止せず

このように、いずれの場合でも 3 億 2000 万桁までは 30 分以内に計算ができるが、3 億 3000 万桁の計算はできなかった。メモリ割り当てに失敗してパニックで終了するのではなく、いつまでも計算が終了しないことから、メモリのスワッピングに時間がかかっているものと推察される。実際に、vm_stat で確認すると、Pageins と Pageouts が計算中に増えていく。16GB のメモリのマシン 2 台で同じ結果となったことから、16GB のメモリの場合はこれが限界の桁数であると考えられる。

なお、rug::float の精度 (prec) が u32 で定義されていることから、メモリとは無関係に 1,292,913,983 桁が上限となっている。

4. 他のプログラムとの比較

円周率計算の世界記録更新によく使われている y-cruncher で、同様に 3 億 2000 万桁を計算した。Mac mini で Docker の Debian コンテナから y-cruncher を起動して計算したところ、87 秒で計算できた。MacBook では、なぜか 2.2 時間もかかった。このプログラムと y-cruncher の計算結果を比較したところ、3 億 2000 万桁すべての数字が一致することが確認された。

Wikipedia のガウス・ルジャンドルのアルゴリズムの記事に掲載されている Python のプログラムをもとに、Paiza でブラウザ上で円周率の指定桁数を計算するプログラムを作成した。「入力」に桁数を入れて実行すると、指定桁数まで計算する（最終桁は丸める）。このプログラムで 1 万桁の計算はできたが、2 万桁の計算は一定時間内に計算が終わらないため Timeout となった。Mac mini でこのプログラムを実行したところ、10 万桁では 11 秒、100 万桁では 3 分かかった。すなわち、Rust の compute-pi では、ガウス・ルジャンドルのアルゴリズムによる円周率 100 万桁の計算を Python プログラムの 100 倍の速度で実行できたということになる。