# A Highly Efficient Implementation of Multiple Precision Sparse Matrix-Vector Multiplication and Its Application to Product-type Krylov Subspace Methods

Tomonori Kouya
Shizuoka Institute of Science and Technology
2200-2 Toyosawa, Fukuroi, Shizuoka 437-8555 Japan[*†]

August 6, 2012

**Abstract**

We evaluate the performance of the Krylov subspace method by using highly efficient multiple precision sparse matrix-vector multiplication (SpMV). BNCpack is our multiple precision numerical computation library based on MPFR/GMP, which is one of the most efficient arbitrary precision floating-point arithmetic libraries. However, it does not include functions that can manipulate multiple precision sparse matrices. Therefore, by using benchmark tests, we show that SpMV implemented in these functions can be more efficient. Finally, we also show that product-type Krylov subspace methods such as BiCG and GPBiCG in which we have embedded SpMV, can efficiently solve large-scale linear systems of equations provided in the UF sparse matrix collections in a memory-restricted computing environment.

## 1 Introduction

Large-scale scientific computation is becoming more and more necessary these days in a variety of fields. Consequently, a number of ill-conditioned problems are created, which cannot be solved precisely with standard double precision floating-point arithmetic. To solve such ill-conditioned problems, multiple precision floating-point arithmetic, such as quadruple or octuple precision is needed as a countermeasure.

At present, although high performance numerical software libraries that support multiple precision floating-point arithmetic are rare, they are being developing. These libraries are built upon primitive and well-tuned arithmetic suitable for standard architecture CPUs and GPUs. Numerical solutions with the required precision can be obtained on a commodity PC.

[*]2010 Mathematics Subject Classification: 65F50, 65F10, 65G50

[†]Keywords and phrases: Sparse Matrix, Multiple Precision Floating-point Arithmetic, Product-type Krylov Subspace Method

As a result, increasing more memory is required for multiple precision floating-point numbers, and hence, memory needs to be specifically allocated for basic linear algebra computation. If the matrix being treated is sparse, the amount of memory can be reduced by storing nonzero elements in the matrix.

Some double precision libraries for treating sparse matrices have been provided such as SuiteSparse[8] and Intel Math Kernel (IMK) Library[3], which have direct and iterative methods to solve sparse linear systems of equations. If you want to build a multiple precision sparse matrix library, you can convert such double precision libraries capable of manipulating multiple precision sparse matrices, such as MPACK[4] which are built to convert LAPACK with QD/GD[1] and GNU MPFR/GMP[6]. However, our multiple sparse matrix-vector multiplication is developed as an additional function in BNCpack[2], which is our own double and multiple precision numerical computation library based on GNU MPFR/GMP, the multiple precision floating-point arithmetic library.

In MPFR/GMP, zero multiplication is made faster to save computational time, and we need to estimate how our multiple precision sparse matrix-vector product (SpMV for short) can reduce computational time in comparison with multiple precision dense matrix-vector multiplication (Dense MV) through numerical experiments. The multiple precision SpMV is shown to be very efficient and to reduce the computational time for product-type Krylov subspace methods in limited memory environments.

In this paper, we first introduce the multiple precision SpMV and the performance of MPFR/GMP. Next, we benchmark the SpMV in a standard PC environment and evaluate the performance in comparison with Dense MV. Finally, we apply SpMV to product-type Krylov subspace methods provided in BNCpack and demonstrate its efficiency by solving test problems using some sparse matrices in the UF Sparse Matrix Collection.

## 2 Sparse matrix structure based on BNCpack

In the current BNCpack, all of the multiple precision functions are based on MPFR/GMP (MPFR over the MPN kernel of GMP). The significant features of MPFR/GMP are as follows:

- Arbitrary precision floating-point arithmetic are executable according to the IEEE754-1985 standard which supports NaN (Not-a-Number), ±Inf (Infinity) and four rounding modes (RN, RZ, RP and RM).
- The multiple precision floating-point data type (`mpfr_t`) in MPFR/GMP is defined as a C structure and the mantissa part is stored in a separate memory area.
- Any mixed precision arithmetic is available everywhere. Each of the `mpfr_t` variables have their own precision (in bits), whereby the precision can change anytime and everywhere throughout the running process.
- An original acceleration mechanism is embedded to save computational time. For example, the arithmetic with fixed precision data types such as integer, float and double variables or zero, is very fast.

In particular, the original acceleration mechanism of MPFR/GMP is very efficient for practical use. Figure 1) shows the performance of multiplication with GMP and MPFR/GMP. The variables `x` and `y` have 10000 decimal digits (333220 bits) for the mantissa, half_x and half_y are half digits, and 0 means storing zero. Comparing x*y, half_x*half_y and 0*x, we see that the 0*x was completed fastest.

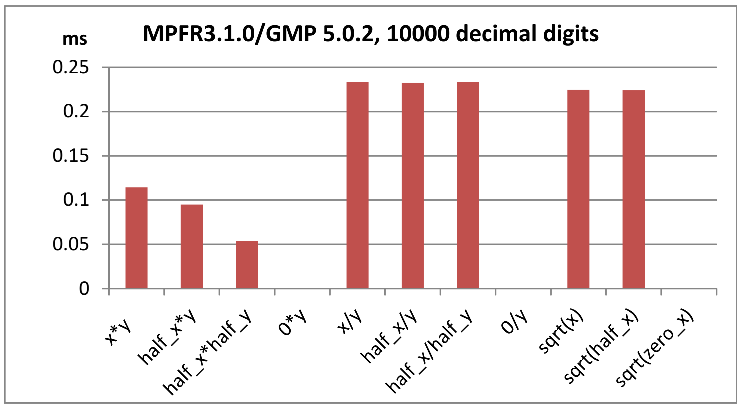


Figure 1: Multiplication performance of variables of set precision: MPFR/GMP

Therefore, the particular efficiency of MV can be obtained using a dense matrix filled with zero elements. The question then is, what is performance of SpMV upon implementing multiple precision sparse matrix data types?

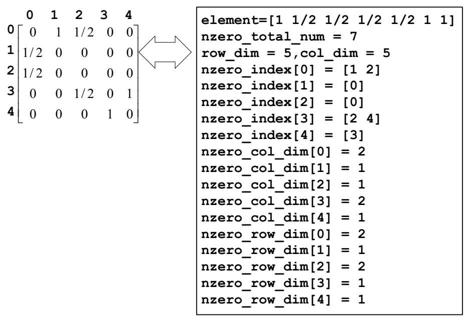
Next, we discuss the multiple precision sparse matrix structure based on MPFR/GMP. The Compressed Sparse Row (CSR) format was selected as it just as the nonzero elements is expressed as a one-dimensional array. The C structure is as follows:

```
typedef struct {
  unsigned long prec;         // Precision in bits
  mpf_t *element;             // Nonzero elements of the sparse matrix
  long int row_dim, col_dim; // Dimension of row and column
  long int **nzero_index;     // Index of non zero elements
  long int *nzero_col_dim;    // Number of nonzero elements per row
  long int *nzero_row_dim;    // Number of nonzero elements per column
  long int nzero_total_num;   // Total number of nonzero elements
} mpfrsmatrix;
```

According to this structure, for example, the $5 \times 5$ random sparse matrix can be as shown in Figure 2.

As the dimension of matrix increases, the memory size can be reduced by using this sparse matrix structure, which can then allow more large-scale matrices to be manipulated, which cannot be stored as dense matrices. As mentioned above, due to the original acceleration mechanism embedded in MPFR/GMP, it is not known to what extent the computational time of sparse matrix-vector multiplication is reduced without benchmark tests.

For more details, the source codes used in this paper can be found in the latest BNCpack[2] library.



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 1/2 & 0 & 0 \\ 1 & 1/2 & 0 & 0 & 0 & 0 \\ 2 & 1/2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1/2 & 0 & 1 \\ 4 & 0 & 0 & 0 & 1 & 0 \end{array} \Longleftrightarrow$$

```
element=[1 1/2 1/2 1/2 1/2 1 1]
nzero_total_num = 7
row_dim = 5,col_dim = 5
nzero_index[0] = [1 2]
nzero_index[1] = [0]
nzero_index[2] = [0]
nzero_index[3] = [2 4]
nzero_index[4] = [3]
nzero_col_dim[0] = 2
nzero_col_dim[1] = 1
nzero_col_dim[2] = 1
nzero_col_dim[3] = 2
nzero_col_dim[4] = 1
nzero_row_dim[0] = 2
nzero_row_dim[1] = 1
nzero_row_dim[2] = 2
nzero_row_dim[3] = 1
nzero_row_dim[4] = 1
```

Figure 2: Sparse matrix structure implemented in BNCpack.

## 3 Performance evaluation of multiple precision SpMV

We evaluate the performance testing described in this paper with the following. Although the environment used has a multi-core CPU, all tests are executed in a single thread.

**H/W** Intel Core i7 920, 8GB RAM

**S/W** CentOS 5.6 x86_64, gcc 4.1.2

**MPFR/GMP** MPFR 3.0.1, GMP 5.0.1

**BNCpack** BNCpack 0.7

**BLAS/SparseBLAS** Intel Math Kernel Library 10.3 Update 1

The sparse matrix for evaluation is the Lotkin matrix which is randomly filled with zeros.

$$A = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1/2 & 1/3 & \cdots & 1/n \\ \vdots & \vdots & & \vdots \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix}$$

Each element has a mantissa of $p$ precision. The final sparse matrix $A^{(s)}$ has $s$% zero elements. We select a real vector **b** given by

$$\mathbf{b} = A^{(s,p)}[1\ 2\ \cdots n]^T$$

and evaluate the real matrix-vector multiplication $A^{(s,p)}\mathbf{b}$. We compare this with $A^{(s)}_{Dense}$, the dense matrix format generated by storing the sparse matrix $A^{(s)}$, by examining the ratio of the computational time for $A^{(s)}_{Dense}\mathbf{b}$, which is defined as the “Speedup Ratio.”

The computational time and the speedup ratio are shown in Figure 3. For reference, the results for the double precision BNCpack and Intel Math Kernel (DGEMV/DCOOGEMV) are also shown.

Double precision SpMV can be effective at 2000-dimesional matrix- vector multiplication. Multiple precision SpMV can be less effective than double precision. The speedup ratio varies between about 1.3 and 3.2. The reason can be interpreted to be the effect of the MPFR/GMP acceleration mechanism, whereby zero multiplication can

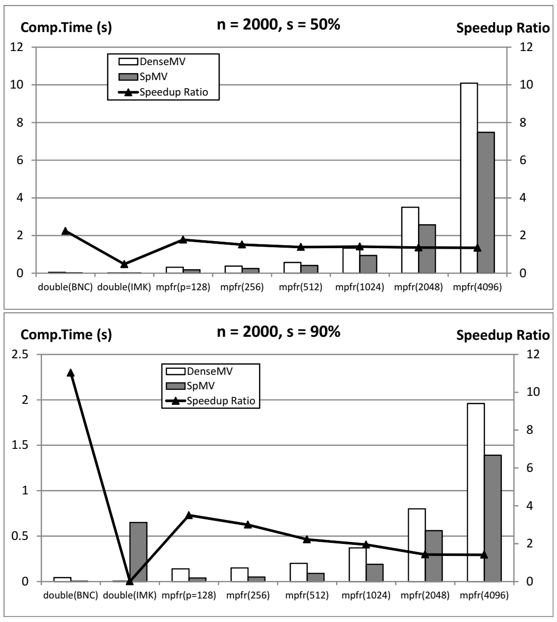


Figure 3: Speedup ratio for SpMV (vs. Dense MV).

be executed very quickly in dense MV. This can be understood from the fact that the higher the precision in SpMV, the lower is the speedup ratio actually gained. Figure 4 explains this mechanism.

The performance evaluation of SpMV shows that the computational time can be reduced if *s* and *p* are sufficiently large and that the speedup of SpMV is effective if the matrix is large and the precision is not very high.

## 4 Application to product-type Krylov subspace methods

The performance evaluation of SpMV described above demonstrates that not only is memory saved but speedup is also achieved. Consequently more efficient product-type Krylov subspace methods[9] with multiple precision SpMV can be implemented. We selected the BiCG, CGS, BiCGSTAB and GPBiCG methods.

Krylov subspace methods and their variations are well-known for their sensitivity to the effect of round-off errors occurring during their iterative processes, and this effect can be reduced by using multiple precision floating-point arithmetic[7]. In this section, we describe the performance evaluation and numerical experiments using multiple precision product-type Krylov subspace methods. We also describe the numerical experiments with the ILU(0) method for left preconditioning.

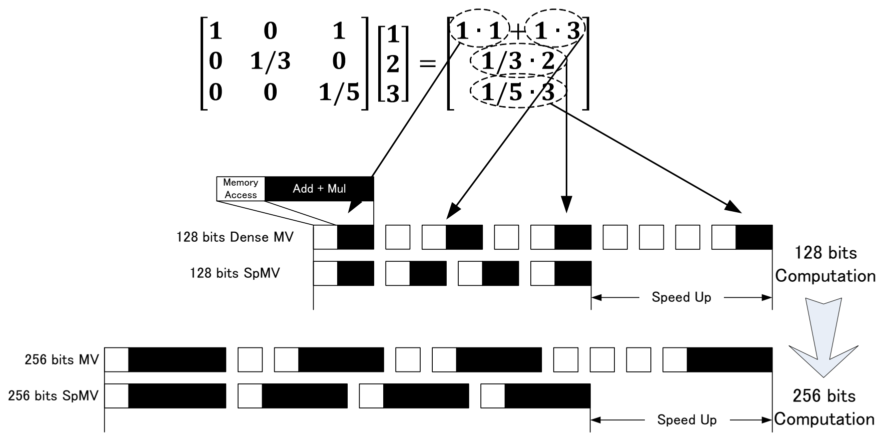

$\begin{bmatrix}1 & 0 & 1\\ 0 & 1/3 & 0\\ 0 & 0 & 1/5\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}1\cdot 1 + 1\cdot 3\\ 1/3\cdot 2\\ 1/5\cdot 3\end{bmatrix}$
Memory Access
Add + Mul
128 bits Dense MV
128 bits SpMV
Speed Up
128 bits Computation
256 bits MV
256 bits SpMV
Speed Up
256 bits Computation

Figure 4: Explanation of speedup using SpMV.

## 4.1 Performance evaluation of multiple precision product-type Krylov subspace methods

From the implemented BiCG, CGS, BiCGSTAB, and GPBiCG, we present the algorithms of BiCG and GPBiCG (Figure 5. The underlined sections indicate the replacement for SpMV.

In the numerical experiments below, all the sparse matrices used are provided in the UF Sparse Matrix Collection[5]. The original MTX files storing the sparse matrices have only double precision elements and hence, these are converted into multiple precision floating-point numbers. The true solutions are commonly $\mathbf{x} = [1\ 2\ \cdots\ n]^T$, and thus, the constant vectors are $\mathbf{b} = A\mathbf{x}$, which are calculated in multiple precision computation. The convergence condition ((*) in Figure 5) is satisfied if

$$\|\mathbf{r}_k\|_2 \leq 10^{-20}\|\mathbf{r}_0\|_2 + 10^{-50}.$$

**DRICAV/cavity04** DRIVCAV/cavity04 ($n = 317$) is selected for small-scale problems. The number of nonzero elements is 7327 ($s = 93.7\%$). Problems of the size of this scale can be completely stored in dense matrix format in RAM, which enables us to find the speedup ratio for the usage of SpMV. The precision of the elements of the matrix is extended from 512 bits (about 154 decimal digits) to 8192 bits (about 2466 decimal digits) and we compare the speedup ratio with Dense MVs.

The structure of cavity04 matrix and the speedup ratio for BiCG and GPBiCG are shown in Figure 6.

The iterative and computational times for the BiCG, CGS, BiCGSTAB, and GPBiCG methods for the cavity04 problem are listed in Table 1.

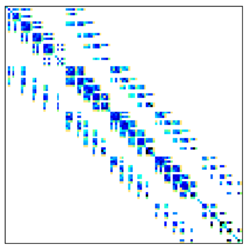
In the case of the cavity04 problem, the history of residual norms (Figure 7) shows that product-type Krylov subspace methods need more than 2048 bits precision to converge. Multiple precision arithmetic is very effective in obtaining more precise numerical solutions.

**Averous/epb3** Averous/epb3 ($n = 84617$) is selected for larger scale problems. The true solution $\mathbf{x}$ and the constant vector $\mathbf{b}$ are the same as for the cavity04 problem.

```
$\mathbf{x}_0$: Initial guess
$\mathbf{r}_0$: Initial residual ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}_0}$: $= \mathbf{r}_0$.
$K$: Matrix for Left preconditioning.
for $i = 1, 2, ...$
    Solve $K\mathbf{w}_{i-1} = \mathbf{r}_{i-1}$ on $\mathbf{w}_{i-1}$.
    Solve $K^T\ \widetilde{\mathbf{w}_{i-1}} = \widetilde{\mathbf{r}_{i-1}}$ on $\widetilde{\mathbf{w}_{i-1}}$.
    $\rho_{i-1} = (\widetilde{\mathbf{w}_{i-1}}, \mathbf{w}_{i-1})$
    if $\rho_{i-1} = 0$ then exits.
    if $i = 1$ then
        $\mathbf{p}_1 = \mathbf{w}_0$
        $\widetilde{\mathbf{p}_1} = \widetilde{\mathbf{w}_0}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $\mathbf{p}_i = \mathbf{w}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1}$
        $\widetilde{\mathbf{p}_i} = \widetilde{\mathbf{w}_i} + \beta_{i-1}\widetilde{\mathbf{p}_{i-1}}$
    end if
    $\underline{\mathbf{z}_i = A\mathbf{p}_i}$
    $\underline{\widetilde{\mathbf{z}_i} = A\widetilde{\mathbf{p}_i}}$
    $\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{p}_i}, \mathbf{z}_i)$
    $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i$
    $\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i\mathbf{z}_i$
    $\widetilde{\mathbf{r}_i} = \widetilde{\mathbf{r}_{i-1}} - \alpha_i\widetilde{\mathbf{z}_i}$
    Convergence check(*).
end for
```

```
$\mathbf{u} = \mathbf{z} = 0$
for $i = 1, 2, ...$
    $\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$
    if $\rho_{i-1} = 0$ then exits.
    if $i = 1$ then
        $\mathbf{p} = \mathbf{r}_0$
        $\underline{\mathbf{q} = A\mathbf{p}}$
        $\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
        $\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$
        $\underline{\mathbf{v} = A\mathbf{t}}$
        $\mathbf{y} = \alpha_i\mathbf{q} - \mathbf{r}_{i-1}$
        $\mu_2 = (\mathbf{v}, \mathbf{t}), \mu_5 = (\mathbf{v}, \mathbf{v})$
        $\zeta = \mu_2/\mu_5$
        $\eta = 0$
    else
        $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\zeta)$
        $\mathbf{w} = \mathbf{v} + \beta_{i-1}\mathbf{q}$
        $\mathbf{p} = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p} - \mathbf{u})$
        $\underline{\mathbf{q} = A\mathbf{p}}$
        $\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
        $\mathbf{s} = \mathbf{t} - \mathbf{r}_{i-1}$
        $\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$
        $\underline{\mathbf{v} = A\mathbf{t}}$
        $\mathbf{y} = \mathbf{s} - \alpha_i(\mathbf{w} - \mathbf{q})$
        $\mu_1 = (\mathbf{y}, \mathbf{y}); \mu_2 = (\mathbf{v}, \mathbf{t})$
        $\mu_3 = (\mathbf{y}, \mathbf{t}); \mu_4 = (\mathbf{v}, \mathbf{y})$
        $\mu_5 = (\mathbf{v}, \mathbf{v});$
        $\tau = \mu_5\mu_1 - \overline{\mu_4}\mu_4$
        $\zeta = (\mu_1\mu_2 - \mu_3\mu_4)/\tau$
        $\eta = (\mu_5\mu_3 - \overline{\mu_4}\mu_2)/\tau$
    end if
    $\mathbf{u} = \zeta\mathbf{q} + \eta(\mathbf{s} + \beta_{i-1}\mathbf{u})$
    $\mathbf{z} = \zeta\mathbf{r}_{i-1} + \zeta\mathbf{z} - \alpha_i\mathbf{u}$
    $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p} + \mathbf{z}$
    Convergence check(*)
    $\mathbf{r}_i = \mathbf{t} - \eta\mathbf{y} - \zeta\mathbf{u}$
    exits if $\zeta = 0$.
end for
```

Figure 5: Algorithms for BiCG (left) and GPBiCG (right). The underlined sections indicate the replacement for SpMV.



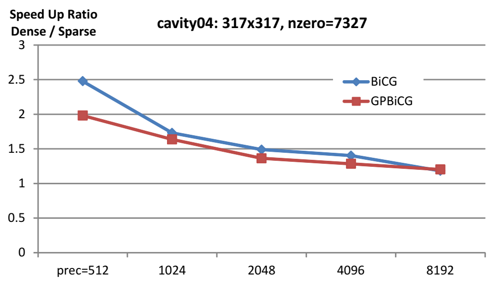


Figure 6: Matrix structure of cavity04 (left) and speedup ratio (right).

Table 1: Iterative and Computational Times (s): cavity04

| Iterative Times (s) | prec=512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|
| BiCG | Not Convergent | NC | 236 (4.83) | 236 (13.74) | 236 (39.91) |
| CGS | NC | NC | NC | 236 (13.74) | 236 (40.29) |
| BiCGSTAB | NC | NC | NC | 260 (16.23) | 236 (42.79) |
| GPBiCG | NC | NC | NC | 24 3(23.99) | 236 (67.48) |

The number of nonzero elements is 463,625 ($s = 99.0035\%$), and hence, if the matrix is stored in dense matrix format, the total amount of memory required is about 320 GB in the case of 128-bit precision and about 70 TB in the case of 8192-bit precision. With our sparse matrix format, about 3.5 MB is required for 128-bit precision and about 456 MB for 8192-bit precision. Hence, numerical experiments with the epb3 matrix can be carried out on standard memory-restricted PCs.

The iterative and computational times are listed in Table 2. "NC" means that the process cannot converge properly.

In the epb3 problem, the effectiveness of multiple precision arithmetic is proven because higher precision can reduce the iterative times of every product-type Krylov subspace methods.

## 4.2 Performance evaluation of the BiCGSTAB method with ILU(0) for left preconditioning

For the double precision Krylov subspace method and its variations, we normally set up the matrix $K$ as a precondition for each problems in order to speed up the iteration. In this case, we use the BiCGSTAB method with incomplete LU decomposition (ILU(0))

Table 2: Iterative and computational (in parenthesis) times for the epb3 problem.

| | 512bits | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|
| BiCG | NC | NC | 8351(4335 s) | 5251(7824 s) | 3567(15551 s) |
| GPBiCG | NC | NC | NC | 7039(12473 s) | 4449(24179 s) |

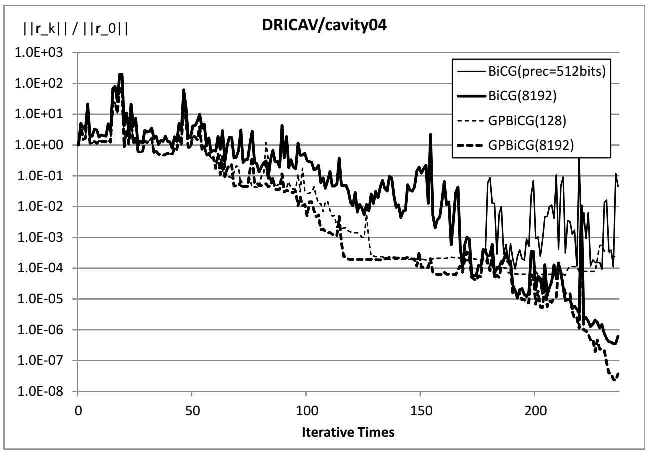


Figure 7: History of $||\mathbf{r}_k||_2/||\mathbf{r}_0||_2$ for the cavity04 problem

as the left precondition, and examine the numerical properties and efficiency of ILU(0).

For the cavity04 problem, the results of BiCGSTAB with ILU(0) are summarized in Table 3.

Table 3: BiCGSTAB method with ILU(0) for left preconditioning: cavity04

| | | prec=512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|
| BiCGSTAB | Iter. Times | NC | NC | NC | 260 | 236 |
| | Time (s) | | | | <u>16.23</u> | 42.79 |
| BiCGSTAB +ILU(0) | Iter. Times | NC | NC | 274 | 238 | 234 |
| | Time (s) | | | <u>45.94</u> | 58.75 | 109.29 |

In this case, 4096-bit precision computation is needed for convergence for the simple BiCGSTAB method. Although the application of ILU(0) can result in convergence of the BiCGSTAB process, the simple 4096-bit BiCGSTAB method is more efficient because the computational time with ILU(0) is long. The results of the numerical experiments show that preconditioning in multiple precision computation is not efficient due to the effect of the matrix structure and other such factors, if it performs better in double precision computation.

## 5 Conclusion

Although multiple precision computation is needed in various types of scientific computation, software libraries applicable to large-scale problems are very rare. Our multiple precision SpMV, based on BNCpack and MPFR/GMP and the product-type Krylov subspace methods, is particularly useful. The numerical in this paper reveals that multiple precision SpMV can save memory space in RAM and can also reduce computational time.

In a future work, we will apply these implemented Krylov subspace methods to large-scale scientific computation in particular, that which includes ordinary and partial differential equations. We will also attempt to further validate BNCpack with SpMV and Krylov subspace methods by applying it to actual problems.

## References

[1] D.H. Bailey. QD. http://crd.lbl.gov/~dhbailey/mpdist/.

[2] Tomonori Kouya. BNCpack. http://na-inet.jp/na/bnc/.

[3] Intel Math Kernel Library. http://www.intel.com/software/products/mkl/.

[4] MPACK. Multiple precision arithmetic BLAS and LAPACK. http://mplapack.sourceforge.net/.

[5] The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/.

[6] MPFR Project. The MPFR library. http://www.mpfr.org/.

[7] Tsubasa Saito, Emiko Ishiwata, and Hidehiko Hasegawa. Analysis of the GCR method with mixed precision arithmetic using QuPAT. *Journal of Computational Science*, Vol. 3, No. 3, pp. 87–91, 2012.

[8] SuiteSparse. http://www.cise.ufl.edu/research/sparse/SuiteSparse/.

[9] Shao-Liang Zhang. A class of product-type krylov-subspace methods for solving nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, Vol. 149, No. 1, pp. 297 – 305, 2002.