An Introduction of Multiple *P*-adic Data Type and Its Parallel Implementation

Chao Lu
Department of Computer & Information Sciences
Towson University, 8000 York Rd
Towson, MD 21252, USA
E-mail: clu@towson.edu

Xinkai Li
Department of Computer & Information Sciences
Towson University, 8000 York Rd
Towson, MD 21252, USA
E-mail: lixinkai901@gmail.com

Abstract

Our research group has been working on the *P*-adic theory and its implementation. Based on the Chinese Remainder theorem and the Hensel code a new data type, called Multiple P-adic Data Type, has been established to realize rational calculation. With this data type all rational number operations are converted to integer calculations, and the fast integer multiplication of modern computer architectures can be fully taken advantage of. This data type can be significantly effective in the parallel and cloud computing environment due to its independent computation at each node during the calculation process.

Keywords: Parallel computing, Computational efficiency, *P*-adic, Multiple modulus, Chinese remainder theorem

1. Introduction

Solutions of very large dense linear systems are required for the estimation of the Earth's gravitational field, boundary element formulations in electromagnetism and acoustics, and molecular dynamics simulations [1]. One of the most important issues is to guarantee the accuracy of the results. The floating-point number system naturally has the defect of generating truncation error. When the matrix size is very large, the accumulated errors make the computational results unreliable and unacceptable. For example, using Gaussian elimination to calculate the matrix inverse of Hilbert matrix 15 by 15, with the double floating point number data type, the result is,

$$\begin{bmatrix} 151 & \cdots & \cdots \\ -11348 & 1138179 & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

While the actual result is

$$\begin{bmatrix} 225 & \cdots & \cdots \\ -25200 & 3763200 & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

Even when the matrix size is 15 by 15, the accumulation of truncation errors make the results unacceptable under the double floating point number system. Error free rational calculation is necessary. But most of the existing symbolic data type, such as in Matlab or Mathematic, the time cost is considered too much. During the past few years, we have been working on *P*-adic theory and its implementation. Based on the Chinese Remainder Theorem and Hensel code, a new

This research project is supported by the US Air Force Office of Scientific Research. FA9550-11-1-0315.

data type has been established to realize a rational calculation called Multiple P-adic Data Type [2-4]. With this data type all rational number operations are converted to integer calculation, and the fast integer multiplication of modern computer architectures can be fully used. This data type can be significantly effective in the parallel and cloud computing environment due to its independent computation at each node during the calculation process. Furthermore, the existing C++ programs can be converted to run with this data type with no (or minimal) changes to the source code. We have developed a computational library based on the data type and the object oriented program using C/C++. Computational algorithms have been developed using the data type for the calculation of matrix inverse, Lower Hessenberg form transformation, Wilkinson form transformation, Frobenius form transformation, post processing from all the transformations, reflexive general matrix inverse, Moore-Penrose inverse, e^{At} calculation, Laplace's method for FTA (Fundamental Theorem of Algebra), Bézoutian formulation of the Resultant and etc.

2. Background

We introduce the theoretical background of the Multiple P-adic Data Type briefly here. The main idea is coming from the Hensel code and the Residue Number System [14]. Hensel code was originally defined by Krishnamurthy, Rao and Subramanian, which was developed from the P-adic number system first proposed by Hensel in the 1900s. The purpose was to realize exact computation for rational numbers. Based on the Chinese Remainder Theory, Morrison devised the method to realize paralleled P-adic arithmetic for rational numbers [2]. The detail can be found in our previous papers [4-8].

2.1. Finite P-adic (Hensel code) Arithmetic[8-11]

Any rational number can be coded into P-adic sequence by the following algorithm: $\alpha = \frac{b}{c} \cdot p^n, a, b, n \in N, b \neq 0$, and $GCD(a,b)$, $GCD(a,p), GCD(b,p) = 1$ can be written as $\propto = \sum_{i=k}^{\infty} a_i p^i , k \in N$.
The conversion process is following:

Sep 1. $\propto \bmod p = a_0$
Sep 2. $\propto = (\propto - a_1)/p$, go to Step 1 to get a_1.
Continue Step 1 and Step 2, to get a_i
Finally, $\alpha = p^n \cdot \sum_{i=0}^{\infty} a_i p^i = \sum_{i=n}^{\infty} a_{i-n} p^i$

The P-adic sequence with point position n will have the following form:

$$a_n a_{n+1} \cdots a_{-2} a_{-1} . a_0 a_1 a_2 \cdots \qquad for \ n < 0$$
$$.a_0 a_1 a_2 \cdots \qquad for \ n = 0$$
$$.000 a_0 a_1 a_2 \cdots \qquad for \ n > 0$$

Conventionally, we write P-adic sequence as:

$a_{i-n} a_{i-n+1} \cdots$ point position = n
point position means the position of a_0 .

2.1.1. Addition/Subtraction

The addition of P-adic is similar to the binary numeral addition. The difference is that P-adic addition process is calculating from left to right.
For example of computing $\frac{1}{6} + \frac{1}{2} = \frac{2}{3}$ for p = 5:

$$\frac{1}{6} = .140404040 \ldots \quad \frac{1}{2} = .32222222 \ldots$$

In the process, the position of the point should be kept in alignment,

$$\begin{array}{r} .140404040\cdots \\ \underline{.322222222\cdots} \\ .413131313\cdots \end{array}$$

We can check that the 5-adic of

$$\frac{2}{3} = .413131313 \ldots$$

2.1.2. Multiplication/Division

The multiplication of P-adic is similar to the binary numeral multiplication, the difference is that P-adic multiplication is calculating from left to right. The point position of the result equals: point1 + point2.
For example of $\frac{1}{3} \times \frac{1}{6} = \frac{1}{18}$ with p = 5:

$$\frac{1}{3} = .231313131\cdots$$
$$\frac{1}{6} = .140404040\cdots$$

The multiplication can be shown:

```
        .2313131313131 · · ·
      × .1404040404040 · · ·
-----------------------------------------
        2313131313131 · · ·
          331313131313· · ·
           00000000000 · · ·
             3313131313 · · ·
          000000000 · · ·
           33131313 · · ·
             0000000 · · ·
              331313 · · ·
               00000 · · ·
                3313 · · ·
                 000 · · ·
                  33 · · ·
 +                 0· · ·
---------------------------------------------------
        .2103341103341 · · ·
```

Check the result with 5-adic representation:

$$\frac{1}{18} = .21033411033411.....$$

Division can be carried out as a multiplication process. First we use the recursive method to get the inverse of the dividend then complete the multiplication. The point position for division equals: point1 - point2.

2.1.3. Hensel code

The encoded *P*-adic sequence is usually infinite. The way to choose a finite *P*-adic sequence used in exact rational computation is called Hensel code arithmetic. The Hensel codes are closed with respect to basic arithmetic operations (ADD/SUBTRACT/MULTIPLY/DIVIDE).

For each Hensel code $H(p,r,\propto)$, p means the prime, r means the length of the *P*-adic sequence, $\propto$ means the finite *P*-adic sequence.

2.2. Multiple P-adic Algorithm

Multiple *P*-adic algorithm is first introduced by Morrison [2]. The algorithm is based on extending the Chinese remainder theorem from integers to rational numbers [4].

The decoding process is as follows,

If $r \sim \{r_1, r_2, \cdots, r_s\}$ is the residue representation of a rational number r with respect to moduli $\{p_1, p_2, \cdots p_s\}$ where $GCD(p_i, p_j) = 1$ for $i \neq j$, then the decoding algorithm is given as follows [10] [13]:

Decoding algorithm

```
Step 1:  Chinese remainder theorem
         $p = \prod_{i=1}^{s} p_i$
         For $i = 1$ to $s$
         Using extended Euclidean algorithm
         to find $p_i'$ by $\frac{p}{p_i} p_i' \equiv 1 \ mod\ p_i$
         End
         $\bar{r} = \sum_{i=1}^{s} \frac{p}{p_i} p_i' r_i \ mod\ p$
Step 2:  Euclidean algorithm
         $u_{-1} = p, u_0 = \bar{r}$
         $v_{-1} = 0, v_0 = 1$
         $i = -1$
         While $u_i < \sqrt{p}$
                  $q_i = \lfloor u_{i-1}/u_i \rfloor$
                  $u_{i+1} = u_{i-1} - q_i u_i$
                  $v_{i+1} = v_{i-1} + q_i v_i$
                  $i++$
         End
Rational solution:
         $r = ((-1)^i u_i / v_i)$
```

Combining *P*-adic arithmetic [7-10] with the extended Chinese remainder theorem (CRT), the multiple *P*-adic algorithm is established. The implementation flow chart follows:

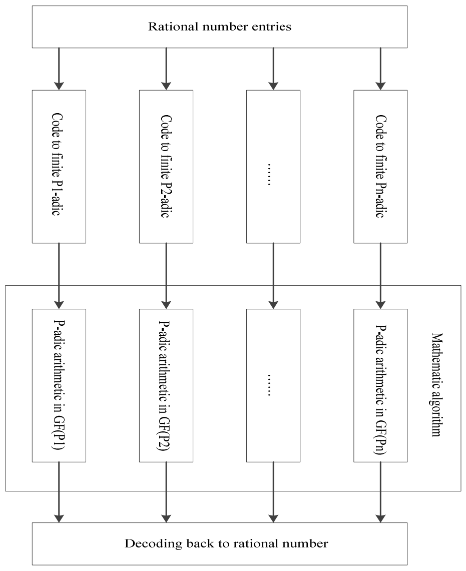

Figure 1. Extended CRT combined with *P*-adic arithmetic for parallel implementation

2.3. Overflow Detection [5] [10]

Extended CRT overflow:
For the extended CRT systems with the prime set $\{p_1, \cdots p_s\}$, when a rational number $\frac{b}{a} \bmod p_i$ set $r \sim \{r_1, \cdots, r_s\}$, $GCD(a, b) = 1$, satisfies $|a, b| > \lambda\sqrt{p}, p = \prod_{i=1}^{s} p_i$ ($\lambda = 0.618 \cdots$ is a root of $\lambda^2 + \lambda - 1 = 0$), the rational number, whose set cannot be uniquely recovered by the inverse transformation. We call this situation extended CRT overflow. One way to detect the overflow is to predict the bound, then decide the size of the prime set $\{p_1, p_2, \cdots p_n\}$ by Newman [14]. Another way to detect overflow is to provide extra digits [5]. This method can detect the overflow by using the prime set $\{p_1, \cdots p_n\}$ and the residue number set itself. In this method, each number set should have extra digits used for verification, the length is kept by k.
With prime set $\{p_1, p_2, \cdots p_i, p_{i+1}, \cdots, p_{i+k}\}$, for any rational number x, we get the set$\{r_1 = x \bmod p_1, \cdots, r_{i+k} = x \bmod p_{i+k}\}$, we record it as: $x \sim \{r_1, r_2, \cdots, r_i, r_{i+1}, \cdots, r_{i+k}\}$.
During the overflow detection process, it will be treated as

$$x \sim (r_0 r_1 \cdots r_i \quad \underbrace{r_{i+1} \cdots r_{i+k}}_{verification\ part}).$$

Notation: *Decoding(x, i)* and *Decoding(X, i)* will be used to donate decoding rational number set *x* and matrix *X* into rational number and rational number matrix by first *i* digits.
Overflow happened, if:

$$Decoding(x, i) \neq Decoding(x, i + k).$$

Overflow did not happen, if:

$$Decoding(x, i) = Decoding(x, i + k).$$

3. The Main Properties of Multiple *P*-adic Data Type

3.1. Error-free Computing in Rational Number Field

Each rational number is represented by a finite sequence of integers. The integers' values are the module results of prime numbers, which can be chosen by developers and also depend on the CPU architectures. The arithmetic calculation process is in a rational number field, thus there will be no truncation error. The arithmetic is transformed to integer arithmetic and module operations.
The structure of the data type can be explained as the following:

$$\frac{b}{a}: \underbrace{\boxed{Integer\ 00}\boxed{Integer\ 01}\cdots\boxed{Integer\ 0k}}_{module\ results\ of\ P_0}$$

$$\cdots \underbrace{\boxed{Integer\ n0}\boxed{Integer\ n1}\cdots\boxed{Integer\ nk}}_{module\ results\ of\ P_n}$$

For example, form prime number set $[257, 251, 241]$, then,

$$\frac{1}{10234567}: \underbrace{\boxed{179}\boxed{235}}_{module\ results\ of\ 257} \quad \underbrace{\boxed{6}\boxed{193}}_{module\ results\ of\ 251}$$

$$\underbrace{\boxed{229}\boxed{114}}_{module\ results\ of\ 241}$$

The integer sequence for $\frac{1}{10234567}$ is:

$$179, 235;\ 6, 193;\ 229, 114.$$

The size of the prime number set and the length for the integer sequence can be self-defined. The size of the prime number set can affect the efficiency of parallel computing. The detail explanation will be introduced in the natural parallel ability section. Because the arithmetic operation is in rational number field as integers, there will be no truncation error. For example of the calculation of $1/2 + 1/3$ with the prime set[257,251,241],

$$\frac{1}{2}: 129{,}128; 126{,}125; 121, 120$$

$$\frac{1}{3}: 86,\ 171;\ 84{,}167; 161, 160$$

$$\begin{array}{rrrrrrr} & 129 & 128 & 126 & 125 & 121 & 120 \\ + & 86 & 171 & 84 & 167 & 161 & 160 \\ \hline & 215 & 42 & 210 & 41 & 41 & 40 \end{array}$$

The sequence: $215{,}42; 210{,}41; 41{,}40$ is transformed to $\frac{5}{6}$

3.2. Integer Calculations Taking Full Use of Computer Architecture

Usually the rational calculation is using arbitrary length integers to represent the numerator and denominator. For example, we use one character (1 byte) to represent each digit of an integer and then link the data structure to realize the arbitrary length of the integer. For example, when calculating $1234567 + 7654321$

$$\begin{array}{rccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ + & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ \hline & 8 & 8 & 8 & 8 & 8 & 8 & 8 \end{array}$$

Including the carry-out operations, there will be 16 possible character operations. While using the Multiple P-adic Data Type, and choosing [46337] as the prime set, the calculation process will be

$$\begin{array}{rrrr} & 29805 & 26 & 0 \\ + & 8716 & 165 & 0 \\ \hline & 38521 & 191 & 0 \end{array}$$

There are only 3 additions and 3 module operations.
For rational operation, the numerator and denominator will cost more due to the reasons shown below:
Addition process:

$$\frac{a_1}{b_1}+\frac{a_2}{b_2}=\frac{a_1b_2+a_2b_1}{b_1b_2}$$

$$=\frac{(a_1b_2+a_2b_1)/GCD((a_1b_2+a_2b_1),b_1b_2)}{b_1b_2/GCD((a_1b_2+a_2b_1),b_1b_2)}$$

Multiplication process:

$$\frac{a_1}{b_1}\times\frac{a_2}{b_2}=\frac{a_1a_2}{b_1b_2}=\frac{a_1a_2/GCD(a_1a_2,b_1b_2)}{b_1b_2/GCD(a_1a_2,b_1b_2)}$$

During these calculation processes, $GCD(numerator, denominator)$ must be found and extra calculation steps will be needed. However, for Multiple P-adic Data Type there will be no difference between fractions and integers.
The Multiple P-adic Data Type can be easily implemented on 32 and 64-bit platforms. The only difference is to choose the right prime number set. On the 32-bit platform, the maximum prime is $P = 46337$, the largest prime numbers smaller or equal to 46337 can be used, while on the 64-bit platform, the maximum prime will be $P = 2147483647$, the largest prime numbers smaller or equal to 2147483647 can be used.

3.3. Natural Parallel Structure Taking Full Use of Multi-core System

According to the features of the Multiple P-adic Data Type, parallel computing can be implemented on any algorithm with basic arithmetic operations and do not depend on the specific algorithm. The parallel structure depends on the size of the chosen prime set. For example, we calculate $(\frac{1}{17}-1)\times\frac{1}{2}$ with prime set$[257,251{,}241]$,

$$\frac{1}{17}: 121{,}60; 192{,}14; 156{,}212$$
$$1: 1{,}0; 1{,}0; 1{,}0$$
$$\frac{1}{2}: 129{,}128; 126{,}125; 121{,}120$$

$$\underbrace{\boxed{\begin{array}{crr} \multicolumn{3}{l}{257\ Line\ Process} \\ & 121 & 60 \\ - & 1 & 0 \\ \hline & 120 & 60 \\ \times & 129 & 128 \\ \hline & 60 & 30 \end{array}}\quad\boxed{\begin{array}{crr} \multicolumn{3}{l}{251\ Line\ Process} \\ & 192 & 14 \\ - & 1 & 0 \\ \hline & 191 & 14 \\ \times & 126 & 125 \\ \hline & 221 & 132 \end{array}}\quad\boxed{\begin{array}{crr} \multicolumn{3}{l}{241\ Line\ Process} \\ & 156 & 212 \\ - & 1 & 0 \\ \hline & 155 & 212 \\ \times & 121 & 120 \\ \hline & 198 & 226 \end{array}}}_{3\ Separated\ Process}$$

After the 3 separated computing processes, we can get the final result: $-\frac{8}{17}: 60{,}30; 221{,}132; 198{,}226$.
Most of the linear processes can directly use this data type to realize parallel computing without modification at the mathematical algorithm level. We have implemented this data type to calculate matrix inverse, Moore-Penrose inverse (General Inverse) and e^{At}.

3.4. Easy for Task Allocation in Cloud Environment

Using the Multiple P-adic Data Type, the total work load is homogeneously allocated into small parts which is as many as the size of the prime set. It will be easier for making task allocations in a cloud environment. Furthermore in the symbolic (numerator-denominator) rational number calculation process, as the size of the arbitrary length number grows, the memory cost will increase quickly, while the Multiple P-adic Data Type will not have that kind of problem. The memory cost for each key will not grow during the calculation process. It is easy to estimate the memory cost before the calculation.

4. Practical Considerations In A Cloud Environment

Using Multiple P-adic Data Type, each module is independent of others, so that each can be computed on a different cluster node and can be done not necessarily at the same time. At each cluster node, a parallel algorithm can also be implemented during the matrix calculation process on multi CPU cores. If the matrix size is too large, the block algorithm can be freely implemented in the calculation process. The efficient formula for calculation time is the following:

$$\begin{aligned} &Total\ Calculation\ Time\ Cost \\ &= Basic\ Integer\ Calculation\ Time\ Cost \\ &\times \left\lceil \frac{Calculation\ Complexity}{Number\ of\ CPU\ Used} \right\rceil \end{aligned}$$

Basic Integer Calculation Time means the time cost on one node calculation by one module prime (64bits integer type or 32bits integer type). Calculation Complexity means the necessary prime set length with no overflow happening. Of course, the total time cost must also include the communication cost between nodes and the host at the beginning and the end of each node calculation.
The implementation process is the following:

1) Analyze the work load. According to the matrix size, the complexity of the number and the matrix calculation algorithms, the size of prime set *s* and the length of the *P*-adic sequence *r* will be decided. For a specific matrix transform, there will be a specific algorithm chosen or created for the work load evaluation. For example, to calculate $Ax = b$, Hadamard's inequality will be used: $2max(n^{\frac{n}{2}}M(A)^n, n(n-1)^{\frac{n-1}{2}}M(A)^{n-1}M(b)) \leq \lambda\sqrt{\prod_{i=1}^{s} p_i^r}$, where $\lambda = 0.618\cdots$ is a root of $\lambda^2 + \lambda - 1 = 0$, $M(X)$ means the largest value of denominator or numerator among elements in matrix *X*, *s* means the number of cluster nodes assigned, and *r* usually can represent the calculation efficiency. The smaller *r* means the less calculation time and less memory usage for a cluster node, while the smaller *r* requires larger *s*, which means to assign more cluster nodes.

2) Work load separation. The original matrix data and a specific prime from a prime set will be sent to different cluster nodes. In each node, the original matrix elements will be module by the prime and generate *P*-adic sequence with length equals to *r*. Then the matrix transformation will be calculated. During this process, parallel and block algorithms can be freely implemented.

3) Generate the final result. All the temporary results will be collected from various nodes in the cloud by the master (host). The final rational result will be generated on the host machine. The Hensel code overflow detection will be used for verification. If overflow does not happen, we get the final result. Otherwise, keep the temporary results and choose a different prime set, then go to step *1)*.

4.1. Compare with the MATLAB Symbolic Toolbox

This new data type can significantly shorten the calculation time by using more CPU-cores. We have compared the calculation time for matrix inverses with the symbolic toolbox in MATLAB, the experimental results are given in Fig. 2. The computer used is the Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz for the experiment. This CPU has 8 CPU cores.
The following results (Fig. 2) show the calculation time of the inverse of $Hilbert\ matrix \times Hilbert\ matrix$.

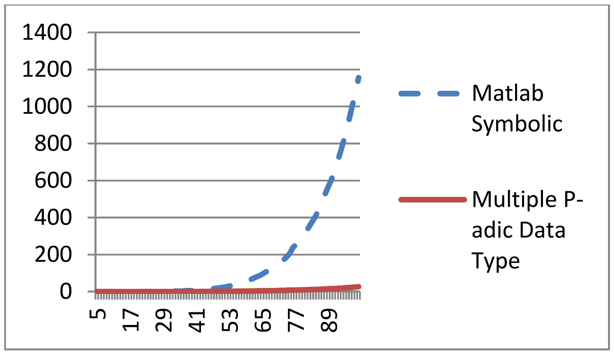

Figure 2. Vertical axis: calculation time (seconds)
Horizontal axis: matrix size (N x N)

The MATLAB code is following:

```
for n = 5:100
        A = hilb(n);
        B = sym(A);
        B = B * B;
        n
        tic
        inv(B);
        t1(n) = toc;
        toc
end
```

During the calculation process, Matlab uses only one core of CPU to do the calculation, while our data type takes full use of all the 8 cores.

4.2. Compare with the Mathematic Symbolic Toolbox

We have compared the calculation time for Moore-Penrose inverses with a symbolic toolbox in Mathematica 8, the experimental results are given in Fig. 3 and Fig. 4.
Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz is used to do the experiment. This CPU has 8 CPU cores. The

following results (Fig. 3) show the calculation time of the inverse of $Hilbert\ matrix \times Hilbert\ matrix$.

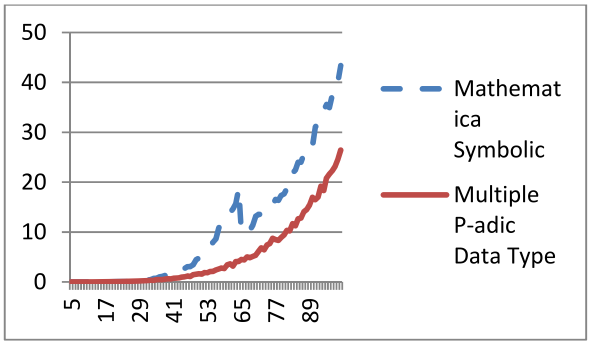

Figure 3. Vertical axis: calculation time (seconds)
Horizontal axis: matrix size (N x N)

The Mathematica code is following:

```
Array[f,100];
For[i=5,i<105,i++,s =
HilbertMatrix[i]*HilbertMatrix[i];f[i-4]
= Timing[Inverse[s];][[1]];Print[f[i-
4]]]
```

The following results (Fig. 4) show the calculation time of the Moore-Penrose inverse of $Hilbert\ matrix \times Hilbert\ matrix$.

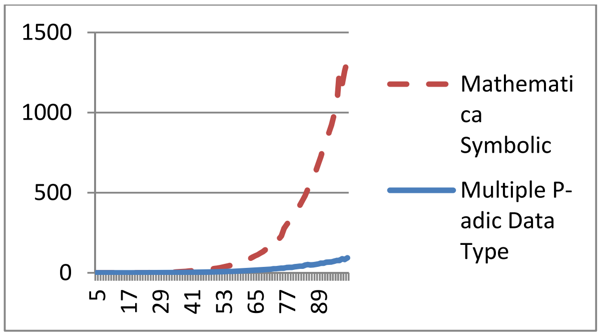

Figure 4. Vertical axis: calculation time (seconds)
Horizontal axis: matrix size

The Mathematica code is following:

```
Array[f,100];
For[i=5,i<105,i++,s =
HilbertMatrix[i]*HilbertMatrix[i];f[i-4]
=
Timing[PseudoInverse[s];][[1]];Print[f[
i-4]]]
```

During the calculation process, Mathematica uses 4 cores of the CPU, while our data type takes full use of all the 8 cores of the CPU. If the input matrix is more complex, the advantage of this new data type is more obvious. This observation can be shown by comparing Fig. 3 and Fig. 4. Calculating the Moore-Penrose Inverse is more complex than the general matrix inverse.

5. Estimate The Data Sequence Length

One of the important issues for using Multiple P-adic Data Type is to estimate the entry sequence length of Multiple P-adic Data Type. As we mentioned, we can estimate the largest possible value to determine the entry sequence length. But this way usually gives a huge length, which is far larger than actually needed. The better way to calculate that is to establish estimate functions depending on the property of different matrix transforms.

For example, we generate an estimate function for non-singular matrix inverse. We collect data by the following experiment. Generate random matrices from 4 by 4 to 60 by 60. For each matrix size, we generate matrix elements with denominator and numerator values in the range of {[0, 10], [0, 50], [0, 100], [0, 500], [0, 1000], [0, 5000], [0, 10000]}. For each range, we generate 15 random matrices. For each n by n matrix m_{ij}, we collect the data as the following:

$$u = \prod_{i=1}^{n} \sum_{j=1}^{n} {m_{ij}}^2.$$

And calculate

$$L = \log_P(2u^2 + 1),$$

where P means the prime value, in this experiment $P = 2147483647$

After doing matrix inverse, we collect the max value of denominator and numerator x from the inversed matrix, and calculate

$$l = \log_P(2x^2 + 1),$$

$$\theta = \frac{l}{L}.$$

Fig. 5 shows the experimental results.

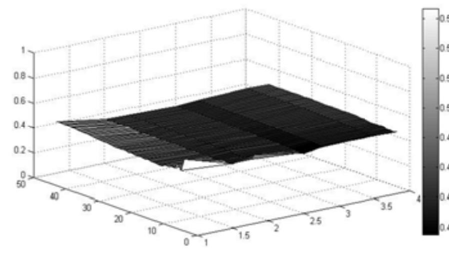

Figure 5. Vertical axis: θ
Left Horizontal axis: matrix size
Right Horizontal axis: $\ln(range)$

From Fig. 5, we can find the value of θ in a specific range. We calculate the average value $\tilde{\theta} = 0.45$ and standard deviation $\sigma = 0.01$.
Usually for the convenience of separating resources, we choose the same length for each P-adic sequence. The estimate function for non-singular matrix inverse is

$$r_i = \frac{\tilde{\theta}L}{s}.$$

If the results fail on overflow detection, we will keep the temporary results and choose a new prime as P-adic sequence to calculate with the length $r_{i+1} = 3\sigma L$.

6. Analysis and Conclusion

Multiple P-adic Data Type has a natural parallel structure, and is completely independent at each node during the calculation process, which is significantly effective in the parallel and cloud computing environment. Experimental results of the matrix inverse calculation with various matrix sizes are given to illustrate computational efficiency. When the matrix sizes are large and the computation is complex, the advantage of the Multiple P-adic Data Type will be more obvious if the computer has more cores/CPUs. An estimation method for choosing the sequence length of Multiple P-adic Data Type is provided for practical implementation.

References

1. M. Marques, G. Quintana-Orti and E. S. Quintana-Orti, Solving “Large” Dense Matrix Problems on Multi-Core Processors, In: 10th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing-PDSEC 2009.
2. J. Morrison, “Parallel P-adic computation, Information Processing Letters”, Vol. 28, Issue 3, 1988.
3. C. Limongelli and H. W. Loidl, “Rational Number Arithmetic by Parallel P-adic Algorithms”, Springer Verlag, editor, Proc. Of Second International Conference of the Austrian Center for Parallel Computation (ACPC), Vol. 734 of LNCS, 1993.
4. X. Li, C. Lu and J. A. Sjogren, “Parallel Implementation of Exact Matrix Computation Using Multiple P-adic Arithmetic”, International Journal of Networked and Distributed Computing (IJNDC) - Atlantis Press, 1(3), August, 2013.
5. X. Li, C. Lu and J. A. Sjogren, “A Method for Hensel Code Overflow Detection”, ACM SIGAPP Applied Computing Review, Vol. 12, Issue 1, p. 6-11, 2012.
6. C. Lu, X. Li and L. Shan, “Periodicity of the P-adic Expansion after Arithmetic Operations in P-adic Field”, ACIS2012.
7. X. Li, M. Zhao, C. Lu and J. A. Sjogren, “Implementation of the Polynomial Method to Calculate e^{At} Using P-adic”, Proceedings of the 2012 ACM Research in Applied Computation Symposium, 2012.
8. X. Li, M, Zhao and C. Lu, “Efficient Algorithms and Implementation for Error-free Computation Using P-adic”, CSNI2011.
9. C. K. Koc, A Tutorial on P-adic Arithmetic, Oregon State University, Technical Report, April 2002.
10. J. D. Dixon, “Exact Solution of Linear Equations Using P-adic Expansions”, Number. Math. 40, 137-141(1982), Springer- Verlag.
11. E. V. Krishnamurthy, Matrix Processors Using P-adic Arithmetic for Exact Linear Computations, IEEE Transactions on computers, Vol. C-26, No, 7, July 1977.
12. C. K. Koc, “Parallel P-adic Method for Solving Linear Systems of Equations”, Parallel Computing, Vol. 23(13), 1997.
13. M. Newman, “Solving Equations Exactly”, Mathematics and Mathematical Physics, Vol. 71B, No. 4, Oct-Dec 1967.
14. P. Kornerup and D. W. Matula, Finite Precision Number Systems and Arithmetic, Cambridge University Press, 2010.
15. T. M. Rao, K. Subramanian and E. V. Krishnamurthy, “Residue Arithmetic Algorithm for Exact Computation of g-inverses of Matrices”, SIAM J. NUMER. ANAL, Vol. 13, No. 2, pp. 155-171, April 1976.
16. R. T. Gregory, “The Use of Finite-segment P-adic Arithmetic for Exact Computation”, BIT, 18(3):282-300, 1978.
17. K. Hensel, “Theorie der Algebraischen Zahlen”, Teubner, Leipzig-Stuttgart, 1908.

18. D. M. Young and R. T. Gregory, “A Survey of Numberical Mathematics”, Addison Wesley, Reading, Mass, 1973.
19. P. Kornerup, R. T. Gregory, “Mapping Integers and Hensel Codes onto Farey Fractions”, BIT 23, 9-20, 1983.
20. M. Miola, “The conversion of Hensel Codes to Their Rational Equivalents: or How to Solve the Gregory’s Open Problem”, ACM Sigsam bulletin, vol. 16, Issue 4, November 1982.
21. G. Bachman, Introduction to P-adic Numbers and Valuation Theory, Acdemic Press, New York, 1964.