社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－108 (1)
2006／10／5

$\pi(x)$ の計算におけるパラメータの選択に関する考察

吉田 仁[†] 黒田久泰[††] 金田康正[††]

与えられた x 以下の素数を数える関数 $\pi(x)$ は，関数自身の定義や近似式を求めることが非常に単純であるのに対し、正確な値を求めることは非常に困難なことである。本研究では、$\pi(x)$ を求める一つの手法において、計算工程にかかる時間や必要メモリ量を変動させるパラメータを様々に変化させ、理論的に導かれる最適化パラメータとの違いを見る。最終的には $\pi(10^{11}), \pi(10^{12}), \pi(10^{13})$ の計算で得られたパラメータを用い $\pi(10^{14}), \pi(10^{15}), \pi(10^{16})$ を求めると、理論計算量より悪い計算時間オーダとなったが、$x \leq 10^{13}$ での計算時間オーダとはほぼ等しいという結果になった。

Discussion about Selecting Parameters in Computing $\pi(x)$

HITOSHI YOSHIDA,[†] HISAYASU KURODA[††] and YASUMASA KANADA[††]

$\pi(x)$ is "prime counting function", that outputs the number of prime numbers less than or equal to x. The definition of $\pi(x)$ is very simple, but the calculation of it is very difficult. In this paper, we measure computation time on some parameters and find optimal parameters. Using the parameters that are selected in computation of $\pi(10^{11})$, $\pi(10^{12})$, and $\pi(10^{13})$, we calculate $\pi(10^{14})$, $\pi(10^{15})$, and $\pi(10^{16})$. The order of computation time of $x \geq 10^{14}$ is similarly equal to that of $x \leq 10^{13}$

1. はじめに

$\pi(x)$ とは実数 x に対し x 以下の素数の個数を与える関数である。具体的な値として $\pi(10)$ を考えると、10 以下の素数は 2,3,5,7 の 4 つなので、$\pi(10) = 4$ ということになる。

そもそも、素数の定義は「1 とその数自身以外に約数を持たない自然数」となっているので 1 が素数に含まれるかどうか議論になることもあるが、今回の $\pi(x)$ の計算では 1 は素数に含まれないものとしている。

$\pi(x)$ の定義それ自体はとても簡単だが、大きな x に対して具体的な値を求めることは大変困難な問題となる。このため、具体的な値を出すための様々な計算手法が発明されている。

$\pi(x)$ の値を求めることは昔から行われていたが、古代ギリシアの時代から 18 世紀までは Eratosthenes の篩を用いることにより x 以下の素数を列挙し、全て数え上げる手法が取られていたが、この方法は人間の手によって行われていたこともあり、x が大きくなると信用の置ける値とは限らなかった。

その後、Legendre(1752~1833) によって $\pi(x)$ を求める関数が提案された[1] が、大きな x について計算しようとなると計算の統制が取りにくくなるため計算を誤りやすかった。実際、Legendre 本人による計算でも

$$\pi(10^6) = 78,526$$

と出されているが、正解は 78,498 である。

Legendre から後も Meissel らによって新しい $\pi(x)$ の計算式が編み出され、次々と大きな x についての $\pi(x)$ が求められていき、2006 年現在では 2001 年に求められた $\pi(4 \cdot 10^{22})$ までの値が知られている。[2]

そこで本研究では、それらのうち最も新しい手法の 1 つに基づいて、東京大学情報基盤センターが所有するスーパーコンピュータ SR11000/J1 の環境を想定したプログラムを作成し、その計算におけるパラメータについての考察を行う。

2. 背景

2.1 $\pi(x)$ の求め方

本研究で用いる計算アルゴリズムを説明する前に、その中で使われるいくつかの関数についての説明を先に行う。

† 東京大学大学院新領域創成科学研究科
Graduate School of Frontier Sciences, The University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

まず、素数列を

$$\{p_i\} = \{2, 3, 5, 7, 11, 13, \cdots\}$$

と定義する。これを用いれば、ある自然数 n についての素因数分解を

$$n = \prod_i p_i^{e_i}$$

の形で表すことができる。

次に、x 以下でかつ p_a より大きな素因数しか持たない数を数える関数 $\phi(x, a)$ を考える。

$$\phi(x, a) = \#\left\{ n \le x;\ \sum_{i=0}^{a} e_i = 0 \right\}$$

そして同時に、x 以下でかつ p_a より大きな素数 k 個の積で表される数を数える関数 $\phi_k(x, a)$ も考えると

$$\phi_k(x, a) = \#\left\{ n \le x;\ \sum_{i=0}^{a} e_i = 0,\ \sum_{i=a+1}^{\infty} e_i = k \right\}$$

となる。特に、$k = 0, 1$ について言えば

$$\phi_0(x, a) = 1$$

$$\phi_1(x, a) = \pi(x) - a$$

であることがわかる。

$\phi(x, a)$ の計算はまた、任意の自然数 k において $P_k = \prod_{i=1}^{k} p_i$ とするとき

$$\phi(x, k) = \left\lfloor \frac{x}{P_k} \right\rfloor \phi(P_k) + \phi(x \bmod P_k, k) \qquad (1)$$

が成り立つことが知られており、前準備として適当な k における $\phi(x, k)$ を $x < P_k$ の範囲で求めておけば計算効率を上げることができる。[3)]

これらの式は以下のような関係になっていることが分かる。

$$\phi(x, a) = \sum_{k=0}^{\infty} \phi_k(x, a)$$

ここで、$x^{1/3} \le y \le x^{1/2}$ となる y を取り、$a = \pi(y)$ とすれば

$$\phi_k(x, a) = 0 \quad (\text{for } k \ge 3)$$

となるので、

$$\phi(x, a) = 1 + \pi(x) - a + \phi_2(x, a)$$

$$\therefore \pi(x) = \phi(x, a) - \phi_2(x, a) + a - 1 \qquad (2)$$

が導かれる。なお、本研究ではアルゴリズムとしてこの数式をさらに展開したものを用いるが、その具体的な数式は 3 節に記述する。

2.2 既知の値

現在求められている $\pi(x)$ のうち、最大のものは pi(x) project[2)] により

$$\pi(4 \cdot 10^{22}) = 783{,}964{,}159{,}847{,}056{,}303{,}858$$

と求められている。

参考までに、これまでの記録のうち x が 10^n の形をしているものを**表 1** に示した[4)]。

表 1 大きな n についての $\pi(10^n)$
Table 1 Known values of $\pi(10^n)$ for large n

n	$\pi(10^n)$
14	3,204,941,750,802
15	29,844,570,422,669
16	279,238,341,033,925
17	2,623,557,157,654,233
18	24,739,954,287,740,860
19	234,057,667,276,344,607
20	2,220,819,602,560,918,840
21	21,127,269,486,018,731,928
22	201,467,286,689,315,906,290

2.2.1 pi(x) project

pi(x) project[2)] は 3 節に示すアルゴリズムを開発した Gourdon が自身のウェブサイトで行っている分散コンピューティングプロジェクトである。これまでに $\pi(2 \cdot 10^{21}), \pi(4 \cdot 10^{21}), \pi(10^{22}), \pi(2 \cdot 10^{22}), \pi(4 \cdot 10^{22})$ を求めている。

2001 年には $\pi(10^{23})$ を求めるプロジェクトも行われ、一旦は $\pi(10^{23}) = 1{,}925{,}320{,}391{,}606{,}818{,}006{,}727$ という結果を得たが、結果確認の際に誤りがあると判断され、それ以降では $\pi(10^{23})$ の計算は行われていない。

2.3 素数定理

$\pi(x)$ の近似値を求める方法として、素数定理[1)] が挙げられる。この定理は $\pi(x)$ が以下の式で近似されるというものである。

$$\pi(x) \sim \mathrm{Li}(x) = \int_2^x \frac{dt}{\log t}$$

この式 (対数積分) より粗い近似でも構わない場合には、もっと簡単な形

$$\pi(x) \sim \frac{x}{\log x}$$

で計算することができるということが知られている。

また、対数積分よりも良い近似式としてリーマン関数が知られている。なお、この式中に現れる関数 $\mu(m)$ については後で説明する。

$$R(x) = \sum_{m=1}^{\infty} \frac{\mu(m)}{m} \mathrm{Li}(x^{1/m})$$

これらの近似式による計算結果をまとめたグラフが**図 1** である。$\pi(x)$ と $\mathrm{Li}(x), R(x)$ をプロットすると違いが見えないので、その差をプロットした。

2.4 値の確認

$\pi(x)$ のように未知の値を計算する場合には

- アルゴリズム上の間違い
- プログラムコーディング上の間違い
- コンピュータの内部エラー

などが原因となって誤った解答を出力する可能性を考

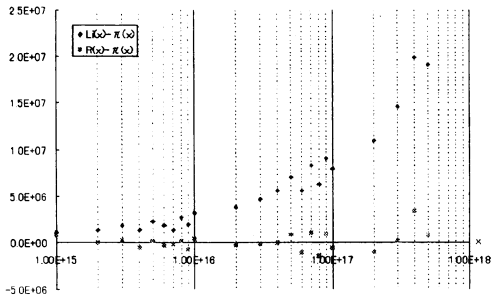

図 1 $\mathrm{Li}(x) - \pi(x)$ と $R(x) - \pi(x)$
Fig. 1 $\mathrm{Li}(x) - \pi(x)$ and $R(x) - \pi(x)$

え、何らかの方法を用いて確認を行わなければならない。実際、5) ではそれ以前に J. Bohman により計算されていた $\pi(10^{13})$ が誤っていたことを述べている。

ここでは、その手法として過去の研究[3),5),6)] で用いられてきた、もしくは提案されてきた値の確認方法を紹介する。

2.4.1 小さな値で確認

入力値となる x に例えば 10^n を代入するなどして、既に算出されている結果と比較することで確認を取る。1 つだけでなく、幅広くできる限り多くの数を試すことでプログラムやアルゴリズムに誤りがあるという確率を小さく見積もることができる。

2.4.2 パラメータの変化

先に述べたように、3) のアルゴリズムでは y と z という 2 つのパラメータ変数がある。これらを変動させることで実際の動作と各項の値が変化する。3) 以外のアルゴリズムに関してもパラメータ変数 y はあるので、それを変動させて確認を取ることができる。

2.4.3 ある区間の素数を数える

例えば $\pi(x)$ と $\pi(x+\Delta x)$ を求め、その差を取れば区間 $[x, x+\Delta x]$ に含まれる素数の数が分かる。そこで、実際に区間 $[x, x+\Delta x]$ に含まれる素数を求めてカウントしたものと比較することで整合性を示す。pi(x) project[2)] では $\Delta x = 10^8$ として確認を取っていた。

2.5 別のアルゴリズム

これまでの研究では使われていない確認方法としては、全く別のアルゴリズムに基づいた複数のプログラムを用いて値が一致することを確認する方法がある。

例えば $\pi(x)$ を求めるアルゴリズムの中では、3), 5), 6) はいずれも式 (2) を元に計算項数を減らして作られたアルゴリズムなので本質的には別の方法と言えないが、7) は ζ 関数の解析に基づいて作られた解析的な手法であり、上記のアルゴリズム類とは別系統として扱うことができる。つまりこちらのアルゴリズムに基づいたプログラムを作成し、同じ値が導かれれば正解であると判断できる。

また、計算能力が十分にあるようならば具体的に x 以下の素数を全て挙げ、数え上げるのも一つの方法ではある。

3. 実装する計算式

今回の研究で実装する計算方式は 3) で提案されている方式に基づいている。この手法は現在の世界記録を導く際にも用いられた方式であり、実際的に高速に動作することが期待できる。

まずは具体的な計算式を紹介する前に、その式中で使われている定数や変数などについて説明する。

3.1 関　　数

$\gamma(n)$ と $\delta(n)$ はそれぞれ引数 n の最大の素因数と最小の素因数を返す関数である。つまり

$$\gamma(n) = \max_{e_i>0} p_i$$
$$\delta(n) = \min_{e_i>0} p_i$$

と定義される。ただし $n = 1$ の場合は例外的に $\gamma(1) = \delta(1) = 1$ とする。

次に Möbius 関数についてである。これは引数 n が 1 以外の平方数で割れる場合には 0 を、そうでない場合は素因数が奇数個あれば -1 を、偶数個あれば 1 を返す関数である。数式で書くならば

$$\mu(n) = \begin{cases} 0 & (\text{if } \exists e_i \geq 2) \\ (-1)^{\sum e_i} & (\text{otherwise}) \end{cases}$$

ということになる。

3.2 パラメータ変数

パラメータ変数は、k と y と z の 3 つがある。これらのうち、k は分岐条件を抑えるためにメモリコストと計算量のトレードオフを調整するだけのパラメータであり、計算環境依存の定数と考えることができる。

一方、y と z は、$x^{1/3} \leq y \leq z < x^{1/2}$ を満たすように設定される。これらの設定次第では各項目での割り当てが異なり、全体的な計算時間が大きく左右される。本研究では主にこのパラメータの割り当て方について調査する。

3.3 定　　数

x^* は後述の y と x を用いて次のように定義される。

$$x^* = \max(x^{1/4}, x/y^2)$$

また、Σ_i の項の計算に用いられる定数 $a \sim d$ の定義は以下の通りである。

$a = \pi(y)$, $b = \pi(x^{1/3})$, $c = \pi((x/y)^{1/2})$, $d = \pi(x^*)$

3.4 計 算 式

具体的な計算式は以下のように纏められている。ここで A, B, C, D と ϕ_0, Σ_i は表記に一貫性は無いが、意味として何らかの違いがあるわけではない。

$$\pi(x) = A - B + C + D + \phi_0 + \sum_{i=0}^{6} \Sigma_i$$

そして各項の詳細は以下の通りである。なお、$\sum$ の中で変動する変数のうち p と q は素数だけを、n は自然数全体を動くものとする。

$$A = \sum_{x^* < p \le x^{1/3}} \sum_{p < q \le (x/p)^{1/2}} \chi\left(\frac{x}{pq}\right) \pi\left(\frac{x}{pq}\right)$$

$$\chi\left(\frac{x}{pq}\right) = \begin{cases} 1 & \left(\text{if } \frac{x}{pq} < y\right) \\ 2 & \left(\text{if } \frac{x}{pq} \ge y\right) \end{cases}$$

$$B = \sum_{y < p \le x^{1/2}} \pi\left(\frac{x}{p}\right)$$

$$C = -\sum_{p_k < p \le x^*} \sum_{n \in R_C} \mu(n) \left(\pi\left(\frac{x}{pn}\right) - \pi(p) + 2\right)$$

$$R_C = \{n \le z < pn, \delta(n) > p, \gamma(n) \le y, n > x/p^3\}$$

$$D = -\sum_{p_k < p \le x^*} \sum_{n \in R_D} \mu(n) \phi\left(\frac{x}{pn}, \pi(p) - 1\right)$$

$$R_D = \{n \le z < pn, \delta(n) > p, \gamma(n) \le y, n \le x/p^3\}$$

$$\phi_0 = \sum_{n \le z, \delta(n) > p_k, \gamma(n) \le y} \mu(n) \phi\left(\frac{x}{n}, k\right)$$

$$\Sigma_0 = a - 1 + \frac{\pi(x^{1/2})(\pi(x^{1/2}) - 1)}{2} - \frac{a(a-1)}{2}$$

$$\Sigma_1 = \frac{(a-b)(a-b-1)}{2}$$

$$\Sigma_2 = a\left[b - c - \frac{c(c-3)}{2} + \frac{d(d-3)}{2}\right]$$

$$\Sigma_3 = \frac{b(b-1)(2b-1)}{6} - b - \frac{d(d-1)(2d-1)}{6} + d$$

$$\Sigma_4 = \pi(y) \sum_{x^* < p \le (x/y)^{1/2}} \pi\left(\frac{x}{py}\right)$$

$$\Sigma_5 = \sum_{(x/y)^{1/2} < p \le x^{1/3}} \pi\left(\frac{x}{p^2}\right)$$

$$\Sigma_6 = -\sum_{x^* < p \le x^{1/3}} \pi\left(\frac{x^{1/2}}{p^{1/2}}\right)^2$$

3.5 計 算 量

この手法を提案している 3) によると、この計算式に基づく計算の時間オーダは

$$O\left(\frac{x^{2/3}}{\log^2 x}\right)$$

メモリオーダが

$$O\left(\frac{x^{1/3}}{\log^3 x \log\log x}\right)$$

とされる。

広く知られている Eratosthenes の篩を用いて x 以下の素数を全て求めようとする場合には

$$O(x \log\log x)$$

の計算時間オーダが必要となる[7] ため、この手法がいかに効率的であるかがわかる。

4. 実験と考察

3 節で述べた数式に基づくプログラムを C 言語を用いて作成した。ちなみにプログラムのサイズは全体で 700 行程度となっている。

4.1 想定マシン環境

今回の実験用のプログラムを作成するにあたり、対象とするマシン環境として東京大学情報基盤センターに設置されている HITACHI のスーパーコンピュータ SR11000/J1[8] を想定している。

SR11000/J1 は 2006 年 9 月現在、**表 2** のような形で運用されている。☆

表 2 SR11000/J1 の演算環境
Table 2 Environment of SR11000/J1

CPU 数	8CPU/node
演算速度	60.8GFLOPS/node
メモリ	64GB/node

4.2 k の 設 定

3 節で述べたように、式 (1) を実装するためには $\phi(x,k)$ の値を保持する P_k 個の配列を必要とする。ここで、1 つの値を格納するために 8Byte の容量が必要であると考えると、P_k の値から必要メモリ量は**表 3** のようになる。

この表から、$k = 8, 9, 10$ のいずれかが妥当な選択肢であると考えることができる。正確には $k = 8$ は一般的な PC でも十分使える程度、$k = 9, 10$ となると SR11000/J1 のような特殊な環境でならば使用可能ということになる。

ただし $k = 10$ の場合は 48GB もの容量を必要とするため、あまり実用性は無く、他のことに使うメモリが少なく見積もれる場合にしか使えない。このため、本研究では $k = 9$ として実験を行う。

☆ 本来の物理的な構成としてはノード辺りの CPU 数、理論速度性能、メモリ容量のいずれもが倍の値になっているものを、半分に分割することでノード数を倍にして運用している状態である。従って、運営形態が変わればノード辺りの性能は表 2 に示した値の倍になることもある。

表 3　P_k と必要メモリ
Table 3　P_k and memory

k	p_k	P_k	Memory
1	2	2	16B
2	3	6	48B
3	5	30	240B
4	7	210	1680B
5	11	2310	18KB
6	13	30030	234KB
7	17	510510	3.9MB
8	19	9899890	74MB
9	23	223092870	1.7GB
10	29	6469693230	48GB
11	31	200560490130	1.5TB

4.3　パラメータの選択

これらの環境を用いて、$x = 10^{11}, 10^{12}, 10^{13}$ において様々な y と z を設定した場合の計算に要する CPU 時間を計測した。(**図 2**, **図 3**, **図 4**) 正確には、$y = x^{i_y/300}$, $z = x^{i_z/300}$ と表記すると、i_y, i_z を $100 \leq i_y \leq i_z \leq 150$ の範囲の整数で動かし、各パラメータの組み合わせにおいて 1 回ずつの計測を行った。

グラフは縦軸が y、横軸が z で、下辺が $y = x^{1/3}$、上辺が $y = x^{1/2}$、左が $z = x^{1/3}$、右が $z = x^{1/2}$ となっており、また、右下のエリアは他と比較して時間勾配が急に上がるので、表示の都合上ある程度の範囲を持って切っている。

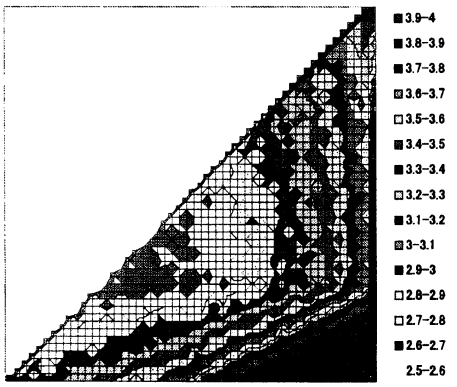

図 2　$x = 10^{11}$ でのパラメータごとの計算時間 [秒]
Fig. 2　Computation time at $x = 10^{11}$ [sec]

これらのグラフを見ると、どのグラフも傾向として $y = z = x^{114/300}$ 周辺が安定的に短い時間をとり、そこから離れるにつれて計算時間がかかるようになっている。

$x = 10^{11}$ では計測した時間の有効桁数の範囲から正確にどれが真に最短時間で計算したのかという判断はできないが、$x^{112/300} \leq y \leq x^{118}$, $x^{112/300} \leq z \leq x^{120/300}$ の辺りで短時間計算をしていることが分かる。そして $x = 10^{12}$ では $y = z = x^{114/300}$ で、$x = 10^{13}$ では $y = z = x^{113/300}$ で最短時間を記録しているが、こちらの場合も次に短いパラメータでの計算時間との差が計測単位である 0.1 秒であるため、真に最適なパラメータであるかどうか疑わしい。

また、$x^{1/300}$ の違いであっても、実際に y, z が取る値の範囲は広大なため、今回の実験で最適とされた周囲のパラメータをもっと細かい単位でばらつかせてみる必要がある。

3) によると理論的には $y = cx^{1/3}\log^3 x$, $z = dy$ とあるが、これに先程の数をあてはめるならば、$x = 10^{12}$ では $c = 1.72 \cdot 10^{-4}$、$x = 10^{13}$ では $c = 1.36 \cdot 10^{-4}$ となる。約 1.27 倍の違いがあるが、先程述べた測定誤差の範囲を考えれば、こちらも十分誤差の範囲内になる。

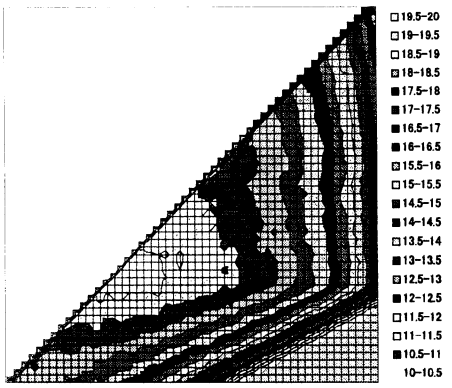

図 3　$x = 10^{12}$ でのパラメータごとの計算時間 [秒]
Fig. 3　Computation time at $x = 10^{12}$ [sec]

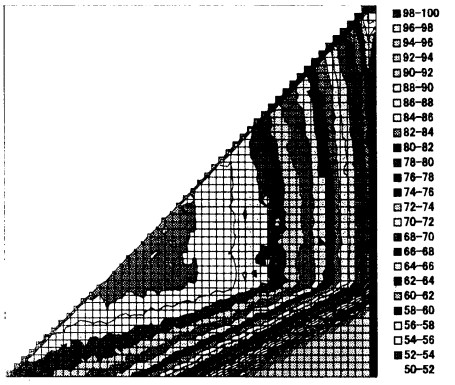

図 4　$x = 10^{13}$ でのパラメータごとの計算時間 [秒]
Fig. 4　Computation time at $x = 10^{13}$ [sec]

4.4 動作時間

動作させた時間の測定結果をグラフ (図 5) にした。これらの測定においては先程の結果を単純に考えて、全て $y = z = x^{113.5/300}$ のパラメータで実験している。

$10^9 \leq x \leq 10^{13}$ ではパラメータを確認したのでほぼ最短の時間が出るということは分かっているが、同じパラメータでどのように時間が延びるかを検証するため、$x = 10^{14}, 10^{15}, 10^{16}$ でも時間を計測してみた。(表 4)

これらの計算時間オーダは累乗関数で最小二乗近似すると

$$O(x^{0.66944})$$

となった。この値は理論的な時間オーダである

$$O\left(\frac{x^{2/3}}{\log^2 x}\right)$$

には近いが、それより上を行っていることが分かる。

表 4 $10^{11} \leq x \leq 10^{16}$ の計算時間
Table 4 Computaton time for $10^{11} \leq x \leq 10^{16}$

x	計算時間 [sec]
10^{11}	2.6
10^{12}	11.1
10^{13}	52.2
10^{14}	256.2
10^{15}	1189.1
10^{16}	5579.5

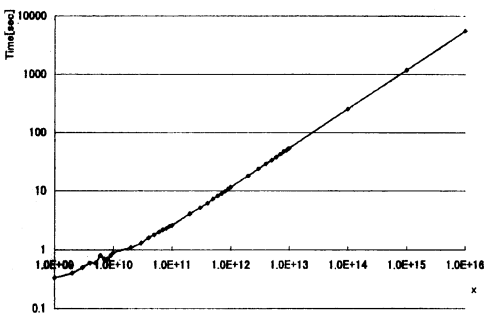

図 5 $\pi(10^9 \leq x \leq 10^{16})$ の計算時間
Fig. 5 Computation time of $\pi(10^9 \leq x \leq 10^{16})$

$x = 10^{14}$ と 10^{15}、$x = 10^{15}$ と 10^{16} を比較すると、時間はそれぞれ約 4.64 倍と約 4.69 倍となる。ここで理論時間オーダに x を当てはめ、その比を取ると約 4.04 倍と約 4.08 倍になるので、理論値よりも多少悪い傾向を示しているということがわかる。

しかし、$x = 10^{11}, 10^{12}, 10^{13}$ での計算時間も見るとほぼ同様の伸び方をしているため、理論時間オーダへ近づくためにはパラメータだけではなくプログラムそれ自体の効率化が必要である。

5. おわりに

本研究では、膨大な計算量を必要とする $\pi(x)$ の計算において、ある計算方式に基づいたプログラムを実装し、そのパラメータにおける計算時間の関係を探った。

しかし、計算可能な範囲では最良のパラメータを確定できる程度に計算時間に差が出なかったため、$y \simeq z \simeq x^{113/300}$ 程度という広く幅を取った結論しか出なかった。

今後は、大きな x についての計算を行うためにメモリの設計を変更しなければならない。というのも、現状では $x^{1/2}$ 以下の素数を保持するために 8Byte 整数を用いており、$x = 10^{23}$ を考えると 96GB ものメモリが必要となるためである。

また、並列化の必要もあるが、素数の保持方式の変更や複数の CPU を使うよう並列化することで最適なパラメータがどのように変化するかを確認する必要がある。

参考文献

1) H.Riesel: *Prime Numbers and Computer Methods for Factorization*, Birkhäuser (1994).
2) Gourdon, X.: The pi(x) project. `http://numbers.computation.free.fr/Constants/Primes/Pix/pixproject.html`.
3) Gourdon, X.: Computation of pi(x) : improvements to the Meissel, Lehmer, Lagarias, Miller, Odlyzko, Deleglise and Rivat method. `http://numbers.computation.free.fr/Constants/Primes/piNalgorithm.ps`.
4) Wolfram Research: mathworld. `http://mathworld.wolfram.com/PrimeCountingFunction.html`.
5) Lagarias, J. C., Miller, V. S. and Odlyzko, A. M.: Computing $\pi(x)$: The Meissel-Lehmer Method, *Mathematics of computation*, Vol.44, No.170, pp.537–560 (1985).
6) Deleglise, M. and Rivat, J.: Computing $\pi(x)$: The Meissel, Lehmer, Lagarias, Miller, Odlyzko method, *Mathematics of computation*, Vol.65, No.213, pp.235–245 (1996).
7) Lagarias, J.C. and Odlyzko, A.M.: Computing $\pi(x)$: An analytic method, *Journal of Algorithms*, Vol.2, No.8, ScienceDirect, pp.173–191 (1987).
8) 東京大学情報基盤センター：Computer Centre, the University of Tokyo HOME PAGE. `http://www.cc.u-tokyo.ac.jp/`.