A Square-Root-Free Matrix Decomposition Method for Energy-Efficient Least Square Computation on Embedded Systems

Fengbo Ren, Chenxin Zhang, Liang Liu, Wenyao Xu, Viktor Öwall, and Dejan Marković

***Abstract*—QR decomposition (QRD) is used to solve least-squares (LS) problems for a wide range of applications. However, traditional QR decomposition methods, such as Gram–Schmidt (GS), require high computational complexity and nonlinear operations to achieve high throughput, limiting their usage on resource-limited platforms. To enable efficient LS computation on embedded systems for real-time applications, this paper presents an alternative decomposition method, called QDRD, which relaxes system requirements while maintaining the same level of performance. Specifically, QDRD eliminates both the square-root operations in the normalization step and the divisions in the subsequent backward substitution. Simulation results show that the accuracy and reliability of factorization matrices can be significantly improved by QDRD, especially when executed on precision-limited platforms. Furthermore, benchmarking results on an embedded platform show that QDRD provides constantly better energy-efficiency and higher throughput than GS-QRD in solving LS problems. Up to 4 and 6.5 times improvement in energy-efficiency and throughput, respectively, can be achieved for small-size problems.**

***Index Terms*—Computational complexity, energy efficiency, least-squares problem, matrix factorization, QR decomposition.**

I. Introduction

QR DECOMPOSITION (QRD) is one of the most widely used matrix factorization techniques for solving least-squares (LS) problems in various applications. Because of its practical importance, enormous algorithmic efforts have been made to improve energy efficiency and routine performance of QRD over the past few decades, especially from a very large scale integration (VLSI) design point of view. Two hardware-friendly methods for computing QRD are commonly used in practice. *Givens rotation* (GR) offers a low-cost computation method by handling one element of a matrix at a time with only additions and table look-ups involved. However, the inherent *sequential* processing of GR limits its applications with stringent real-time requirements [1]. Alternatively, the *Gram–Schmidt* (GS) algorithm allows *parallel* computations with higher processing throughput by factorizing a matrix at the vector-level. Therefore, GS-QRD is often in favor for throughput-driven applications, where the parallelization of computations is desired [2], [3].

However, neither of the above methods are ideal for real-time implementations on embedded systems. Due to limited memory and computation resources available on such platforms, GR-QRD will have limited throughput while GS-QRD will face design challenges on precision loss and energy-efficiency [1]–[4]. Prior analysis has revealed that nonlinear operations in GS-QRD, namely square-root operations and divisions, are the primary cause of the aforementioned design issues [2]. Nonlinear operations are not cost-effective for an explicit implementation, and they are usually the accuracy bottleneck of the overall system. Most existing work implements nonlinear operations using iterative approximation methods, such as Newton–Raphson, with word-length optimization applied [2]–[4]. Nonetheless, these methods trade computational complexity with throughput or accuracy, which is not desired for real-time applications.

In this paper, we tackle the abovementioned issues jointly by deriving an alternative QDR decomposition (QDRD) method that has both the throughput advantage of GS-QRD and the intrinsic square-root-free feature of GR-QRD. Specifically, a new diagonal factorization matrix $\boldsymbol{D}$ is introduced as the scaling factor to exclude the normalization process of orthogonal basis. As a result, both the square-root operations and the divisions in the backward substitution for solving LS problems are eliminated. To the best of our knowledge, a similar idea has been suggested in [5] while this is the first paper that analyzes and evaluates its performance for LS computation based on hardware implementation results.

II. The QDRD Method for Solving LS Problems

A LS problem is formulated as finding an $x = \hat{x}$ that gives

$$min \parallel \boldsymbol{A}x - y\|_2^2 \tag{1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ (usually $m \geq n$) and $y \in \mathbb{R}^m$. If $\boldsymbol{A}$ has a zero-null space, the solution to (1) can be computed by solving the normal equation given as

$$\boldsymbol{A^T A}x = \boldsymbol{A^T}y. \tag{2}$$

Manuscript received May 30, 2014; accepted August 18, 2014. Date of publication August 22, 2014; date of current version November 20, 2014. This manuscript was recommended for publication by A. Coskun

F. Ren and D. Marković are with the Department of Electrical Engineering, University of California, Los Angeles, CA 90095 USA (e-mail: renfengbo@ucla.edu; dejan@ee.ucla.edu).

C. Zhang, L. Liu, and V. Öwall are with the Department of Electrical and Information Technology, Lund University, Lund 221 00, Sweden (e-mail: chenxin.zhang@eit.lth.se; Liang.Liu@eit.lth.se; Viktor.Owall@eit.lth.se).

W. Xu is with the Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 14214 USA (e-mail: wenyaoxu@buffalo.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2014.2350997

According to QRD, a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with a zero-null space can be decomposed as

$$\boldsymbol{A} = \boldsymbol{QR} \tag{3}$$

where $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$ is an orthonomal matrix with $\boldsymbol{Q^T Q} = \boldsymbol{I}$ ($\boldsymbol{I} \in \mathbb{R}^{n\times n}$ is identity matrix) and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is an upper-triangular matrix with positive diagonal elements. By substituting $\boldsymbol{A}$ with (3), (2) becomes

$$\boldsymbol{R}x = \boldsymbol{Q^T}y. \tag{4}$$

Since $\boldsymbol{R}$ is an upper-triangular matrix, (4) can be solved by using backward substitution.

Note that in GS-QRD, square-root operations are required for computing the diagonal elements of $\boldsymbol{R}$, $\mathrm{diag}(\boldsymbol{R})$, as Euclidean norms [2]. In addition, the i^{th} column of $\boldsymbol{Q}$ and the i^{th} row of $\boldsymbol{R}$ (except for the diagonal elements) is normalized by the i^{th} element of $\mathrm{diag}(\boldsymbol{R})$, respectively. Therefore, there must exist a diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ with $\mathrm{diag}(\boldsymbol{D}) = \mathrm{diag}(\boldsymbol{R})$, and unnormalized matrices $\hat{\boldsymbol{Q}} \in \mathbb{R}^{m\times n}$ and $\hat{\boldsymbol{R}} \in \mathbb{R}^{n\times n}$ that satisfy $\boldsymbol{Q} = \hat{\boldsymbol{Q}}\boldsymbol{D}^{-1}$ and $\boldsymbol{R} = \boldsymbol{D}^{-1}\hat{\boldsymbol{R}}$, where $\boldsymbol{D}$ serves as a normalizer for both $\hat{\boldsymbol{Q}}$ and $\hat{\boldsymbol{R}}$. According to this, (4) can be formulated as

$$\boldsymbol{D}^{-1}\hat{\boldsymbol{R}}x = \boldsymbol{D}^{-1}\hat{\boldsymbol{Q}}^{\boldsymbol{T}}y. \tag{5}$$

Note that the normalizer $\boldsymbol{D}$ appears on both sides of (5) and cancels each other, which indicates that the normalization of factorization matrices in QRD is essentially redundant for solving LS problems. Therefore, the square-root operations involved in the normalization process are also redundant.

A. Square-Root-Free QDRD

Based on the above analysis, we propose an alternative factorization method that essentially eliminates the square-root operations by removing the normalizer $\boldsymbol{D}$ from both $\boldsymbol{Q}$ and $\boldsymbol{R}$ in (3) as $\boldsymbol{A} = (\boldsymbol{QD}^{-1})(\boldsymbol{DD})(\boldsymbol{D}^{-1}\boldsymbol{R})$. By defining $\boldsymbol{Q'} = \boldsymbol{QD}^{-1} = \hat{\boldsymbol{Q}}\boldsymbol{D}^{-2}$, $\boldsymbol{R'} = \boldsymbol{D}^{-1}\boldsymbol{R} = \boldsymbol{D}^{-2}\hat{\boldsymbol{R}}$, and $\boldsymbol{D'} = \boldsymbol{DD} = \boldsymbol{D}^2$, we derive the QDRD of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ that has a zero-null space as

$$\boldsymbol{A} = \boldsymbol{Q'D'R'} \tag{6}$$

where $\boldsymbol{Q'} \in \mathbb{R}^{m\times n}$ is an orthogonal matrix, $\boldsymbol{D'} \in \mathbb{R}^{n\times n}$ is a diagonal matrix, and $\boldsymbol{R'} \in \mathbb{R}^{n\times n}$ is an upper-triangular matrix.

The new factorization matrices have several important properties. Since $\boldsymbol{D}$ is a diagonal matrix, we can derive

$$\boldsymbol{Q'^T Q'} = \left(\boldsymbol{QD}^{-1}\right)^{\boldsymbol{T}}\left(\boldsymbol{QD}^{-1}\right) = \boldsymbol{D}^{-2} = \boldsymbol{D'}^{-1}. \tag{7}$$

Equation (7) indicates that $\boldsymbol{Q'}$ is an orthogonal matrix, which can be normalized by $\boldsymbol{D'}$ as

$$\boldsymbol{Q'^T Q'D'} = \boldsymbol{I}. \tag{8}$$

Additionally, from $\mathrm{diag}(\boldsymbol{D}) = \mathrm{diag}(\boldsymbol{R})$ and $\boldsymbol{R'} = \boldsymbol{D}^{-1}\boldsymbol{R}$, we must have $\mathrm{diag}(\boldsymbol{R'}) = \vec{1}$, meaning the diagonal elements of $\boldsymbol{R'}$ must be all unit ones.

The QDRD of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ shown in (6) can be computed column-wise in n iterations. In each iteration, (6) can be partitioned as

$$[a_1 \boldsymbol{A_2}] = [{q'}_1 \boldsymbol{Q'}_{\boldsymbol{2}}]\begin{bmatrix} {d'}_{11} & \vec{0} \\ \vec{0} & \boldsymbol{D'_{22}} \end{bmatrix}\begin{bmatrix} 1 & R'_{12} \\ \vec{0} & \boldsymbol{R'_{22}} \end{bmatrix} \tag{9}$$

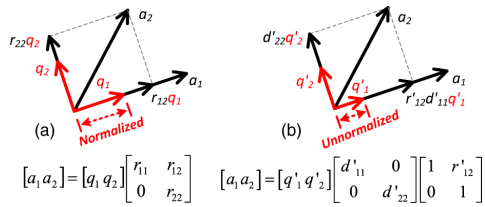

Fig. 1. Illustrations of (a) the QRD and (b) the QDRD of a 2×2 matrix.

where $a_1, {q'}_1 \in \mathbb{R}^{m\times 1}$ is the column vector of $\boldsymbol{A}$ and $\boldsymbol{Q'}$, $R'_{12} \in \mathbb{R}^{1\times(n-1)}$ is a row vector of $\boldsymbol{R'}$, ${d'}_{11}$ is a scalar element of $\mathrm{diag}(\boldsymbol{D'})$, respectively. Given that $\boldsymbol{Q'_2} \in \mathbb{R}^{m\times(n-1)}$ and $\boldsymbol{D'_{22}}, \boldsymbol{R'_{22}} \in \mathbb{R}^{(n-1)\times(n-1)}$ are known submatrices from the previous iteration, the unknown variables ${q'}_1$, ${d'}_{11}$, and R'_{12} can be derived as follows. According to (7), we know that

$${{q'}_1}^T {q'}_1 = 1/{d'}_{11} \tag{10}$$

and

$$\boldsymbol{Q'_2}^{\boldsymbol{T}} {q'}_1 = \vec{0}. \tag{11}$$

In addition, by expanding (9) we have

$$a_1 = {d'}_{11} \cdot {q'}_1 \tag{12}$$

and

$$\boldsymbol{A_2} = {d'}_{11} \cdot {q'}_1 R'_{12} + \boldsymbol{Q'}_{\boldsymbol{2}}\boldsymbol{D'_{22}}\boldsymbol{R'_{22}}. \tag{13}$$

Therefore, from (10) and (12), ${d'}_{11}$ can be derived as ${a_1}^T a_1 = {{d'}_{11}}^2 \cdot {{q'}_1}^T {q'}_1 = {d'}_{11}$, and ${q'}_1$ can be calculated as ${q'}_1 = a_1/{d'}_{11}$. R'_{12} can be derived by multiplying both sides of (13) by ${{q'}_1}^T$ as

$${{q'}_1}^T \boldsymbol{A_2} = {d'}_{11} \cdot {{q'}_1}^T {q'}_1 R'_{12} + {{q'}_1}^T \boldsymbol{Q'}_{\boldsymbol{2}}\boldsymbol{D'_{22}}\boldsymbol{R'_{22}}. \tag{14}$$

Inserting (10) and (11), (14) can be simplified to ${{q'}_1}^T \boldsymbol{A_2} = R'_{12}$. As indicated by (13), the QDRD of the next iteration is given by

$$\boldsymbol{A_2} - {d'}_{11} \cdot {q'}_1 R'_{12} = \boldsymbol{A_2} - a_1 R'_{12} = \boldsymbol{Q'}_{\boldsymbol{2}}\boldsymbol{D'_{22}}\boldsymbol{R'_{22}}. \tag{15}$$

Note that by introducing the new factorization matrix $\boldsymbol{D'} = \boldsymbol{D^2}$ to avoid the normalization process, the proposed QDRD method completely eliminates the square-root operations. To better explain this concept in comparison to QRD, the two different matrix decompositions of a 2×2 matrix are illustrated using 2-D vectors in Fig. 1. Note that the basis vectors in QRD (q_i) are always *orthonormal*, while those in QDRD (${q'}_i$) are scaled by r_{ii} and no longer normalized.

B. Solving LS Problems using QDRD

To solve LS problems using the proposed QDRD method, the normal equation in (2) can be substituted by (6) and simplified according to (8) as

$$\boldsymbol{R'}x = \boldsymbol{Q'^T}y \tag{16}$$

then solved through backward substitution. Equation (16) also indicates that solving LS problems only requires finding the directions of the orthogonal basis of $\boldsymbol{A}$, regardless of the normalization of their Euclidean norms. Therefore, the introduced

scaling factor $\boldsymbol{D}'$ in QDRD can be disregarded in the backward substitution. Moreover, since the diagonal elements of $\boldsymbol{R}'$ are all unit ones in QDRD, the divisions required in the backward substitution in (16) can be safely eliminated.

III. Matrix Factorization Performance Analysis

In general, two characteristics of the factorization matrices are critical to the numerical accuracy of the solutions to LS problems. One is how well the factorization matrices can reconstruct the original matrix $\boldsymbol{A}$. The other is how close the transpose of the orthogonal matrix is to its inverse. In this study, we define two new generic metrics to quantify both of the criteria. The reconstruction signal to noise ratio (RSNR) for QRD and QDRD is defined as $RSNR_{\mathrm{QRD}} = 20\log_{10}(\frac{\|\boldsymbol{A}\|_F}{\|\boldsymbol{A}-\boldsymbol{Q}\boldsymbol{R}\|_F})$, and $RSNR_{\mathrm{QDRD}} = 20\log_{10}(\frac{\|\boldsymbol{A}\|_F}{\|\boldsymbol{A}-\boldsymbol{Q}'\boldsymbol{D}'\boldsymbol{R}'\|_F})$, respectively, where the operator $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Note that RSNR measures the reconstruction error of the factorization matrices relative to the original matrix as the ratio of $\boldsymbol{A}$'s energy over that of the reconstruction error. Using a similar concept, the orthogonality signal to noise ratio (OSNR) for QRD and QDRD is defined as $OSNR_{\mathrm{QRD}} = 20\log_{10}(\frac{\|\boldsymbol{I}\|_F}{\|\boldsymbol{Q}^T\boldsymbol{Q}-\boldsymbol{I}\|_F})$, and $OSNR_{\mathrm{QDRD}} = 20\log_{10}(\frac{\|\boldsymbol{I}\|_F}{\|\boldsymbol{Q}'^T\boldsymbol{Q}'\boldsymbol{D}'-\boldsymbol{I}\|_F})$, respectively. Ideally, if infinite precision can be preserved during matrix factorization, both OSNR and RSNR should approach positive infinity. In practice, both metrics are subject to quantization noise. Given a fixed machine precision, higher OSNR and RSNR performance indicates the superiority of the factorization method in terms of accuracy and reliability.

Matrix factorization experiments are carried out in MATLAB using double precision in order to compare the performance of QDRD against GS-QRD. The input matrices $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ are randomly generated for different sizes ranging from 10 to 100 with an aspect ratio of $m = n$. For each matrix size, 1000 trials are performed using Gaussian random matrices. The impact of the precision of nonlinear operations is investigated by injecting rounding errors at different precision—their results are rounded to different number of significant bits (ranging from 10 to 52 bits). Besides, all linear operations are performed with the highest precision to exclude their effects in the analysis. Note that the upper bound of the precision is limited by the word-length of the mantissa in double precision.

Fig. 2 shows the comparison results measured from general matrix factorizations. The RSNR and OSNR gain of QDRD (over GS-QRD) is calculated as $RSNR_{\mathrm{QDRD}} - RSNR_{\mathrm{QRD}}$ and $OSNR_{\mathrm{QDRD}} - OSNR_{\mathrm{QRD}}$, respectively. Note that both results are averaged across all matrix sizes. Figure 2(a) shows that QDRD provides nearly the same RSNR performance as GS-QRD, indicating the independence of RSNR performance on the precision of square-root operations. On the other hand, QDRD is able to achieve significant OSNR gain, especially when the precision of square-root operations dominates the overall OSNR performance ($w_{\mathrm{sqrt}} \leq w_{\mathrm{div}}$). It is because the rounding errors of square-root operations can be accumulated in GS-QRD over iterations, which deteriorates its OSNR as a dominating factor. On the contrary, QDRD is square-root-free, and thereby its OSNR will not have such precision loss. Overall, QDRD is guaranteed to provide more accurate results than GS-QRD when nonlinear operations are performed given precision constraints.

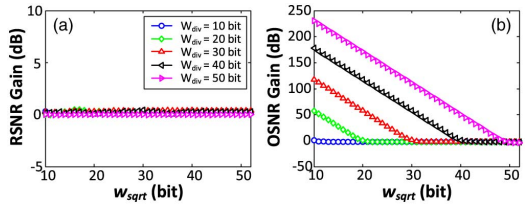

Fig. 2. Average (a) RSNR and (b) OSNR gain of QDRD over GS-QRD with respect to the word-length of square-root (w_{sqrt}) and divisions (w_{div}).

IV. Evaluation of LS Computation on Embedded Systems

In order to evaluate the performance of LS computation on embedded systems, we implemented the LS solver on an ultra-low-power MSP430 microprocessor using GS-QRD and QDRD, respectively. Based on the results, we analyze and compare the computational complexity and the throughput of the two methods. Potential applications are also discussed.

A. Experimental Platform

The embedded platform is equipped with an ultra-low-power 16-bit RISC MSP430F2274 microprocessor. The system has a supply voltage of 3.3 V and an operation frequency of 1 MHz. The computational resources in the hardware include a customized 16-bit multiplier and an arithmetic logical unit (ALU) that contains a 16-bit full adder. The microprocessor has a 256 Kb local register file with external access to a 1 KB RAM and a 32 KB Flash memory. Overall, 27 instructions and seven addressing modes are supported.

B. Computational Complexity Analysis

To simplify the analysis, we approximate the computational complexity by normalizing different operations to that of a 1-bit addition. Note that the complexity here refers to the hardware cost and indicates the energy consumption rather than the speed of the operation. Specifically, our analysis assumes that a w-bit addition has a complexity of w and a w-bit multiplication has one of w^2. Since only adder and multiplier are available, a w-bit division is given a complexity of $(w + 2w^2)k$, assuming the Newton-Raphson method is used to approximate $y = 1/x$ by computing $y_{i+1} = y_i(2 - xy_i)$ in k iterations with a given initial value y_0. Similarly, a w-bit square-root operation has a complexity of $(w + 2w^2)k^2 + wk$ assuming $y = \sqrt{x}$ is approximated by computing $y_{i+1} = \frac{1}{2}(y_i + x/y_i)$ using the same method.

Based on the above models, Table I summarizes the overall computational complexity of solving a LS problem using GS-QRD and QDRD, respectively, on the embedded platform. Both methods have the same operation count in terms of linear operations and require $mn^2 + mn$ additions and $mn^2 + 2mn$ multiplications for computing the matrix factorization shown in (3) and (6), respectively. In addition, $mn + n^2/2$ additions and multiplications are involved in calculating the backward substitution shown in (4) and (16). However, QDRD cuts down significantly on the nonlinear operations required. Solving a LS problem using GS-QRD requires n square-root operations and $2n$ divisions, while only n divisions with no square-root

TABLE I
THE COMPUTATIONAL COMPLEXITY OF SOLVING LS PROBLEMS

Method	Complexity*
GS-QRD	$\left(mn^2+\frac{n^2}{2}+2mn\right)(w+w^2)$ $+n(2k+k^2)(w+2w^2)+nw(mw+k)$
QDRD	$\left(mn^2+\frac{n^2}{2}+2mn+kn\right)(w+w^2)+nkw^2$

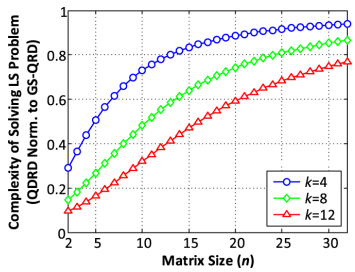

Fig. 3. Computational complexity of solving LS problems using QDRD with different matrix sizes. The result is normalized to that of using GS-QRD assuming different iterations k of the Newton–Raphson method.

operations are necessary for QDRD. Therefore, QDRD always has a lower computational complexity than QRD. Moreover, since the complexity also reflects the total energy consumption of the hardware platform, QDRD is more energy-efficient for solving LS problems.

Fig. 3 shows the computational complexity of solving LS problems of different sizes by using QDRD in the case of $m = n$ and $w = 16$. The complexity is normalized to that of GS-QRD for comparison. Note that when higher precision is required, more iterations k of the Newton-Raphson method have to be performed in GS-QRD. In contrast, QDRD is free of such precision loss, thereby leading to a larger complexity reduction as k increases. In addition, complexity reduction for QDRD becomes more prominent for small matrices, where the overall complexity is less dominated by additions and multiplications.

C. Throughput Evaluation from Hardware Emulation

To investigate the algorithm performance in practice, we implement both algorithms in C language and compile them with the IAR Embedded Workbench tool [6] on MSP430F2274 for emulation. In the compiler settings, the optimization level is set to high for both the C codes (function in-lining, instruction scheduling, common subexpression elimination, etc., are performed). Due to the loop-carried data dependency in both algorithms, the computation in different loops cannot be pipelined. Consequently, the hardware throughput is inversely proportional to the execution time of the algorithm. Table II summarizes the measured execution time of solving a single LS problem at different sizes when $m = n$ and $k = 8$. In this case, the number of clock cycles required to execute an addition, multiplication, division, and square-root operation is 4, 6, 128, and 1056, respectively. Clearly, executing nonlinear operations is orders of magnitude slower, and such overhead degrades the performance. As shown in Table II, by eliminating the square-root operations and alleviating divisions, QDRD is able to provide constantly higher throughput than GS-QRD in solving LS problems. Particularly, a speedup of 6.5 times can be achieved for small-size problems ($n \leq 8$).

TABLE II
THE EXECUTION TIME OF SOLVING A SINGLE LS PROBLEM

Clock Cycles	Matrix Size (n)				
	2	4	8	16	32
GS-QRD	2,828	6,384	17,600	69,888	401,408
QDRD	436	1,552	7,744	49,408	357,376
Speedup	6.5x	4.1x	2.3x	1.4x	1.1x

D. Potential Applications

Many real-life applications of LS problems can benefit from using QDRD, especially when implemented on resource- or energy-limited platforms. Multiple-input–multiple-output (MIMO) communication is one of such applications, where a small-size LS problem (depending on the antenna count) needs to be solved for signal detection [1]. As the MIMO technique has been adopted in modern wireless standards, using QDRD can benefit the low-power communication systems. For instance, in an 4-antenna MIMO system ($m = n = 4$), up to 4 times better energy-efficiency and higher throughput can be achieved compared to GS-QRD (see Fig. 3 and Table II). Other application examples include the real-time curve-fitting used in control systems, such as motion controllers [7]. In these applications, the size of LS problems is limited by the number of fitting parameters and data samples. Adopting QDRD can potentially improve the battery life and real-time performance of such applications.

V. CONCLUSION

Square-root operations required in QRD are intrinsically redundant for solving LS problems. To enable efficient LS computation on embedded systems for real-time applications, this paper presents an alternative QDRD method that relaxes the system requirements to achieve high throughput. Simulation results confirm that QDRD provides more accurate and reliable results than GS-QRD for general matrix factorizations. Furthermore, benchmarking results based on an MSP430 microprocessor testbed show that, due to complexity reduction of nonlinear operations, QDRD provides constantly better energy-efficiency and higher throughput than GS-QRD in solving LS problems. More specifically, up to 4 and 6.5 times improvement in energy efficiency and throughput, respectively, can be achieved for small-size problems.

REFERENCES

[1] Y.-H. Zheng *et al.*, "Efficient implementation of QR decomposition for gigabit MIMO-OFDM systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 10, pp. 2531–2542, Oct. 2011.
[2] P. Luethi *et al.*, "Gram-Schmidt-Based QR decomposition for MIMO detection: VLSI implementation and comparison," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Nov. 2008, pp. 830–833.
[3] S. Aslan, "Realization of area efficient QR factorization using unified division, square-root, and inverse square-root hardware," in *Proc. IEEE Int. Conf. Electr. Inf. Technol.*, Jun. 2009, pp. 245–250.
[4] B. Gestner, "VLSI implementation of a lattice reduction algorithm for low-complexity equalization," in *Proc. IEEE Int. Conf. Circuits Syst. Commun.*, May 2008, pp. 643–647.
[5] Å. Björck, "Numerical methods for least square problems," *SIAM*, p. 62, 1996.
[6] "IAR Embedded Workbench for TI MSP430," [Online]. Available: http://www.iar.com/Products/IAR-Embedded-Workbench/TI-MSP430/
[7] J.-B. Wang *et al.*, "Universal Real-Time NURBS interpolator on a PC-Based Controller," *Int. J. Adv. Manufacturing Technol.*, vol. 71, no. 1–4, pp. 297–507, Mar. 2014.