Linear Algebra Operations using Quadruple-precision Arithmetic on GPU

Daichi Mukunoki *poster presenter
Research Fellow of the Japan Society for the Promotion of Science
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
mukunoki@hpcs.cs.tsukuba.ac.jp

Daisuke Takahashi
Faculty of Engineering, Information and Systems, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
daisuke@cs.tsukuba.ac.jp

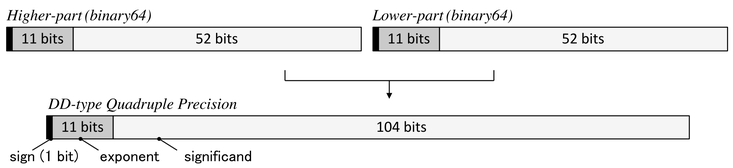
Acknowledgment: A part of this work was supported by JST, CREST and JSPS Grant-in-Aid for JSPS Fellows No. 251290.

Overview

- Floating-point operations have rounding errors and these errors may become a critical issue for some applications. With the advances realized in computational science, there is a need for more accurate computation, and the accumulation of numerical errors may lead to even more serious problems in the future. Therefore, we need to improve the accuracy and precision in floating-point operations on linear algebraic operations
- We implemented some linear algebraic kernels and sparse iterative solvers using quadruple-precision floating-point arithmetic and evaluated the performance on a Tesla K20c GPU
- Since the GPU has relatively high floating-point performance compared to the memory bandwidth, the computation time of quadruple-precision arithmetic is small compared to the memory access time. As a result, the execution times of the quadruple-precision operations are close to 1/2 of that on double-precision
- In such cases, the quadruple-precision operations implemented in software achieve sufficient performance without hardware support of the quadruple-precision arithmetic
- For sparse iterative methods, we found cases where the methods using quadruple-precision arithmetic can reach a solution faster than those only using double-precision arithmetic

Quadruple-precision Floating-point Arithmetic using Double-double Arithmetic on GPUs

- The quadruple-precision floating-point format was standardized in IEEE 754-2008 as binary128 and some compilers supported the format and operations in software for the x86, however, the support is not available on GPUs
- Double-double (DD) arithmetic [1][2] has been proposed and is often used to compute quadruple-precision floating-point arithmetic in software
- The DD arithmetic represents a quadruple-precision floating-point number a using a pair of two double precision floating-point numbers, a_{hi} and a_{lo}: $a=a_{hi}+a_{lo}$ ($|a_{lo}| \leqq 0.5\mathrm{ulp}(a_{hi})$).
- The total significand precision of a DD value is 52 + 52 = 104 bits (106 bits including the implicit bits)



QuadAdd ($c=a+b$)

$$\begin{array}{rr} & a_{hi} \quad a_{lo} \\ + & b_{hi} \quad b_{lo} \\ \hline & a_{hi}+b_{hi} \quad a_{lo}+b_{lo} \\ + & e(a_{hi}+b_{hi}) \\ \hline & c_{hi} \quad c_{lo} \end{array}$$

Normalization: $|c_{lo}| \leqq 0.5\ \mathrm{ulp}(c_{hi})$

$$\begin{array}{rr} & c_{hi} \quad c_{lo} \end{array}$$

QuadMul ($c=a\times b$)

$$\begin{array}{rr} & a_{hi} \quad a_{lo} \\ \times & b_{hi} \quad b_{lo} \\ \hline & a_{hi}b_{hi} \quad a_{hi}b_{lo} \\ & \quad\ b_{hi}a_{lo} \\ + & e(a_{hi}b_{hi}) \quad \text{Not used: } \cancel{a_{lo}b_{lo}} \\ \hline & c_{hi} \quad c_{lo} \end{array}$$

Normalization: $|c_{lo}| \leqq 0.5\ \mathrm{ulp}(c_{hi})$

$$\begin{array}{rr} & c_{hi} \quad c_{lo} \end{array}$$

$(a = a_{hi}+a_{lo},\ b = b_{hi}+b_{lo},\ c = c_{hi}+c_{lo})$

- On the DD arithmetic, quadruple-precision floating-point arithmetic operations are performed using only double-precision floating-point arithmetic operations
- For example, the multiply-add operation ($a*b+c$) requires 20 double-precision floating-point instructions: it requires 20 times as many cycles as the double-precision arithmetic on GPUs
- On the Tesla K20c GPU, the theoretical peak performance is:
  - 706[MHz] × 13[SMX] × 64[Core] × (2[(Double)Flop]/1[cycle]) ≈ 1175[G(Double)Flops]
  - 706[MHz] × 13[SMX] × 64[Core] × (2[(Quad)Flop]/20[cycle]) ≈ 58.7[G(Quad)Flops]

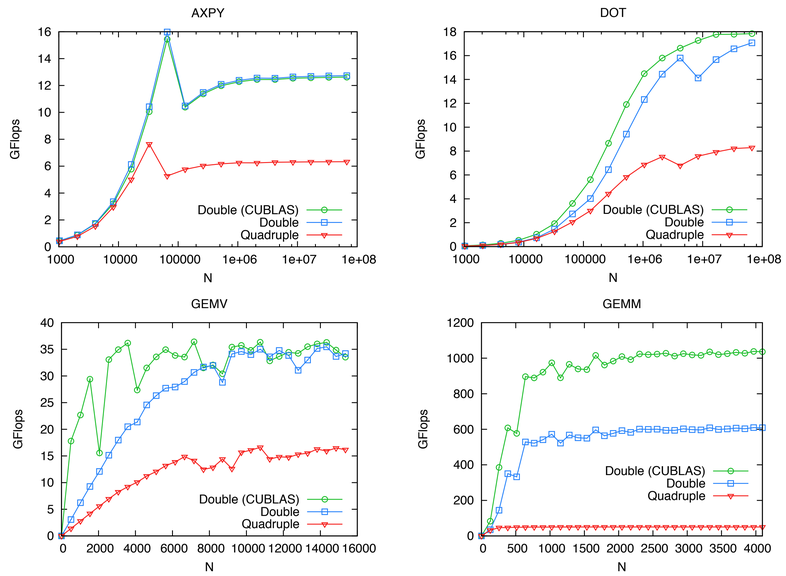
  The theoretical peak performance of the DD arithmetic is 20 times more than that of the double-precision arithmetic on the multiply-add operation

Evaluation Environment:

- GPU: NVIDIA Tesla K20c (5 GB, GDDR5, ECC-enabled)
- CPU: Intel Xeon E5-2609 (2.40 GHz, 4-cores, 1 socket)
- RAM: 16 GB (DDR3)
- OS: CentOS release 6.4
- CUDA: Version 5.5
- Compiler: nvcc 5.5 & gcc 4.4.7 with –O3

BLAS Subroutines (Dense Operation)

- We implemented Basic Linear Algebra Subprograms (BLAS) subroutines: AXPY (y=αx+y), DOT (r=<x, y>), GEMV (y=αAx+βy) and GEMM (C=αAB+βC) in both double- and quadruple-precision [3]
- The performance of quadruple-precision AXPY, DOT and GEMV is limited by the bandwidth of the global memory, and the computation time of the DD arithmetic is hidden by the memory access time: the performance is close to 1/2 of that of the double-precision subroutines
- The performance of quadruple-precision GEMM is computationally-intensive: it is close to 1/20 of that on double-precision (the theoretical peak performance of the quadruple-precision GEMM is 1/20 of that of double-precision one)



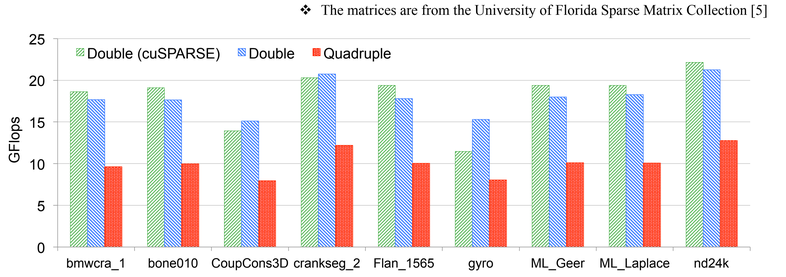
- For quadruple-precision, "Flops" means "*quadruple-precision* floating-point operations per second"
- Input: $N\times N$ matrices with col-major, length N vector (all input data and scalar values of α and β are composed of uniform random numbers)
- Performance does not include the time spent transferring data between CPU and GPU

SpMV (Sparse Matrix-Vector Multiplication)

- SpMV is one of the most important operations in scientific and engineering computing
- We implemented double- and quadruple-precision SpMV routines which perform $y=\alpha Ax+\beta y$ for the Compressed Row Storage (CRS) on Kepler architecture GPUs
- We optimized the routines using some of the new features of the Kepler architecture such as shuffle instructions [4]

Matrix	Rows	Nonzeros	Structure	Application
bmwcra_1	148,770	10,641,602	sym	structural problem
bone010	986,703	47,851,783	sym	model reduction problem
CoupCons3D	416,800	17,277,420	asym	structural problem
crankseg_2	63,838	14,148,858	sym	structural problem
Flan_1565	1,564,794	114,165,372	sym	structural problem
gyro	17,361	1,021,159	sym	model reduction problem
ML_Geer	1,504,002	110,686,677	asym	structural problem
ML_Laplace	377,002	27,582,698	asym	structural problem
nd24k	72,000	28,715,634	sym	2D/3D problem

- The matrices are from the University of Florida Sparse Matrix Collection [5]



- For quadruple-precision, "Flops" means "*quadruple-precision* floating-point operations per second"
- Performance does not include the time spent transferring data between CPU and GPU

CG and BiCGStab Methods (Sparse Iterative Solver)

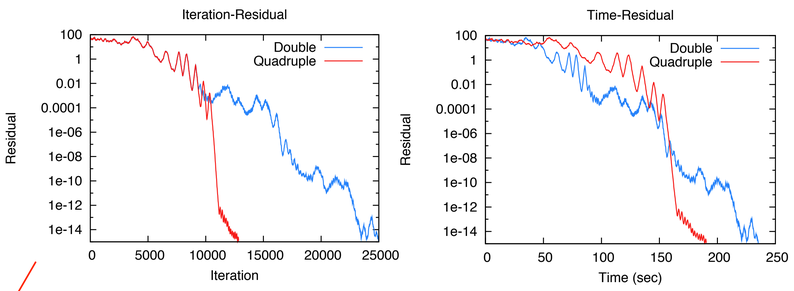
- The convergence of the Krylov subspace methods, such as Conjugate Gradient (CG) and Bi-Conjugate Gradient Stabilized (BiCGStab), iterative methods which are often used to solve large sparse linear systems $Ax=b$, is significantly affected by rounding errors and there are cases where reducing rounding errors with quadruple-precision arithmetic causes the algorithm to converge more quickly when compared to double-precision arithmetic [6]
- We implemented un-preconditioned CG and BiCGStab methods using both double and quadruple-precision arithmetic and compared the performances [7]
- SpMV is generally the most time-consuming operation in the methods
- For the quadruple-precision version, on $Ax=b$, where the input, the coefficient matrix A and the vector b are given in the *double-precision* format, and the vector x and all other floating-point data are stored in the quadruple-precision format.
- Quadruple-precision arithmetic is used instead of double-precision arithmetic everywhere except for checking convergence.

CG method

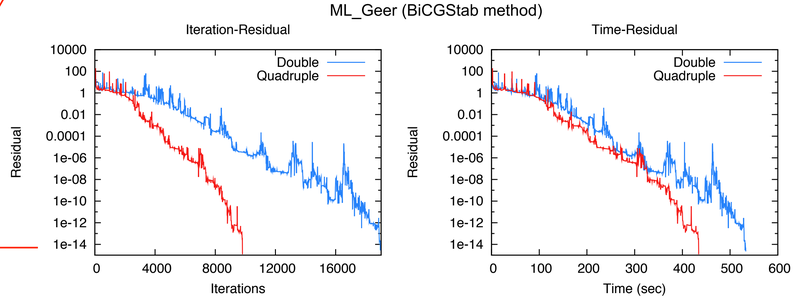
$r_0 = b - Ax_0$
for : $k = 1, 2, \ldots$ **do**
  $\rho_{k-1} = \langle r_{k-1}, r_{k-1} \rangle$
  if $k=1$ **then**
    $p_1 = r_0$
  else
    $\beta_{k-1} = \rho_{k-1}/\rho_{k-2}$
    $p_k = r_{k-1} + \beta_{k-1} p_{k-1}$
  end if
  $q_k = Ap_k$
  $\alpha_k = \rho_{k-1}/\langle p_k, q_k \rangle$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k q_k$
  if $||r_k||_2/||r_0||_2 \leq \varepsilon$ **break**
end for

- We found cases where the quadruple-precision version can reach a solution faster than the double-precision version even when the double-precision version converged, *although such cases were rare*
- We expect the use of quadruple-precision arithmetic, along with the use of preconditioning may be an effective method for accelerating the computation of Krylov subspace methods on GPUs

bone010 (CG method)



ML_Geer (BiCGStab method)



- $b=(1, 1, \ldots, 1)^T$ and the initial vector of x is $x_0=0$
- The residual (vertical axis) represents $||r_k||_2/||r_0||_2$
- Compressed Row Storage (CRS) format is used for storing sparse matrices
- The implementations of the CG method does not utilize the symmetric properties of matrices for storing symmetric matrices

References:

1. T. Dekker: A Floating-Point Technique for Extending the Available Precision, Numerische Mathematik, Vol. 18, pp. 224–242 (1971).
2. D. Bailey. QD (C++ / Fortran-90 double–double and quad-double package). http://crd.lbl.gov/~dhbailey/mpdist/.
3. D. Mukunoki and D. Takahashi: Implementation and Evaluation of Triple Precision BLAS Subroutines on GPUs, Proc. 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW 2012), The 13th Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-12), pp. 1378-1386, May 2012.
4. D. Mukunoki and D. Takahashi: Optimization of Sparse Matrix-vector Multiplication for CRS Format on NVIDIA Kepler Architecture GPUs, Proc. 13th International Conference on Computational Science and Its Applications (ICCSA 2013), Lecture Notes in Computer Science, Vol. 7975, pp. 211-223, Jun 2013.
5. T. Davis and Y. Hu: The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/.
6. H. Hasegawa: Utilizing the quadruple-precision floating-point arithmetic operation for the Krylov Subspace Methods, Proc. 8th SIAM Conference on Applied Linear Algebra (LA03) (2003).
7. D. Mukunoki and D. Takahashi: Using Quadruple Precision Arithmetic to Accelerate Krylov Subspace Methods on GPUs, Proc. 10th International Conference on Parallel Processing and Applied Mathematics (PPAM'2013) (to appear).