Application of multiple-precision arithmetic to direct numerical computation of inverse acoustic scattering

Hiroshi Fujiwara and Yuusuke Iso
Graduate School of Informatics, Kyoto University, Kyoto, 6068501, Japan
E-mail: fujiwara@acs.i.kyoto-u.ac.jp

Abstract. The aim of the paper is to show an effective application of multiple-precision arithmetic to numerical computations of ill-posed problems. Multiple-precision arithmetic enables us to realize virtually numerical computations without rounding errors, and we apply it, without any stabilization methods, to an inverse acoustic scattering problem to obtain remarkable results. We also introduce our new multiple-precision arithmetic environment exflib, which is designed and implemented for fast computation of large scale scientific numerical simulations, and it works with the language C++ and FORTRAN90.

1. Introduction
In the present paper, use of the multiple-precision arithmetic is proposed as a new approach to numerical computation of inverse and ill-posed problems. Since inverse problems arising in engineering, geophysics, or medical science are usually ill-posed in the sense of Hadamard, their discretization schemes ought to be numerically unstable. The instability yields the rapid growth of computational errors in their numerical processes, and numerical computation fails. In order to prevent it, a stabilization method such as the Tikhonov regularization [1] is often used in conventional way, but it obliges us to satisfy with so-called regularized solutions which are sometimes far from exact solutions. It is desired to establish a new accurate and reliable numerical reconstruction algorithm in numerical analysis of inverse and ill-posed problems.

In the standard numerical computation, floating-point numbers are used to approximate real numbers and their calculation is defined as floating-point arithmetic. The double precision arithmetic defined in IEEE754 [2] is most widely used today in the programming language C and FORTRAN on computers, and it has 53 bits precision in binary, which corresponds to about 15 decimal digits accuracy. We can not evade errors between an exact value and a floating-point number in either representation or arithmetic. The errors are called rounding errors.

When we restrict ourselves to well-posed differential equations or integral ones, their discretization schemes should be expected to be stable with some suitable condition, and the influence of rounding errors are not serious. Inverse problems are usually ill-posed and their discretization schemes become unstable, and the instability leads us to the fatal stage that computational errors growth rapidly; the direct numerical computation is impossible.

To overcome the difficulty we have proposed the use of multiple-precision arithmetic [3, 4]. Some multiple-precision arithmetic softwares have been used in the research of number theory.
and cryptography, but they are not standard in usual numerical computation, since it requires very much cost in computational time and memory. Numerical simulations of partial differential equations or integral equations often become large scale problems, and they require much computing time and memory. It is inevitable, in application of multiple-precision arithmetic to inverse problems, to design and implement a software which is suitable for computations of large scale scientific numerical simulations, and we have succeeded and introduce our software in the present paper.

In the proposed multiple-precision arithmetic environment we approximate real numbers and their arithmetics with arbitrary precision. In other words, we can reduce the rounding errors arbitrarily, and we can virtually realize numerical computation without rounding errors. Taking enough accuracy dependent on each problem, we are able to hide the influence of rounding errors and realize the direct numerical computation of ill-posed problems as far as we ignore observation errors.

We will discuss our proposed multiple-precision library exflib in the next section. In the current research we perform the direct numerical computation of an inverse acoustic scattering problem and discuss it in the section 3.

2. Multiple-Precision Arithmetic Environment for 64-bit Computing

We shall introduce a new multiple-precision arithmetic environment which is designed and implemented by the authors and which is used in numerical computations shown in the section 3. Our environment is designed for fast computations of large scale scientific numerical simulations with arbitrarily accurate approximation of real numbers in the programming language C++ and FORTRAN90 with polymorphic interface. Since our target is the precision of one hundred up to several thousands decimal digits, we choose optimal arithmetic algorithms for the range based on ALU operation in 64-bit computer architectures.

On digital computers main data structures for a number are a fixed point type (integer) and a floating-point type, which are processed with ALU (Arithmetic Logical Unit) and FPU (Floating Point Unit) respectively. ALU results are always exact except carry, and FPU results include rounding error. Employing ALU instructions, we can use all the bits to store a multiple-precision number. On the other hand, using FPU instructions, we can not avoid memory waste. For example, in double precision floating-point multiplication, both the multiplicand and the multiplier have at most 26 significant bits because the double precision type is at most 53 bits precision, hence $(1 - 26/64) \times 100 \sim 60\%$ memory storage is wasted. The advantage of FPU is that FPU multiplication and division is faster than those of ALU on 32-bit processors. In fact, some FORTRAN multiple-precision packages which use double precision arithmetic have been developed for fast computation on 32-bit processors [6]. Nowadays personal computers, workstations, and supercomputers are designed as 64-bit architecture and we do not utilize computational resources for multiple-precision arithmetic softwares developed for 32-bit architectures, because both an integer and a double-precision number have 64-bit width in 64-bit architectures and because ALU is more advantageous in accuracy than FPU in 64-bit architectures.

For the effective use of 64-bit architecture computer, we have defined a data structure and designed arithmetic algorithms for the proposed multiple-precision environment, which is based on 64-bit integers and ALU operations. Each real number is approximated and normalized in the form $(-1)^s \times 2^e \times 1.F$, where a sign part $s$ has one bit, an exponent part $e$ has 63 bits, and a fraction part $F = f_1f_2\cdots f_n$ has $64 \times n$ bits (each $f_i$ has 64 bits). We store it in memory with a 64-bit unsigned long integer array with $(n + 1)$ elements shown in Fig. 1. Hence a real value
of a number represented in Fig. 1 is

\[ (-1)^s \times 2^{e_b-b} \times \left( 1 + \sum_{i=1}^{n} f_i \times 2^{-64i} \right), \]

where \( b = 2^{62} - 1 \) is a bias and its accuracy is \( 64n + 1 \) bits (\( \approx 19.26 \times n \) decimal digits). We may choose the parameter \( n \) to attain required precision in each computation. We mainly use unsigned 64-bit integer instructions on ALU in multiple-precision arithmetic algorithms.

We implement basic arithmetic operations with rounding control, comparisons, output and assignment operations, and basic mathematical functions. Our aim is to perform fast computation in a range of one hundred up to several thousands decimal digits at most. We use the classical elementary algorithms for addition and multiplication, Karatsuba’s method for multiplication with large digits, and Ozawa’s method for division [7]. Basic arithmetic rules of multiple-precision numbers, and multiplication and division of multiple-precision numbers by integers are written in an assembly language for fast computing. Since ALU operations do not contain rounding errors, we can use all the 64 bits in a 64-bit integer. This enables us to represent a multiple-precision number with less memory, and less memory access causes fast computation. We note that the proposed environment works in parallel computations with MPI or OpenMP.

The proposed multiple-precision arithmetic environment provides a polymorphic interface for the four basic arithmetic rules, comparisons, assignments, and mathematical functions in the programming languages C++ and FORTRAN90. More concretely we use class and operator overloading features in the language C++, and module and user defined interface features in FORTRAN90. Polymorphism enables us to write the arithmetic statements with multiple-precision numbers as same as a built-in type numbers, and it also enables us to realize high productivity and portability of existing programs.

We discuss assignment algorithm precisely. For example, in the language C++, a statement

\[ x = 0.1; \]

is ambiguous to determine the precision of assigned value to \( x \) even if \( x \) is declared as a multiple-precision type; for the literal 0.1 on the right hand side is interpreted as a double or single precision number and it includes rounding error in binary. To specify precision of the assignment operators, we provide assignment operators of multiple-precision number with a string. In proposed interface we can set a literal constant in decimal, a mathematical constant such as \( \pi \), and their arithmetic in strings on the right hand sides. For multiple-precision type number \( x \), \( y \), and \( z \), the following are valid in the language C++:

\[ x = "0.1"; \]
\[ y = "\#PI/2"; \]
\[ z = "(1.23+4.5)*\#E"; \]

In FORTRAN90, we can write as follows;
Table 1. Computation time for basic arithmetic (unit : micro-sec.)

| digits | operation | Maple | Mathematica | MPFR [9] | Pari | proposed library |
|--------|-----------|-------|-------------|----------|------|------------------|
| 100    | mul       | 22    | 1.5         | 0.24     | 0.29 | 0.13             |
|        | div       | 26    | 5.9         | 0.52     | 1.2  | 0.30             |
| 1000   | mul       | 100   | 17          | 9.2      | 15   | 3.4              |
|        | div       | 93    | 50          | 16       | 23   | 8.0              |
| 10000  | mul       | 3400  | 580         | 360      | 1400 | 257              |
|        | div       | 3000  | 1800        | 730      | 1500 | 606              |

digits means decimal digit, mul: multiplication, div: division

Table 2. Computation time of proposed library (unit : micro-sec.)

| digits | operation | 64-bit architecture | 32-bit architecture |
|--------|-----------|---------------------|---------------------|
|        |           | Opteron150 2.4GHz    | Athlon64 3200+ 2.2GHz | Core 2 Duo E6600 2.4GHz |
|        |           | 32-bit architecture | 64-bit architecture |
|        |           | Xeon 3.2GHz         | Pentium-M 1GHz       |
| 100    | add       | 0.028               | 0.031               | 0.084               | 0.11             | 0.25             |
|        | mul       | 0.12                | 0.13                | 0.23                | 0.65             | 1.1              |
|        | div       | 0.27                | 0.30                | 0.39                | 1.5              | 3.4              |
| 1000   | add       | 0.15                | 0.17                | 0.36                | 0.67             | 1.5              |
|        | mul       | 3.1                 | 3.4                 | 4.4                 | 29               | 44               |
|        | div       | 7.2                 | 8.0                 | 12                  | 64               | 126              |
| 10000  | add       | 1.3                 | 1.5                 | 3.4                 | 6.5              | 14               |
|        | mul       | 232                 | 257                 | 342                 | 1420             | 3294             |
|        | div       | 547                 | 606                 | 987                 | 5533             | 10244            |

div : addition, mul: multiplication, div: division

\[ x = '0.1' \]
\[ y = '#PI/2' \]
\[ z = '(1.23+4.5)*#E' \]

Strings on the right hand sides are parsed at run-time by the recursive descent manner [8], and they are calculated and assigned to multiple-precision numbers on the left hand sides with a requested precision.

We show the arithmetic operation speed in Table 1,2 and the computational time of LU decomposition of the Hilbert matrix in Table 3. We know that our proposed environment realizes faster computation than other multiple-precision arithmetic environments.

The proposed software exflib is distributed via the Internet [10] and works in 64-bit mode on AMD64/EM64T, SPARC V9 architectures, and in 32-bit mode on IA-32 architecture. It works with UNIX or Windows operating system, with C++ compilers as GCC, Borland C++, and Visual C++, and FORTRAN90 compilers as g95, pgf95 and so on. It works on supercomputers with parallel computation libraries, and we realize large scale multiple-precision arithmetic computations.
Table 3. CPU time of LU decomposition of Hilbert Matrix (unit : sec.)

| digits, size of matrix | Opteron150 2.4GHz | Athlon64 3200+ 2.2GHz | Core 2 Duo E6600 2.4GHz | Xeon (32-bit) 3.2GHz | Pentium-M 1GHz |
|------------------------|-------------------|------------------------|------------------------|---------------------|-----------------|
| 1000, 400              | 77                | 88                     | 108                    | 708                 | 1005            |
| 2000, 400              | 251               | 284                    | 360                    | 2570                | 3650            |
| 3000, 400              | 527               | 598                    | 761                    | 4910                | 10500           |
| 2000, 800              | 1990              | 2280                   | 2820                   | 20200               | 28600           |
| 3000, 800              | 4140              | 4720                   | 5930                   | 38400               | 82200           |

3. Application to an Inverse Acoustic Scattering

We deal with an inverse scattering problem to determine an unknown sound-soft obstacle in the 2D wave field governed by the Helmholtz equation. Our inverse problem is to determine the shape and the location of the obstacle from the measurement of the far field pattern for one incident plane wave. We follow the decomposition method [12], which reduces the inverse problem to an integral equation of the first kind, and we apply multiple-precision arithmetic to overcome the numerical instability without any stabilization methods as far as we omit observation errors.

Prior to the inverse problem, we explain a direct problem connected with it. Consider an exterior boundary value problem of the Helmholtz equation in \( \mathbb{R}^2 \):

\[
\begin{align*}
\Delta u^s + k^2 u^s &= 0 \quad \text{in } \mathbb{R}^2 \setminus \bar{D}, \\
u^s + u^i &= 0 \quad \text{on } \partial D, \\
\frac{\partial}{\partial |x|} u^s + \sqrt{-1} k u^s &= o \left( \frac{1}{\sqrt{|x|}} \right), \quad |x| \to +\infty,
\end{align*}
\]

where \( D \) is a simply connected domain corresponding to an obstacle and \( k \) is the wave number. We denote the given incidental wave and the scattered wave by \( u^i \) and \( u^s \) respectively. Since the fundamental solution to the 2D Helmholtz equation is \( E(x) = \frac{1}{4} H_0^1(k|x|) \), where \( H_0^1 \) is the Hankel function of order zero and of the first kind, the unique solution to (1)-(3) is written in

\[
u^s(x) = \int_{\partial D} \left\{ u^s(y) \frac{\partial}{\partial n_y} E(x - y) - \frac{\partial u^s}{\partial n_y}(y) E(x - y) \right\} ds_y, \quad x \in \mathbb{R}^2 \setminus \bar{D},
\]

where \( n \) is the unit outward normal vector to \( \partial D \). By the asymptotic expansion of \( E(x) \) in (4), there exists a function \( u_\infty(\hat{x}) \) on \( S^1 \) called the far field pattern such that

\[
u^s(x) = e^{i k |x|} \left\{ u_\infty(\hat{x}) + O \left( \frac{1}{|x|} \right) \right\}, \quad |x| \to +\infty,
\]

and

\[
u_\infty(\hat{x}) = \frac{e^{ik\hat{x}}}{\sqrt{8\pi k}} \int_{\partial D} \left\{ u^s(y) \frac{\partial e^{-ik\hat{x} \cdot y}}{\partial n_y} - \frac{\partial u^s}{\partial n}(y) e^{-ik\hat{x} \cdot y} \right\} ds_y, \quad \hat{x} = \frac{x}{|x|} \in S^1.
\]

Our inverse problem is to determine an unknown obstacle \( D \) by the measurement of the far field pattern \( u_\infty(\hat{x}) \). Colton and Kress [12] proposes the decomposition method to reconstruct the unknown obstacle \( D \) using a priori information that it contains the unit circle \( S^1 \) and that the solution \( u^s \) of (1)-(3) is written by the single layer potential defined on \( S^1 \) as

\[
u^s(x) = \int_{S^1} E(x - \hat{y}) \mu(\hat{y}) \, ds_{\hat{y}},
\]
where $\mu(\hat{y})$ is an unknown density function on $S^1$. From (7), the asymptotic expansion

$$u^s(x) = \frac{e^{i\pi}e^{i|k|x}}{\sqrt{8\pi k|x|}} \left\{ \int_{S^1} e^{-ik\hat{x} \cdot \hat{y}} \mu(\hat{y}) \, ds_\hat{y} + O \left( \frac{1}{|x|} \right) \right\}, \quad |x| \to +\infty,$$

holds and its comparison with (5) leads us an integral equation for $\mu(\hat{y});$

$$\frac{e^{i\pi}}{\sqrt{8\pi k}} \int_{S^1} e^{-ik\hat{x} \cdot \hat{y}} \mu(\hat{y}) \, ds_\hat{y} = u^\infty(\hat{x}), \quad \hat{x} \in S^1. \quad (8)$$

We firstly determine the density function $\mu(\hat{y})$ by the measured far field pattern $u^\infty(\hat{x})$ and give the solution $u^s(x)$ by (7). Once $u^s(x)$ is determined, we reconstruct the unknown boundary $\partial D$ as the contour line of $u^s + u^i = 0$. Our inverse problem is reduced to the equation (8).

We note that the kernel of the integral equation (8) is analytic, and that the equation becomes ill-posed within any class of the Sobolev spaces. In the conventional algorithm, the Tikhonov regularization is required to stabilize numerical processes of (8). In the current research, as far as we do not consider the influence of observation errors, we take a direct approach with a multiple-precision arithmetic; we do not use any stabilization techniques.

We explain our numerical experiments. We firstly solve the direct problem to construct the far field pattern $u^\infty$ by numerical computation of (1)-(3) with the boundary element method [13]. Let $x \in \mathbb{R}^2 \setminus D$ tend to $z \in \partial D$ in (4), we obtain a boundary integral equation

$$\int_{\partial D} E(z - y)q(y) \, ds_y = \frac{1}{2} u^i(z) - \int_{\partial D} \frac{1}{\partial n_y} E(z - y)u^i(y) \, ds_y, \quad z \in \partial D$$

for the unknown Neumann data $q := \partial u^s / \partial n$. The integral equation is uniquely solvable when $k^2$ belongs to the resolvent set $\rho(-\Delta)$. Once obtaining a numerical approximation to $q$, we can obtain the numerical far field pattern $u^\infty,N(x)$ by (6), and we start our inverse problem.

For the case of $k = 10$, $\partial D = \{(2 \cos \theta, \frac{1}{2} \sin \theta) ; 0 \leq \theta < 2\pi\}$, and the incidental plane wave is $u^i(x) = e^{ikx \cdot \vec{d}}$ with a direction $\vec{d} = (1, 0)$, we obtain the numerical far field pattern shown in Fig. 2. For the sake of inverse analysis discussed later, we use both 200 decimal digits arithmetic in numerical computations and the Martensen-Kussmaul rule in each numerical integration [12].

![Figure 2](computed_far_field_pattern_200digit_N=300_k=10.png)

**Figure 2.** Direct problem, computed far field pattern $u^\infty,N; \, N = 300, \, k = 10.$
In inverse problem, we firstly calculate the potential $\mu$ in the integral equation of the first kind (8) with the analytic kernel, of which the numerical process is ill-conditioned. We show a numerical result by the standard double precision in Fig. 3 and we know the numerical computation fails. On the other hand, using 200 decimal digits computation with multiple-precision arithmetic, we obtain a numerical solution $\mu_N$ shown in Fig. 4 for $N = 300$. We use our multiple-precision software exflib and we remark that computational time is about 40 seconds on Opteron280 (2.4GHz). In the numerical computation of the integral equation (8), we adopt the Martensen-Kussmaul integration rule and the collocation method. Finally we find zero contour line of $|u^i + u^s_N|$ where $u^s_N$ is given by (7) with $\mu = \mu_N$. In the current numerical examples, the profile of $|u^i + u^\infty_N|$ is shown in Fig. 5, and the most dark ring coincide well with the exact boundary curve $\partial D = \{(2 \cos \theta, \frac{3}{2} \sin \theta) : 0 \leq \theta < 2\pi\}$. We define numerical error "error$_N$" by the maximum of $|u^i + u^\infty_N|$ on the exact $\partial D$, and we obtain the error decay shown in Fig. 6. From the figure we know that the decomposition method has exponential convergence with respect to discretization parameter $N$. And we know that 100 decimal digits is not enough for $N = 240$. In that case extending the computational precision to 150 or 200 decimal digits, we can evade the influence of rounding errors, and the decomposition method works well.

These simple numerical examples clearly explain the efficiency of applications of multiple-
precision arithmetic to ill-conditioned problems. We discuss details in [14].

4. Concluding Remarks

We introduce our fast multiple-precision arithmetic exflib [10] and propose its use in numerical analysis of ill-posed problems. The numerical results imply that our approach enables precise numerical simulations for numerically unstable problems without any stabilizations.

From the view point of realistic numerical computations, the estimation of the growth of rounding errors is important. Rounding errors depend on the various factors such as the problem, discretization manner, and user program and mathematically a priori qualitative estimation is almost impossible. We propose an a posteriori quantitative approach by multiple-precision interval arithmetic [15]. The discretization errors are also crucial in its numerical computation, and we have proposed the effective use of the spectral method [16] on multiple-precision arithmetic [17, 18].

In the practical cases, we should consider the influence of measurement errors, and it is necessary to combine the discretized regularization method with the multiple-precision arithmetic. We give a remark that the mathematical stability does not always imply the numerical stability. If we need accurate numerical solution by the Tikhonov regularization method, we choose a regularization parameter, for which the discretization of Tikhonov regularized equation is numerically ill-conditioned and standard numerical computation does not give an optimal solutions [19]. To realize the reliable and effective numerical method of inverse problems, we must establish a criterion for discretization parameter, computational precision, and stabilization parameters.

Acknowledgments

The authors would like to express their thanks to Professor Sin Hitotumatu for his fruitful comments in the design of exflib. The research was supported by Grant-in-Aid for Scientific Research (No. 17740057, 16340024, 17654023) from Japan Society for the Promotion of Science.
References

[1] Tikhonov A N and Arsenin A 1977 Solutions of Ill-Posed Problems (New York : Wiley)
[2] IEEE 1985 IEEE standard for binary floating-point arithmetic : ANSI/IEEE std 754-1985, Reprinted in SIGPLAN 22 (1987) pp 9–25
[3] Iso Y, FUJIWARA H, Saito K 2001 Numerical computations for ill-conditioned problems by multiple-precision systems Mathematical Modeling and Numerical Simulation in Continuum Mechanics, LNCSE 19 pp 185–194
[4] Fujiwara H and Iso Y 2002 Some Remarks on the Choice of Regularization Parameter under Multiple-Precision Arithmetic Theoretical and Applied Mechanics Japan 51 pp 387–393
[5] Patterson D A and Hennessy J L 1994 Computer Organization and Design (Morgan Kaufmann)
[6] Smith D M 1991 A Fortran package for floating-point multiple-precision arithmetic Trans. Mathematical Software 17 pp 273–283
[7] Knuth D E 1998 The Art of Computer Programming Vol. 2 3rd ed. (Addison-Wesley)
[8] Aho A V, Sethi R and Ullman J D 1986 Compilers, Principles, Techniques, and Tools (Addison-Wesley)
[9] http://www.medicis.polytechnique.fr/~pphd/mpfr/timings.html
[10] http://www-an.acs.i.kyoto-u.ac.jp/~fujiwara/exflib
[11] Kress R 1989 Linear Integral Equations (Springer-Verlag)
[12] Colton D and Kress R 1998 Inverse Acoustic and Electromagnetic Scattering Theory, 2nd ed. (Springer-Verlag)
[13] Kirsch A and Kress A 1987 An optimization method in inverse acoustic scattering Boundary Elements IX 3 pp 3–18
[14] Fujiwara H and Iso Y Numerical analysis of inverse and ill-posed problems by multiple-precision arithmetic and the regularization method (in preparation)
[15] Fujiwara H Quantitative analysis of ill-conditioned problems with multiple-precision interval arithmetic (in preparation)
[16] Gottlieb D, Hussaini M Y and Orszag S A 1984 Theory and applications of spectral methods Spectral Methods for Partial Differential Equations pp 1–54 (SIAM)
[17] Imai H and Takeuchi T 2002 Some advanced applications of the spectral collocation method GAKUTO Internat. Ser. Math. Sci. Appl. 17 pp 323–335
[18] Fujiwara H 2003 Numerical method for integral equation of the first kind under multiple-precision arithmetic Theoretical and Applied Mechanics Japan 52 pp 192–203
[19] Fujiwara H and Iso Y 2002 Some remarks on the choice of regularization parameter under multiple-precision arithmetic Theoretical and Applied Mechanics Japan 51 pp 387–393