Utilizing the quadruple-precision floating-point arithmetic operation for the Krylov Subspace Methods

Hidehiko Hasegawa *

Abract. Some large linear systems are difficult to solve by the Krylov subspace methods. The reason is that they include mathematically not good conditions for such iterative methods. If more accurate calculation is achieved on computers then such problems can be solved easily. However, in general, multiple precision arithmetic operations to achieve more accurate calculation are very costly on many computing environments, and more memory space and much computation time are required. We show the quadruple-precision floating-point arithmetic operation is cost-effective for some class of problems on new high-performance computers. This method will be able to get good performance without any difficulties to parallelize and vectorize some preconditioners.

1 Introduction

The Krylov subspace methods, include Conjugate Gradient (CG) method[6], BiConjugate Gradient (BiCG) method[3], and other methods[2], are widely used for solving the large linear systems. These methods are consist of a combination of matrix-vector operations: $A\boldsymbol{x}$, $A^{\mathrm{T}}\boldsymbol{x}$, $K^{-1}\boldsymbol{r}$, $K^{-\mathrm{T}}\boldsymbol{r}$, an inner product: $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$, and other vector operations[1] where A is the coefficient matrix and K is a preconditioner. As one of effective preconditioners K, an incomplete LU factorization of A [7] [8] is used, however it includes a serial processing part which is not suitable for the parallel and vector processing[1]. For many class of problems, the Krylov subspace methods show good convergence, however for some class of problems they may stagnate or not converge[1]. The following facts about the Krylov subspace methods for linear systems are known:

* The convergence is improved by more accurate computation[4].
* As a mathematically good preconditioner, such as an incomplete factorization, includes a serial processing part, it is difficult to get good performance on vector and parallel computers[1].

In general, multiple precision floating-point arithmetic operations are believed to be costly. However if the quadruple-precision floating-point arithmetic operation, one of the multiple precision arithmetic operations, has reasonable computation cost and good convergence, we have a possibility to get a cost-effective solver based on the Krylov subspace methods. We think about followings:

*School of Library, Information and Media Studies, University of Tsukuba, Tsukuba 305-8550, Japan (hasegawa@slis.tsukuba.ac.jp)

* The quadruple-precision floating-point arithmetic operation should have better convergence than the double-precision floating-point arithmetic operation with an incomplete LU factorization (ILU) preconditioner, however the computation cost may be expensive.

* If there is no preconditioner, it is easy to parallelize and vectorize the Krylov subspace methods.

In this paper, we examine the convergence and the computation time of the double-precision floating-point arithmetic operation with the ILU(0) preconditioner, and the quadruple-precision floating-point arithmetic operation without any preconditioners on some computing environments.

2 Numerical Comparison

The purpose of this paper is to compare the quadruple-precision floating-point arithmetic operation and the double-precision floating-point arithmetic operation. However the convergence property and its performance strongly depend on the problem to be solved. Mathematical property of the problem affects to its convergence, and the structure of the problem such as a sparsity affects to its performance. We solve the system by BiCG method [1] which includes all of basic matrix and vector operations of the Krylov subspace methods, $A\boldsymbol{x}$, $A^{\mathrm{T}}\boldsymbol{x}$, $K^{-1}\boldsymbol{r}$, $K^{-\mathrm{T}}\boldsymbol{r}$, $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$, and etc.

2.1 Model Problem

We use a Toeplitz matrix A of order $N = 200$ with a parameter γ in this paper. It is known that the system is difficult to solve by the Krylov subspace methods when the parameter is large[5]. The important thing is that a difference occurs whether or not between at the quadruple-precision and at the double-precision floating-point arithmetic operations.

$$A := \begin{bmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ \gamma & 0 & 2 & 1 & \\ & \gamma & 0 & 2 & \ddots \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

- Conditions
  - Parameter: $\gamma = 1.3, 1.7, 2.1$ and 2.5.
  - Right-hand side vector: $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$, and starting vector: $\boldsymbol{x}_0 = \boldsymbol{0}$.
  - The stopping criterion: $\varepsilon = 10^{-12}$, and the maximum iteration: 2000 steps.

2.2 Convergence

We examine the convergence under the following conditions (1–3) on a workstation Sun Enterprize 3000.

1. Double-precision floating-point arithmetic operation without preconditioner.
2. Double-precision floating-point arithmetic operation with ILU(0) preconditioner.
3. Quadruple-precision floating-point arithmetic operation without preconditioner (auto-quadruple option is used without any tunings).

Table 1: The iteration counts, computation time and true residual norm on a workstation Sun Enterprize 3000.

		$\gamma = 1.3$	$\gamma = 1.7$	$\gamma = 2.1$	$\gamma = 2.5$
1. Double	Iter.	188	No	No	No
	Time	1.7×10^{-1}			
	True residual	3.1×10^{-13}			
2. Double + ILU(0)	Iter.	40	75	148	179
	Time	6.8×10^{-2}	8.5×10^{-2}	1.1×10^{-2}	1.4×10^{-2}
	True residual	1.7×10^{-13}	5.6×10^{-13}	8.3×10^{-10}	5.2×10^{-2}
3. Quadruple	Iter.	122	152	255	446
	Time	2.0×10^{2}	5.3×10^{2}	5.5×10^{3}	1.9×10^{5}
	True residual	6.6×10^{-13}	6.4×10^{-13}	3.6×10^{-13}	5.4×10^{-13}

This comparison is shown as a table 1. We have got following results:

* The residual norm doesn't converge on the double-precision floating-point arithmetic operation without preconditioner for the parameters $\gamma = 1.7, 2.1$ and 2.5.
* We can't get the approximate solution on the double-precision floating-point arithmetic operation with the ILU(0) preconditioner for the parameters $\gamma = 2.1$ and 2.5.
* We can get the solution only when the quadruple-precision floating-point arithmetic operation is used.
* A lot of computation time is required for the quadruple-precision floating-point arithmetic operation on Sun Enterprize 3000.

2.3 Computation time

The computation time per iteration depends on the hardware, and weights of each basic operation depend on the machine type. For example, as a preconditioner ILU(0) is not vectorized on the vector computer, its computation time is longer than other basic operations,

and its weight is much higher. We investigate the computation time under the following conditions (1–2) on the five computing environments. Here, the order of Toeplitz matrix is 10000. In below tables, "ratio" means the quotient of the computation time per iteration at the quadruple-precision floating-point arithmetic operation without preconditioner by it at the double-precision floating-point arithmetic operation with the ILU(0) preconditioner. MATVEC, MATVECT, PSOLVE, and PSOLVET mean $A\boldsymbol{x}$, $A^{\mathrm{T}}\boldsymbol{x}$, $K^{-1}\boldsymbol{r}$, and $K^{-\mathrm{T}}\boldsymbol{r}$ respectively. DAXPY, DDOT and DNORM2 mean $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ and other main vector operations.

1. Double-precision floating-point arithmetic operation with ILU(0) preconditioner.
2. Quadruple-precision floating-point arithmetic operation without preconditioner(Auto-quadruple option is used without any tunings).

Table 2: Results on a workstation: Sun Enterprize 3000.

	Double + ILU(0)	Quadruple	ratio
MATVEC	4.69×10^{-3}sec. (17.5%)	0.149sec. (24.7%)	31.7
MATVECT	4.89×10^{-3}sec. (18.2%)	0.159sec. (26.4%)	32.5
DAXPY, DDOT DNORM2	7.47×10^{-3}sec. (27.8%)	0.290sec. (48.1%)	38.8
PSOLVE	4.87×10^{-3}sec. (18.1%)	—	—
PSOLVET	4.92×10^{-3}sec. (18.3%)	—	—
Total	2.68×10^{-2}sec.	0.602sec.	22.4

Table 3: Results on a vector computer: Fujitsu VPP800.

	Double + ILU(0)	Quadruple	ratio
MATVEC	3.92×10^{-5}sec. (1.06%)	1.20×10^{-3}sec. (9.52%)	30.6
MATVECT	3.93×10^{-5}sec. (1.07%)	1.20×10^{-3}sec. (9.52%)	30.5
DAXPY, DDOT DNORM2	1.21×10^{-3}sec. (32.9%)	1.01×10^{-2}sec. (80.1%)	8.3
PSOLVE	1.35×10^{-3}sec. (36.7%)	—	—
PSOLVET	1.03×10^{-3}sec. (28.0%)	—	—
Total	3.67×10^{-3}sec.	1.26×10^{-2}sec.	3.4

Table 4: Results on a SMP: Hitachi SR8000.

	Double + ILU(0)	Quadruple	ratio
MATVEC	1.01×10^{-4}sec. (3.5%)	6.12×10^{-4}sec. (11.3%)	6.0
MATVECT	9.96×10^{-5}sec. (3.4%)	6.01×10^{-4}sec. (11.1%)	6.0
DAXPY, DDOT DNORM2	7.81×10^{-4}sec. (27.2%)	4.19×10^{-3}sec. (77.5%)	5.3
PSOLVE	7.38×10^{-4}sec. (25.7%)	—	—
PSOLVET	1.15×10^{-3}sec. (40.9%)	—	—
Total	2.87×10^{-3}sec.	5.40×10^{-3}sec.	1.8

Table 5: Results on a mainframe: Hitachi MP5000.

	Double + ILU(0)	Quadruple	ratio
MATVEC	1.35×10^{-3}sec. (16.9%)	4.68×10^{-3}sec. (27.3%)	3.4
MATVECT	1.34×10^{-3}sec. (16.8%)	4.72×10^{-3}sec. (27.6%)	3.5
DAXPY, DDOT DNORM2	2.44×10^{-3}sec. (30.6%)	7.19×10^{-3}sec. (42.0%)	2.9
PSOLVE	1.46×10^{-3}sec. (18.3%)	—	—
PSOLVET	1.35×10^{-3}sec. (16.9%)	—	—
Total	7.96×10^{-3}sec.	1.71×10^{-2}sec.	2.1

Next, we examine the speed-up of the quadruple-precision floating-point arithmetic operation without a ILU(0) preconditioner. We display the results on a vector parallel computer Fujitsu VPP700 without using its vector facility as a parallel computer.

Table 6: Speed-up on the quadruple-precision: Fujitsu VPP700 ($N = 10000$).

	1PE	2PE	4PE	8PE
MATVEC	317.4 sec. (1.0)	158.4sec. (2.0)	79.3sec. (4.0)	39.8sec. (7.9)
MATVECT	268.6 sec. (1.0)	136.1sec. (1.9)	68.3sec. (3.9)	33.9sec. (7.9)
DAXPY, DDOT DNORM2	839.4 sec. (1.0)	416.9sec. (2.0)	211.4sec. (3.9)	109.1sec. (7.6)
Total	1425.4 sec. (1.0)	711.4 sec. (2.0)	359 sec. (3.9)	182.8 sec. (7.8)

From table 2–6 we have got following results:

* On the ordinary workstation Sun Enterprize 3000, the quadruple-precision floating-point arithmetic operation is almost thirty times slower than the double-precision floating-point operation, however on the classic mainframe Hitachi MP5000 its ratio is four.

* On the vector computer Fujitsu VPP800, as its vector processing facility works well for the double-precision floating-point arithmetic operation, the ratio of Matrix-vector operation is thirty, and the ratio of other vector operations is eight, but the ILU(0) preconditioning is very costly operation on this machine.

* On the SMP Hitachi SR8000, the quadruple-precision floating-point operations for all type of basic operations is only six time slower than the double-precision floating-point arithmetic operation, and the ILU(0) preconditioning is also costly operation.

* The ratio per iteration between the double-precision floating-point arithmetic operation with the ILU(0) preconditioner and the quadruple-precision floating-point arithmetic operation without any preconditioners is about twice on the SMP Hitachi SR8000 and the mainframe Hitachi MP5800, and is about three times on a new vector computer Fujitsu VPP800.

* The speed-up of the quadruple-precision floating-point arithmetic operation without any preconditioners on Fujitsu VPP700 is ideal.

On the Hitachi MP5000 and SR8000, if iteration count of the quadruple-precision floating-point arithmetic operation without any preconditioners is half of the double-precision floating-point arithmetic operation with the ILU(0) preconditioner, then the total computation time

becomes to be same. It is also possible to tune for both the quadruple-precision and the double-precision floating-point arithmetic operation. We can state almost same conclusion for the result on the Fujitsu VPP800. On the simple parallel (without using vector facility) computer Fujitsu VPP700, the quadruple-precision floating-point arithmetic operation without any preconditioners can be forced an ideal speed-up. This is the greatest and the simplest result, because the parallelization of a mathematically powerful preconditioner is very difficult problem but more accurate calculation may work well instead of the preconditioner.

In addition, for the quadruple-precision floating-point arithmetic operation, the size of memory is twice, and the computation time may be more than seven times. This fact fits for the RISC based computers, especially for the parallel computers, because of a small size of data to be transferred per computation. These results of the quadruple-precision floating-point arithmetic operation is measured by only changing its Fortran compiler's option without any tunings, it is possible to reduce the computation time on the quadruple-precision floating-point arithmetic operation by the following improvement:

* To use the quadruple-precision floating-point arithmetic operation together with a mild (not so robust) preconditioner, which means parallelizable and vectorizable.
* To tune the code.
* There is a small weak point that the required memory of the quadruple-precision floating-point arithmetic operation is twice of that of the double-precision floating-point arithmetic operation. However many high-performance computers have enough memory.

3 Conclusion

The quadruple-precision floating-point arithmetic operation is believed to be costly. But it is not true when some large linear systems of equations are solved by the Krylov subspace methods. On the new high-performance computers, the quadruple-precision floating-point arithmetic operation may become to be cost-effective because its convergence is accelerated by more accurate computation, and it is parallelized scalably without any difficulties. Although, in this paper, we could not show that the total computation time at the quadruple-precision floating-point arithmetic operation is really faster than that of at the double-precision floating-point arithmetic operation with the ILU(0) preconditioner for solving a large linear system by the Krylov subspace methods. Also the problem tested in this paper is not a practical problem.

We found that we can get the good convergence on the quadruple-precision floating-point arithmetic operation, and the quadruple-precision floating-point arithmetic operation can bring better results than the double-precision floating-point arithmetic operation for the convergence. Also the computation time on the quadruple-precision floating-point arithmetic operation is about twice on the double-precision floating-point arithmetic operation when the vector facility or multiple CPUs are used. On the other hand, the preconditioner ILU(0) is not useful for reducing the computation time on vector and parallel computers because

the preconditioning includes a serial processing part. We expect that the computation time would be shorter when we employ the mild preconditioner, which can be vectorized and parallelized.

We propose to utilize the quadruple-precision floating-point arithmetic operation with some mild preconditioner to get good performance for some class of problems, they are difficult to be solved by the double-precision floating-point arithmetic operation with the ILU(0) preconditioner, on new high-performance computing environments.

Finally we believe that we should utilize the quadruple-precision floating-point arithmetic operation on new high-performance computing environments, and its high-performance should be used for the quality of computation.

References

[1] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.

[2] Bruaset, A. M., *A Survey of Preconditioned Iterative Methods*, Frontiers in Applied Mathematics 17, Longman Scientific and Technical, London, 1995.

[3] Fletcher, R., Conjugate gradient methods for indefinite systems, in *Numerical Analysis Dundee 1975,* ed. by Watson, G., Lecture Notes in Mathematics, **506**(1976), Springer-Verlag, pp.73-89.

[4] Greenbaum, A., *Iterative Methods for solving Linear Systems*, SIAM, Philadelphia, 1997.

[5] Gutknecht, M. H., Variants of BiCGSTAB for Matrix with Complex Spectrum, *SIAM J. Sci. Comput.*, **14** (1993), 1020-1033.

[6] Hestenes, M. R. and Stiefel, E., Methods of Conjugate Gradients for Solving Linear Systems, *J. Res. Nat. Bur. Standards*, **49** (1952), 409-435.

[7] Meijerink, J. A. and van der Vorst, H. A., An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-matrix, *Mathematics of Computation*, **31** (1977), 148-162.

[8] Saad, Y., *Iterative Methods for Sparse Linear Systems*, PWS, Boston, 1996.