Mathematical Problems of Computer Science 45, 44--52, 2016.

Performances of Methods for Solving a Linear System of Equations in the Architecture of GPU Accelerator

Hrachya V. Astsatryan, Edita E. Gichunts

Institute for Informatics and Automation Problems of NAS RA
e-mail: hrach@sci.am, editagich@ipia.sci.am

Abstract

We consider some important issues related to the solution of linear system of equations that arise in multi-processor and graphics processing unit architecture. A more effective method for solving a linear system of equations is considered through the LU factorization. Investigations are conducted in case of general complex matrices, because for those matrices the random butterfly transformation is used. The paper presents performances of several ways of solving methods on the graphic processor NVIDIA K40c.

Keywords: LU factorization, linear system of equations, Random Butterfly Transformation, GEPP, GENP, MAGMA, GPU accelerator.

1. Introduction

Similar to LAPACK, MAGMA [1, 2, 3] is being built as a community effort, incorporating the newest developments in hybrid algorithms and scheduling, and aiming at minimizing synchronizations and communication in these algorithms. The goal of these efforts is to redesign the dense linear algebra algorithms in LAPACK to fully exploit the power of current heterogeneous systems of multi/manycore CPUs and accelerators, and deliver the shortest possible time to an accurate solution within the given energy constraints. Indeed, the algorithms included so far in MAGMA 1.6 manage to overcome bottlenecks associated with just multicore or GPUs, to significantly outperform the corresponding packages for any of these components taken separately.

For the linear system solvers on current multicore or GPU architectures, a bottleneck in terms of communication cost and parallelism comes from the pivoting, a technique used to prevent divisions by too-small numbers in the Gaussian Elimination (GE) process. Current libraries like LAPACK implement GE using a block algorithm, which factors the input matrix by iterating over its blocks of columns (panels). Pivoting not only requires communication (or

synchronization in a shared memory environment), but it also limits the exploitation of asynchronicity between the block operations.

The solution of linear system of algebraic equations has the form AX = B, where A is a square matrix, B is either a right side vector or a matrix, consisting of columns of the right sides. X is the solutions of system equations and it is a vector if B is a vector and it is a matrix, if B is a matrix.

Special block algorithms are used to solve the system equations. The system solution is divided into two parts:

1. System matrix factorization,
2. System solution through factorization.

Depending on the matrix feature the case of factorization is different. The following two cases of LU factorization are used for the general matrix:

1. The LU factorization with partial pivoting and row interchanges is used to factor A as A=PLU, where P is a permutation matrix, L is a lower triangular matrix, the main diagonal elements of which are 1, and U is the upper triangular matrix
2. The LU factorization with no pivoting is used to factor A as A = LU, where L is the lower triangular unit, and U is the upper triangular one.

Several ways of solutions of linear system of equations on NVIDIA K40c GPU accelerator are presented in the paper which are carried out through the two mentioned cases of LU factorization. They are presented only for general matrices, because to get high performance for them, Random Butterfly Transformation (RBT) was used which was repeatedly increasing the solution performance of the system of equations. Note that RBT has been studied in the systems with multicore [4] and distributed memory [5], but its performance has not been studied on GPU accelerator. Section 2 of the paper describes the cases of solving the linear system of equations. Section 3 presents the performances of solutions of the mentioned cases on GPU accelerator. Section 4 presents the conclusion.

2. Solution Methods

2.1 LU Factorization with Partial Pivoting

The LU factorization (or decomposition) of a matrix A has the form A = PLU, where L is a unit lower triangular matrix, U is an upper triangular matrix and P is a permutation matrix. The block LU factorization algorithm [6] proceeds in the following steps: initially, a set of NB columns (the panel) is factored and a pivoting pattern is produced. Then the elementary transformations, resulting from the panel factorization, are applied in block fashion to the remaining part of the matrix (the trailing submatrix). First, NB rows of the trailing submatrix are swapped, according to the pivoting pattern. Then a triangular solve is applied to the top NB rows of the trailing submatrix. Finally, matrix multiplication of the form $A_{ij} \leftarrow A_{ij} - A_{ik} \times A_{kj}$ is performed, where A_{ik} is the panel without the top NB rows, A_{kj} is the top NB rows of the trailing submatrix and A_{ij} is the trailing submatrix without the top NB rows. Then the procedure is applied repeatedly, descending down the diagonal of the matrix.

The solution of linear system of equations, where the LU factorization is made with Partial Pivoting, is performed by MAGMA 1.6.1 library through cgesv and cgesv_gpu functions. The difference between these two functions is as follows: in the first case the function itself carries the matrices from the CPU to GPU and vice versa, while in the second case it is realized by the user. The solution sequence is as follows:

- A and B matrices are transferred from the CPU to the global memory of GPU.
- LU factorization of the matrix A is performed through cgetrf_gpu function of MAGMA library using a partial pivoting with row interchanges.
- A * X = B is solved through the function cgetrs_gpu of MAGMA library.
- X derived solutions are transferred from the GPU to CPU.

To implement cgetrs_gpu function, claswp subprojects of lapackf77 library and ctrsm subprojects of MAGMA library are used. The claswp routine swaps rows of the trailing submatrix according to the pivoting pattern, established in the panel factorization. This operation only performs data motion and the GPUs are very sensitive to the matrix layout in memory. In raw-major layout, threads in a warp can simultaneously access consecutive memory locations. The ctrsm routine uses the lower triangle of the NB × NB diagonal block to apply triangular solve to the block of right-hand-sides formed by the top NB rows of the trailing submatrix. An efficient implementation of this routine on a GPU is difficult due to the data-parallel nature of GPUs and small size of the solve ($32 \leq NB \leq 288$) [7].

The claswp routines are implemented in LAPACK [8], while the ctrsm routines are the part of the Basic Linear Algebra Subroutines (BLAS [9]) standard. LAPACK is an academic project and, therefore, the source code is freely distributed online. BLAS is a set of standardized routines, and it is available in commercial packages (e.g., MKL [10] from Intel, ACML [11] from AMD, ESSL [12] from IBM), in academic packages (e.g., ATLAS [13]) and also as a reference implementation in FORTRAN 77 from the Netlib software repository.

2.2 LU Factorization without Pivoting

The LU factorization (or decomposition) of a matrix A consists of writing of that matrix as a matrix product A = LU, where L is the lower triangular and U is the upper triangular. It is a central kernel in linear algebra because it is commonly used in many important operations such as solving a nonsymmetric linear system, inverting a matrix, computing a determinant or an approximation of a condition number. LU decomposition is an algebraic process that transforms a matrix A into a product of a lower triangular matrix L the elements of which are only on the diagonal and below, and an upper triangular matrix U the elements of which are only on the diagonal and above determinant and the inverse of a matrix.

The solution of linear system of equations, where the LU factorization is made without Pivoting, is performed by MAGMA 1.6.1 library through cgesv_rbt and cgesv_nopiv_gpu functions. Here also in the first case the function itself carries the matrices from the CPU to GPU and vice versa, while in the second case it is realized by the user.

In case of xgesv_nopiv_gpu.cpp functions the solution sequence is as follows:

- A and B matrices are transferred from the CPU to the global memory of GPU.
- LU factorization of the matrix A is performed through cgetrf_nopiv_gpu function of MAGMA library without any pivoting.
- A * X = B is solved through the function cgetrs_nopiv_gpu of MAGMA library.
- X derived solutions are transferred from the GPU to CPU.

To implement cgetrs_nopiv_gpu function, only the ctrsm subproject of MAGMA library is used.

The cgesv_rbt function is the optimized version of the solution of linear system of equations.

The GENP algorithm can be unstable due to a potentially large growth factor. This is why we systematically perform iterative refinement on the computed solution of the randomized system. Algorithm 1 describes how the iterative refinement is performed in our implementations.

We improve the computed solution until we reach the required accuracy or we reach a defined maximum number of iterations.

Algorithm 1 Iterative refinement.

Input: A the original matrix.
Input:b the right hand side.
Input:x the computed solution.
Input:L and U the factorized form of A
Input:N size of the matrix A
Result: An improved solution x
1: EPS = Machine precision
2: ITERMAX = 30
3: ITER = 0
4: ANRM = $||A||_\infty$
5: XNRM = max|x|
6: Cte = ANRM * EPS * √N
7: r = b – Ax
8: RNRM = max|r|
9: **while** RNRM > XNRM **and** ITER < ITERMAX **do**
10: **Solve:** Ly = r
11: **Solve:**Uz = y
12: x = x+r
13: r = b – Ax
14: XNRM = max|x|
15: RNRM = max|r|
16: ITER = ITER + 1
17: **end while**

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have:
RNRM < SQRT(n)*XNRM*ANRM*EPS*BWDMAX where

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual
- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by SLAMCH('Epsilon')
- The values ITERMAX and BWDMAX are fixed to 30 and 1.0D+00, respectively.

Note that EPS is determined by xlamch("Epsilon") function of the lapackf77 library, and ANRM is determined by the xlange() function of the magmablas library.

The following consecutive steps are made for the solution of this case:

- A and B matrices are transferred from the CPU to the global memory of GPU.
- LU factorization of the matrix A is performed through cgetrf_nopiv_gpu function of MAGMA library without any pivoting.
- The cgesv_rbt function of the MAGMA library is called.

Note that the cgesv_rbt function takes the factorized A matrix and by the cgetrs_nopiv_gpu function it gets the solutions of linear system of equations, afterwards for iterative refinement it improves the computed solution to a system of linear equations.

Random Butterfly Transformation (RBT) is a randomization technique initially described by Parker and recently revisited for dense linear systems. The method for randomizing has been described in [14, 15]. It consists of a multiplicative preconditioning U^TAV where the matrices U and V are chosen among a particular class of random matrices called recursive butterfly matrices. Then Gaussian Elimination with No Pivoting (GENP) is performed on the matrix U^TAV and, to solve Ax = b, we instead solve $(U^TAV)y= U^Tb$ followed by x = Vy.

The solution of linear system of equations, where the random butterfly transformation is applied on A and B matrices and the LU factorization is made without Pivoting, is implemented through the cgerbt_gpu function of the MAGMA 1.6.1 library.

The implementation of this form of solution is realized in the following sequence:

- We generate the random matrices U and V in packed storage on the CPU.
- The matrix A and the packed representation of U and V are sent from the host memory to the device memory.
- Randomization is performed on the GPU, updating A in the device memory.
- Perform Partial Random Butterfly Transformation on the GPU with magmablas_cprbt() function.
- We compute U^Tb on the GPU, $A_ry= U^Tb$ is solved on the GPU, followed by the solution x = Vy with magmablas_cprbt_mtv() function.
- The solution is sent to the host memory.

3. Results of Experiments

The experiments were conducted on NVIDIA K40c GPU. The architecture of NVIDIA K40c consists of 2880 CUDA processor cores. It is endowed with much higher bandwidth 288 GB/s of message transfer between CPU and GPU, having 12 GB of global memory, GDDR5 memory interface, and CUDA C programming environment. The operation system of K40c is Ubuntu 14.04.2 LTS. MAGMA 1.6.1 package is installed. The code is compiled using the GNU gcc version 4.8, gfortran-4.8, g ++ - 4.8 and the nvcc version 7.0 with the optimization flag -O3 and linked with the Atlas Library.

Figures 1 and 2(a,b) show the time and performance schedules of methods for solving of linear system of equations.

The obtained results show that the performance of solutions defined by LU factorization without pivoting is higher than that with pivoting, especially when the solutions of Random Butterfly Transformation are repeatedly endowed with high performance. The results show that the performance of solutions defined by cgesv_rbt function is the lowest of the mentioned cases but it is the optimized version of solutions because for iterative refinement it improves the computed solution to a system of linear equations. For the solutions defined by LU factorization with pivoting the cgesv_gpu function performance is higher than that of cgesv function. This is because the data have been loaded into the global memory of GPU before the function appeal.

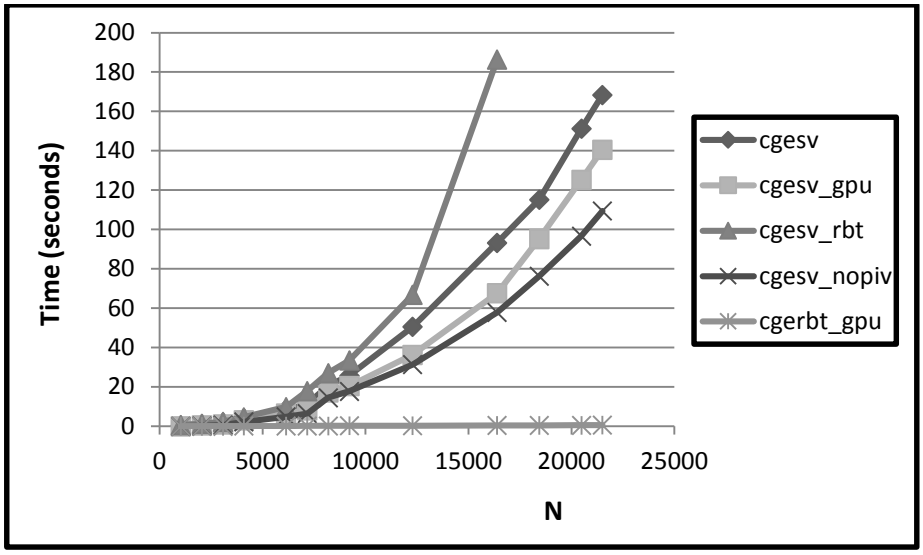
Time (seconds)
200
180
160
140
120
100
80
60
40
20
0
0
5000
10000
15000
20000
25000
N
cgesv
cgesv_gpu
cgesv_rbt
cgesv_nopiv
cgerbt_gpu

Fig. 1.

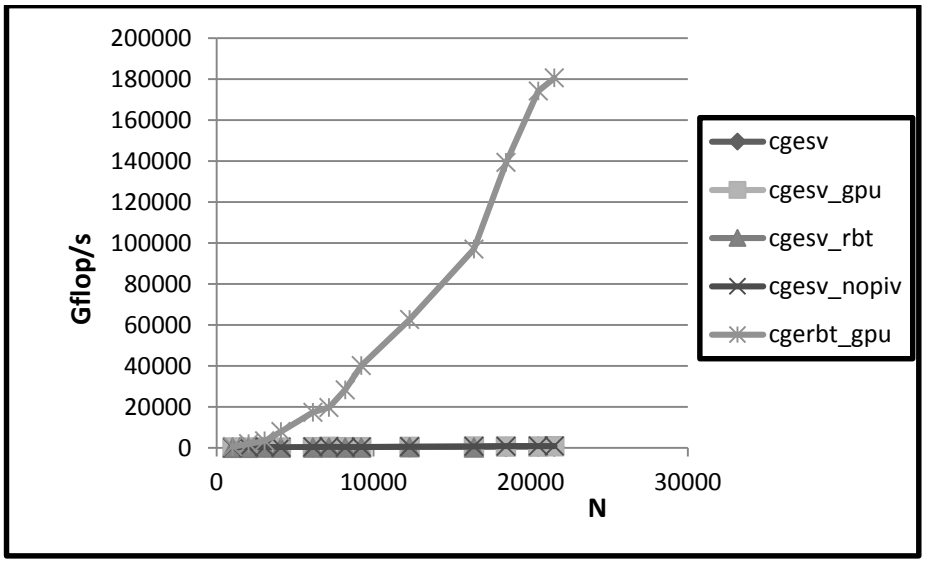
Gflop/s
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0
10000
20000
30000
N
cgesv
cgesv_gpu
cgesv_rbt
cgesv_nopiv
cgerbt_gpu

Fig. 2(a)

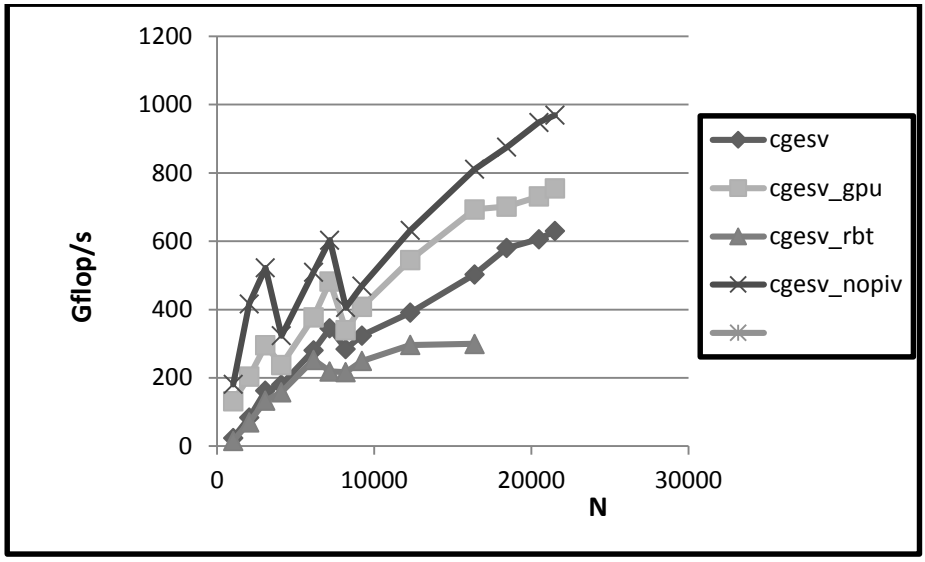

Fig. 2(b) without cgerbt_gpu.

3. Conclusion

We presented methods for solving of linear system of equations on GPU accelerator where the LU factorization is performed with and without pivoting. The received performance results lead to the following conclusion that to achieve high performance in solutions of linear system of equations, Random Butterfly Transformation solution method is definitely in the first place.

References

[1] R. Nath, S. Tomov and J. Dongarra, "An improved MAGMA GEMM for Fermi GPUs", *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 511–515, 2010.

[2] S. Tomov, J. Dongarra and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems", *Parallel Computing*, vol. 36(5&6), pp. 232–240, 2010.

[3] S. Tomov, R. Nath and J. Dongarra, "Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing", *Parallel Computing*, vol. 36, no. 12, pp. 645–654, 2010.

[4] M. Baboulin, D. Becker and J. J. Dongarra, "A parallel tiled solver for dense symmetric indefinite systems on multicore architectures", *Parallel & Distributed Processing Symposium (IPDPS)*, 2012.

[5] M. Baboulin, D. Becker, G. Bosilca, A. Danalis and J. J. Dongarra, "An efficient distributed randomized algorithm for solving large dense symmetric indefinite linear systems", *Parallel Computing*, vol. 40, no. 7 , pp. 212--223, 2014.

[6] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, 1997. ISBN: 0898713897

[7] J. Kurzak, P. Luszczek, M. Faverge, and J. Dongarra, "LU factorization with partial pivoting for a multicore system with accelerators", *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1613—1621, 2013.

[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK User's Guide*, SIAM, 1999, Third edition.
[9] K. Goto, GotoBLAS. Texas Advanced Computing Center, University of Texas at Austin, USA. http: // www. otc. utexas. edu/ ATdisplay. jsp, 2007.
[10] Intel. Math Kernel Library (MKL). http://www.intel.com/software/products/mkl/.
[11] AMD. AMD Core Math Library (ACML). [Online]. Available: http://developer. amd. com/acml. jsp, 2012.
[12] IBM Corporation. IBM Parallel Engineering and Scientific Subroutine Library. Guide and Reference. (GC23-3836), 1995.
[13] R. C. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software. Technical Report UT-CS-97-366, University of Tennessee, December 1997. [Online]. Available: http://www.netlib.org/lapack/lawns/lawn131.ps.
[14] D. S. Parker, "Random butterfly transformations with applications in computational linear algebra", Technical Report CSD-950023, UCLA Computer Science Department, 1995.
[15] D. S. Parker and B. Pierce, "The randomizing FFT: an alternative to pivoting in Gaussian elimination", Technical Report CSD-950037, Computer Science Department, UCLA, 1995.

Submitted 04.09.2015, accepted 12.01.2016

Գծային հավասարումների համակարգի լուծման մեթոդների արտադրողականությունները GPU արագագործչի ճարտարապետությունում

Հ. Ասցատրյան, Է. Գիչունց

Ամփոփում

Մենք դիտարկում ենք գծային հավասարումների համակարգի լուծմանը վերաբերող որոշ կարևորագույն հարցեր, որոնք առաջանում են բազմապրոցեսորային և գրաֆիկական պրոցեսորային ճարտարապետությունում: Դիտարկվում է LU վերլուծության միջոցով գծային հավասարումների համակարգի լուծման մեթոդներից ավելի արդյունավետ եղանակը: Ուսումնասիրությունները կատարվում են կոմպլեքս ընդհանուր մատրիցների դեպքում, քանի որ այդ մատրիցների համար կիրառվում է թիթեռնիկի պատահական ձևափոխությունը: Աշխատանքում ներկայացվում են մի քանի մեթոդներով լուծման եղանակների արտադրողականությունները NVIDIA K40c գրաֆիկական պրոցեսորի վրա:

Производительности методов решения систем линейных уравнений в архитектуре GPU ускорителя

Г. Асцатрян, Э. Гичунц

Аннотация

Рассмотрены некоторые важные вопросы, связанные с решением систем линейных уравнений, возникающих в многопроцессорных и графических процессорных архитектурах. С помощью LU факторизации рассматривается более эффективный метод для решения линейной системы уравнений. Исследования проводятся для случая общих комплексных матриц, потому что для этих матриц используется случайное преобразование бабочки. В статье представлены производительности методов нескольких способов решения на графическом процессоре NVIDIA K40c.