Mixed Precision Dense Linear System Solvers for High Performance Reconfigurable Computing

JunKyu Lee, Gregory D. Peterson
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee, USA
[jlee57, gdp]@utk.edu

Robert J. Harrison, Robert J. Hinde
Department of Chemistry
University of Tennessee
Knoxville, Tennessee, USA
[robert.harrison, rhinde]@utk.edu

***Abstract*— The iterative refinement method for linear system solvers can improve performance while maintaining numeric accuracy. Previous work addressing iterative refinement exploits single precision and double precision for CPU, GPU, or Cell/BE processors. Due to only two different precisions supported, iterative refinement is limited on those platforms. Reconfigurable Computing (RC) is a great candidate to exploit iterative refinement since it is able to employ any precision computation as long as the hardware resources are sufficient. In iterative refinement for RC, the choice of working precision for Gaussian elimination is extremely important since its computational complexity is $O(n^3)$ while the other steps are $O(n^2)$. In this paper, we explore RC architecture and working precision for Gaussian elimination to obtain both high performance and satisfactory numerical solutions.**

I. INTRODUCTION

Iterative refinement is a method proposed in 1948 to refine the solution of linear system solvers to obtain both numerically accurate results and high performance [1, 3-6]. Many applications in computational science can be described as dense linear system solvers [2]. For example, electromagnetic, quantum scattering, and large least-squares problems are part of dense linear system solver applications. Extensive previous work addresses iterative refinement, but mostly uses single and double precision computation [1, 3, 4]. There is previous work [6] proposing a high performance RC architecture which employs Field Programmable Gate Arrays (FPGAs) for Gaussian Elimination with Partial Pivoting (GEPP) supporting multiple different precisions to solve a linear system, followed by a microprocessor for iteratively updating the solution and residual. The GEPP was implemented on FPGAs in [6]. RC is able to exploit the iterative refinement while employing variable precision for GEPP. The iterative refinement algorithm is described in Procedure I. Most of the computation is performed in Step 1. We recognize the initial precision, P_I, is important due to the $O(n^3)$ operation. For example, if we choose unnecessarily high precision, it may degrade the performance. If we choose too low precision, the $O(n^3)$ computations may not produce a good numerical solution.

This work was partially supported by the National Science Foundation, grant NSF CHE-0625598.

Procedure I:

Step 1: GEPP (A); -- $O(n^3)$ <= precision P_I;
Solve $LUx(1) = P \times b$; -- $O(n^2)$ <= precision P_I;
for (i = 1 to max iterations or x(i) accurate enough)
Step 2: $r(i) = b - A_H x(i)$; -- $O(n^2)$ <= precision P_H;
Step 3: $LUz(i) = P \times r(i)$; -- $O(n^2)$ <= precision P_I;
Step 4: $x(i) = x(i) + z(i)$; -- $O(n^2)$ <= precision P_H;
end

where P is a permutation matrix generated by GEPP, P_I represents a low precision (initial precision), P_H is a high precision, and A_H is a matrix represented by the high precision.

In spite of the importance of the choice of the initial precision, we did not find any previous work exploring how to choose the initial working precision for GEPP, although others have noted that condition numbers are related to obtaining satisfactory numerical solutions [1, 4, 5]. Dense matrices may have structure according to their applications. For example, Vandermonde matrices, Toeplitz matrices, or orthogonal matrices have their own structures [2]. We investigate uniformly distributed random dense matrices in this paper. We note that

1. Random dense matrices have a relation between the condition numbers and the matrix size.
2. There is a relation between the convergence rates and the condition numbers.
3. A condition number estimator for triangular matrices takes $O(n^2)$ [7].

We propose a dense linear system solver for high performance RC based on 1, 2, and 3. This research differs from previous work as follows:

- This is the first approach to choose the appropriate initial working precision for GEPP by exploiting matrix attributes.
- This is the first failure predictor employing either convergence rate or condition number estimation and providing an appropriate precision for GEPP.

II. METHODOLOGY

The results from the methodology are based on software numerical simulation. In order to apply different working precisions for GEPP, we implemented arithmetic functions (e.g., addition, subtraction, multiplication, and division) working with various mantissa bit width (8 - 52 bits) in C. We tested 1000 n×n uniformly distributed random matrices for n = 32, 64, 128, and 256. 1000 512×512 test matrices are additionally examined to find the Cumulative Density Function (CDF) of the condition numbers of the matrices. The test matrices are generated by the modified lagged Fibonacci generator with default parameters given by the Scalable Parallel Random Number Generators (SPRNG) library [8].

We first seek the required mantissa bit widths (W_r) for a 95% convergence success rate using statistics for the matrices and 10^{-9} accuracy. Then, we seek the 95% convergence success condition numbers (K_m) in the CDFs (the maximum condition number such that 95% of matrices will converge). Since the 95% convergence success rate using the statistics mainly depends on condition numbers, we consider W_r is the minimum sufficient mantissa bit width when the condition number = K_m.

We assume that the elements of dense matrices in the applications are uniformly distributed. If the dense matrices have different characteristics, then a different empirical approach may be required.

A. *Initial working precision for GEPP*

The initial working precision for GEPP is important for high performance RC. In order to pick the appropriate initial precision for GEPP, we investigate the convergence success rate which produces high precision results according to different matrices sizes. We pick 10^{-9} as a user accuracy for 2 norm forward error. We consider solutions using GEPP and three iterations with double precision as the correct solutions for comparing error. Table I shows the success rate representing the percentage converging to high accuracy (10^{-9}) results when we apply different mantissa bit widths for GEPP. We estimated the mantissa bit widths for 95% successful convergence rate using linear interpolation in Table I. Fig. 1 shows the relation between the required mantissa bit widths for GEPP and matrix's size for 95% successful convergence rates.

TABLE I. SUCCESSFUL CONVERGENCE RATE

Mantissa bit width	Matrix size			
	32×32	64×64	128×128	256×256
8 bits	19.1 %	0 %	0 %	0 %
12 bits	***85.5 %***	57.8 %	8.7 %	0 %
16 bits	***98.6 %***	***94.9 %***	***81.1 %***	49 %
20 bits	99.8 %	**99.5 %**	***97.9 %***	***93.6 %***
24 bits	100 %	100 %	99.8 %	***99.5 %***
More than 28 bits	100 %	100 %	100 %	100 %
95% W_r	14.90 bits	16.09 bits	19.31 bits	20.95 bits

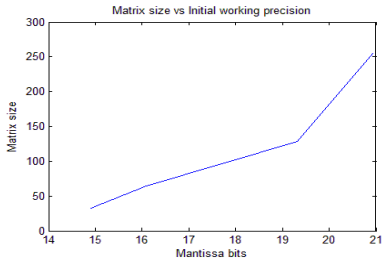

Figure 1. Matrix sizes vs required mantissa bit widths for 95% successful convergence

Based on Fig. 1, we employ linear fitting to describe the relation between the matrix sizes and mantissa bit widths. Based on this fit, we can apply the appropriate initial precision for GEPP according to a system matrix size, statistically expecting 95% success with 10^{-9} accuracy. The mantissa bit-widths for 95% convergence success vary according to user-required accuracy. However, the maximum condition numbers for 95% successful convergence rate hardly change since the success rates mainly depend on the matrix's condition number [1, 4, 6]. Therefore, we assume the maximum condition numbers with 95% success do not change according to user accuracies.

B. *Failure predictor for 5% failure*

When the condition numbers increase, the probability of failure increases [1, 4, 5, 6]. We empirically seek the condition numbers associated with 95% of the CDF according to five different size matrices. A matrix having a condition number less than the 95% point generally guarantees high likelihood of success. The condition numbers are computed using the LAPACK `dgecon` function.

Fig. 2 shows the probability density function of the condition numbers for 1000 512×512 matrices. Fig. 3 shows the CDF from the 1000 test matrices and the relation between matrix size and the condition number pointing out 95% of CDFs. Table II shows the condition numbers.

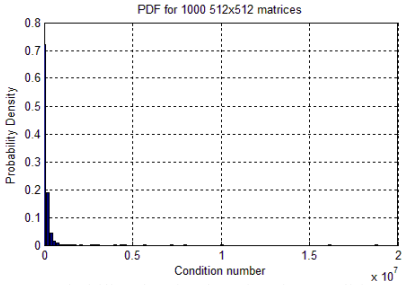

Figure 2. Probability density function for condition numbers

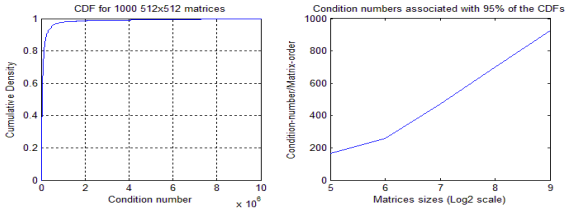

Figure 3. Cumulative density function for condition numbers

TABLE II. CONDITION NUMBERS POINTING OUT 95% OF CDFS

Matrices Sizes	32×32	64×64	128×128	256×256	512×512
95% CNs	5273	16590	60520	179000	475300

The relation between the condition numbers and matrix sizes can be described by linear fitting. Based on this fit, we can estimate the 95% condition number in the CDF according to matrix size.

We propose two possible approaches for the failure predictor. One approach is to use the condition number of the matrix to estimate failure likelihood. The condition number estimation requires $O(n^2)$ [7]. The other approach is to exploit the relation between convergence rates and condition numbers. The convergence rate is defined as *"$||x(1)||_2/||z(1)||_2$"* in Procedure I. We find that the convergence rate is high when the condition number is low. The convergence rate estimation takes O(n) since the complexity comes from the dot product of x(1) and z(1). The $O(n^2)$ or O(n) computation hardly affect the overall performance compared to the $O(n^3)$ GEPP operation.

In the first approach, the computed condition number can be compared to the condition number of the linear fit of Fig. 3. If the computed condition number is greater than the condition number of the linear fit of Fig. 3, the algorithm predicts the failure and computes GEPP again with higher precision. Similarly, if the convergence rate is too low, the GEPP is performed again with higher precision.

III. RECONFIGURURABLE COMPUTING ARCHITECTURE

Based on the observations from the methodology, we propose a high performance RC architecture for a dense linear system solver for a uniformly distributed matrix. Fig. 4 shows the high performance RC architecture.

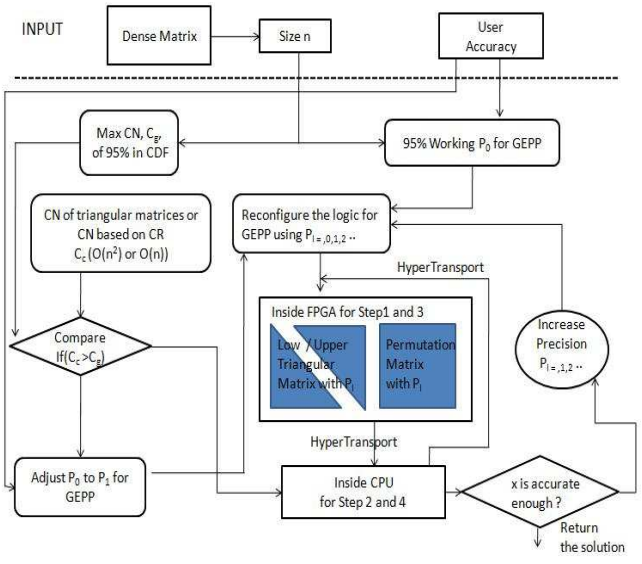

Figure 4. Proposed reconfigurable computing architecture

Based on a matrix's size and user accuracy, the P_0 is calculated using the linear fit of Fig. 1. GEPP logic for P_0 is reconfigured on FPGAs. After GEPP is done on FPGAs, the condition number is computed in $O(n^2)$ [7]. If the computed condition number is larger than the condition number of the linear fit of Fig. 3, the failure flag is generated. Once the flag is generated, the P_0 is changed to P_1 using both the linear fit of Fig. 3 and Table I and II. If the algorithm does not produce a satisfactory numerical solution using P_1, the working precision is increased by a step size.

IV. PERFORMANCE MODELING

We consider only GEPP computation in performance modeling since the complexity of GEPP is $O(n^3)$ while the others are $O(n^2)$. We set several variables for the performance modeling. We consider that applying P_0 provides around 95% success probability.

T_{P0} : the execution time for GEPP with P_0,
T_{P1} : the execution time for GEPP with P_1,
(if applying P_0 does not work for the iterative refinement),
ε_1, ε_2, and ε_3: very small numbers.
T: the total execution time for iterative refinement.

$$T = (0.95 \pm \varepsilon_1) \times T_{P0} + (0.05 \pm \varepsilon_2) \times (T_{P0} + T_{P1}) + \varepsilon_3 \times (etc)$$

(etc: time it takes unless applying P_1 produces success)

V. CONCLUSIONS

The RC architecture selects the initial working precision with 95% success expectation based on the nature of the system matrix. The failure predictor also improves the performance not only because the number of iterations is decreased but also because it chooses a new appropriate precision for GEPP.

For future work, performance from GPU, CPU, Cell/BE, and RC implementations can be compared according to system matrix condition number.

REFERENCES

[1] Demmel, J., et al., Error bounds from extra-precise iterative refinement. ACM Trans. Math. Softw., 2006. 32(2): p. 325-351.

[2] Edelman, A., Large dense numerical linear algebra in 1993: The parallel computing influence. Journal of supercomputing applications, 1993. (7): p. 113-128.

[3] Kurzak, J. and J. Dongarra, Implementation of mixed precision in solving systems of linear equations on the Cell processor: Research Articles. Concurr. Comput. : Pract. Exper., 2007. 19(10): p. 1371-1385.

[4] Langou, J., et al., Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems), in Proceedings of the 2006 ACM/IEEE conference on Supercomputing. 2006, ACM: Tampa, Florida.

[5] Moler, C.B., Iterative Refinement in Floating Point. J. ACM, 1967. 14(2): p. 316-321.

[6] Sun, J., G.D. Peterson, and O.O. Storaasli, High-Performance Mixed-Precision Linear Solver for FPGAs. IEEE Trans. Comput., 2008. 57(12): p. 1614-1623.

[7] Viswanath, D. and L,N. Trefethen, Condition numbers of random triangular matrices, 1998, 19(2): p. 564-581.

[8] Scalable parallel pseudo random number generators library, http://sprng.fsu.edu/