2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

Applying Modified Householder Transform to Kalman Filter

Farhad Merchant*, Tarun Vatwani‡, Anupam Chattopadhyay§, Soumyendu Raha¶,
S K Nandy†, Ranjani Narayan‖ and Rainer Leupers*
*Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany
{farhad.merchant, leupers}@ice.rwth-aachen.de
†CADLab, Indian Institute of Science, Bangalore, India 560012
{{tarun}@cadl, {raha,nandy}@cds}.iisc.ac.in
‡Indian Institute of Technology, Jodhpur
§School of Computer Engineering, Nanyang Technological University, Singapore
anupam@ntu.edu.sg
¶Scientific Computation Laboratory, Indian Institute of Science, Bangalore, India 560012
‖Morphing Machines Pvt. Ltd.
ranjani.narayan@morphingmachines.com

***Abstract*—Kalman filter (KF) is a key operation in many engineering and scientific applications ranging from computational finance to aircraft navigation. Recently, there have been proposals in the literature for acceleration of KF using modified Faddeeva algorithm (MFA) where the classical Householder transform (HT) is used in implementation of MFA on a custimizable platform called REDEFINE. REDEFINE is a coarse-grained reconfigurable architecture that has capabilities of recomposing data-paths at run-time and on-demand. In this paper, we present realization of KF using MFA where we implement MFA using modified Householder transform (MHT) presented in the literature. We call this as M^{2}FA. It is shown that the implementation of KF using M^{2}FA clearly outperforms the implementation of KF using MFA on REDEFINE and also the realization of KF on REDEFINE is scalable. Performance improvements over state-of-the-art implementations are also discussed.**

***Index Terms*—kalman filter, instruction level parallelism, state estimation, reconfigurable computing**

I. Introduction

Kalman filter (KF) is the most common and popular data fusion technique in use today [1] [2]. KF has applications ranging from vehicle tracking to freeway traffic estimation [3] [4]. Typically, KF is rich in matrix operations and parallel realization of KF poses challenges in acceleration to meet the real-time constraints demanded by the application domain [1]. Recently, modified Faddeeva algorithm (MFA) is contemplated as a prime candidate due to versatility offered by the MFA [1] [5]. In the MFA, a compound matrix M is created using input matrices A, B, C, and D and based on the initial values of the input matrices a desired output is obtained by computing a schur complement that is $D+CA^{-1}B$ [5] [6]. The compound matrix M is given by

$$M = \begin{bmatrix} A & B \\ -C & D \end{bmatrix} \quad (1)$$

Computing schur complement has two steps: 1) upper triangularize matrix A and update matrix B, and 2) annihilate matrix C using diagonal elements of upper triangularized matrix A and update matrix D. For upper triangularization of matrix A, QR factorization that is based on Givens rotation (GR) or Householder transform (HT) (geqrf) is used while for annihilation of matrix C, LU factorization (getrf) without pivoting is used. For updating trailing matrices that are B and D, general matrix multiply (gemm) is used. All these routines exhibit time complexity of $O(n^3)$ for problem size of $n \times n$. GR is preferred for sparse matrices and exhibits fine grained parallelism while HT is preferred for dense matrices and has fewer computations compared to GR and exhibits coarse grained parallelism [7] [8].

There have been several proposals in the literature for acceleration of these routines through algorithm-architecture co-design where macro operations in the routines are identified and realized on a reconfigurable data-path (RDP) [9] [10]. The RDP is tightly coupled to a processing element (PE) pipeline [11] [12]. In this paper, we present acceleration of MFA by applying modified Householder transform (MHT) instead of classical HT where MHT is presented in [13]. We identify macro operations in MHT (now onward dgeqr2ht), dgemm, and dgetrf and realize them in PE along with RDP where letter "d" is to indicate double precision routines. The realization of MFA with dgeqrf is termed as MFA while the realization of MFA with dgeqr2ht is termed as M^{2}FA. We choose REDEFINE CGRA as a platform for parallel realization of MFA and M^{2}FA since the REDEFINE is capable of composing and executing custom instructions at run-time [14] [15]. Further description of micro-architecture of REDEFINE is provided in section II-A2. Major contributions in this paper are as follows:

- It is proposed to replace classical HT in MFA by MHT where MHT is capable of exhibiting 25% more parallelism compared to the classical HT as presented in [13]
- Macro operations in the classical HT and MHT are identified and realized on the RDP that is tightly coupled

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00092

to the PE that results in accelerated implementation of KF over state-of-the-art realizations of KF
- MFA and M^{2}FA are realized on REDEFINE where different configurations of REDERINE are used to show scalablility in our solution

For our experiments, we have used floating point unit (FPU) presented in [16] with recommendations on optimal pipeline depths provided in [17]. In future, we plane to replace the FPU by posit arithmetic unit presented in [18]. Organization of the paper is as follows: In section II background on MHT and REDEFINE is discussed along with some of the recent implementations of KF. In section III, implementation of M^{2}FA is presented and also implementation of KF with M^{2}FA is presented. Implementation in the PE and results are discussed in section IV. We conclude our work in V.

II. Background and Related Work

MHT is discussed in section II-A1 and REDEFINE CGRA and merits of REDEFINE CGRA are discussed in section II-A2. Recent implementations of KF are discussed in section II-B.

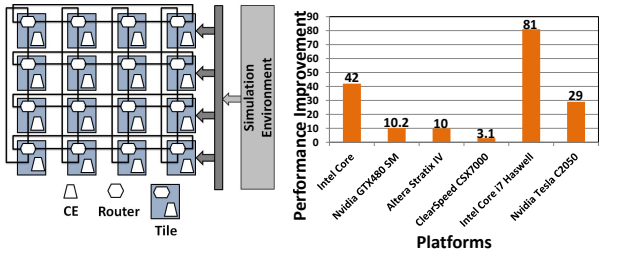
A. Background

1) Modified Householder Transform: MHT is presented in [8] with extensive study in [13]. The MHT pseudo-code is reproduced in algorithm 1. It can be observed in the algorithm

Algorithm 1 Pseudo code of Modified Householder Transform

```
1: Allocate memories for input matrix
2: for i = 1 to n do
3:     Compute Householder vectors for block column m × k
4:     Compute PA where PA = A − 2vv^T A
5: end for
```

1 that the innermost loops in the classical HT are fused to arrive at the MHT. While quantifying parallelism in MHT, it is observed that the MHT is capable of exhibiting 25% higher parallelism compared to the classical HT and in custom implementation of MHT, it is observed that the MHT is able to attain performance that is as good as the performance of dgemm [13]. Further information on comparison of classical HT, MHT and their implementation can be found in [8] and [13].



(a) REDEFINE CGRA

(b) Performance of REDEFINE PE in dgemm over multicore, GPGPU, FPGA and ClearSpeed Platforms

Fig. 1. REDEFINE CGRA and Perofrmance of REDEFINE PE in dgemm

2) REDEFINE CGRA: REDEFINE is a CGRA where several Tiles are connected through a packet switched network-on-chip (NoC) as shown in figure 1(a). Each Tile consists of a compute element (CE) and a Router. Acceleration for an application domain in REDEFINE is attained by attaching a custom function unit (CFU) in the CEs of REDEFINE where CFU is tailored for a particular application domain. In the literature it is shown that the REDEFINE is capable of attaining superior performance for dense linear algebra (DLA) computations and signal processing applications [19] [20]. Performance in PE of REDEFINE for dgemm compared to other platforms is shown in figure 1(b).

B. Related Work

Due to immense popularity and wide range of applicability, there is a rich literature on implementation of KF. Traditionally, systolic arrays have been preferred for implementation of KF since KF is rich in matrix computations and systolic arrays are the most suitable platforms for matrix operations [21]. Due to non-scalability of systolic arrays, in the recent implementations, a systolic array with a scalable controller is preferred that ensures scalability. Once such implementation is presented in [1] where authors have shown that with a RDP that is tightly coupled to the pipeline of a PE can attain 3-100x better performance over state-of-the-art platforms like multicores, field programmable gate arrays (FPGAs) and general purpose graphics processing units (GPGPUs). A low cost laboratory implementation that is based on FPGA is presented in [22]. The FPGA implementation presented in [22] is not scalable and also does not focus on acceleration of KF on an FPGA. A real-time estimation of a vehicle tire velocity and acceleration is presented in [23]. Major drawbacks of the extended KF implementation presented [23] are scalability and compatibility since the implementation uses custom format for arithmetic instead of standard fixed or floating point formats. In this paper, we present acceleration of KF through algorithm-architecture co-design where we first propose to replace classical classical HT by MHT. Later, we present systematic acceleration of MFA and M^{2}FA through algorithm-architecture co-design. We use these implementations as a basic building block in KF and show that the performance attained in KF with M^{2}FA is better than the performance attained in KF with MFA.

III. M^{2}FA in KF

A simplistic KF is shown in figure 2 where shaded blocks depict matrix computations. It can be observed in the figure 2 that the KF is rich in matrix operations and MFA is a right candidate for realization of these matrix operations due to versatility of MFA.

In [1], the authors have shown that the HT attains up to 62-66% of the theoretical platform of the REDEFINE PE. It is also shown that the performance attained in the REDEFINE CGRA is scalable. Recalling equation 1 the compound matrix M is given by

$$M = \begin{bmatrix} A & B \\ -C & D \end{bmatrix} \tag{2}$$

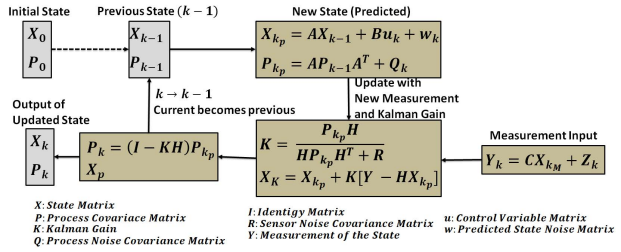


Fig. 2. A Simplistic Kalman Filter

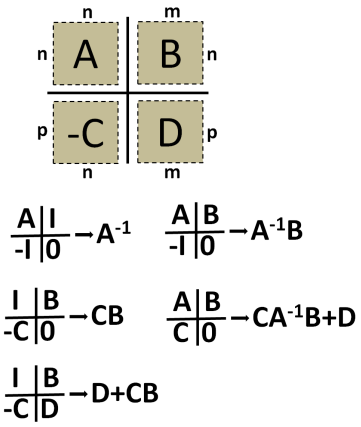


Fig. 3. Faddeeva Algorithm and Its Versatility

where A, b, C and D can be of the sizes $n \times n$, $n \times m$, $p \times n$ and $p \times m$ respectively as shown in figure 3. Different operations that can be supported with MFA is also depicted in the figure 3. MFA has two steps: i) upper triangularization of matrix A, and ii) annihilation of matrix C using diagonal elements of the matrix A. Matrices B and D are updated in the step i and step ii respectively. MFA is an improvement over classical Faddeeva algorithm where orthogonal transform is applied for upper triangularization of the matrix A. This improvement leads to higher numerical stability in Faddeeva algorithm as explained in [5]. Pseudo code for MFA is shown in algorithm 2. For the first step where QR factorization is computed to upper triangularize the matrix A, we propose to apply MHT presented in [13] instead of classical HT.

Algorithm 2 M^2FA: Pseudo code of Modified Faddeeva Algorithm with MHT

1: Allocate memories for matrices
2: Upper triangularize matrix A using MHT
3: Update matrix B
4: Annihilate matrix C using diagonal elements of matrix A
5: Update matrix D that results in Schur complement $D + CA^{-1}B$

Applying algorithm 1 to the MFA results in M^2FA operation given in algorithm 2. This is a similar routine as classical MFA presented in [1]. Due to fused inner and intermediate loops in MHT, the amount of parallelism exhibited in MFA. As explained in [13], MHT exhibits 1.33x higher parallelism than classical HT. Theoretical comparison of latencies between

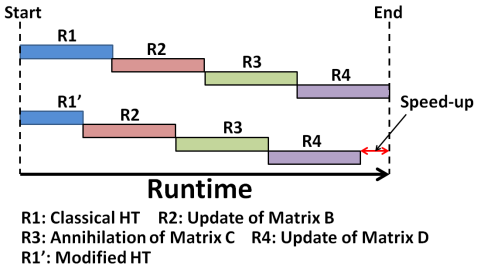


Fig. 4. Theoretical View of MFA and M^{2}FA

MFA and M^{2}FA is shown in figure 4. It can be observed in the figure 4 that in M^{2}FA the overall latency of the operation is reduced due to application of MHT since MHT is faster than classical HT. To get experimental results, we realize MFA in the PE. In implementation, a software pipeline scheme is also applied which is applicable to MFA and M^{2}FA.

IV. Implementation and Results

The PE design is shown in figure 5. Operation of the PE is described in the following steps:

- Load input data from global memory (GM) to the local memory (LM) where GM could be upper level of the memory while LM is part of Load-Store CFU as depicted in the figure 5
- Load data from LM to the register file (RF) where RF is part of floating point sequencer (FPS)
- Perform computations on the data and store the results back to LM from RF
- Store the final result to GM while reuse the intermediate results for further computations

As explained in [20], such a PE is capable of attaining 90% of computation to communication overlap and 74% of the theoretical peak performance in double precision general matrix multiplication (dgemm) at a very low power and area. An RDP is also depicted in the figure 5 where macro operations identified in the DLA computations can be realized. We apply similar technique that is presented in [13] for realization of M^{2}FA.

Contents of the different memories in the PE while realizing MFA and M^{2}FA are shown in figure 6. The FPS is responsible for performing only computations where the instructions are written in terms of the operations that are realized on RDP. The memories in the Load-Store CFU are responsible for memory transactions. Performance of PE in terms of theoretical peak performance is shown in figure 7(a). The functions $DHT()$ implements classical HT, $DGEMM()$ implements dgemm routine, $DMHT()$ is MHT realization, $DUPDATEB()$ is updating of matrix B, $DLU()$ is LU factorization without partial pivoting and $DUPDATED()$ is updating of matrix D. The prefix "D" suggests that the arithmetic used for realization of these routines is a double precision arithmetic. It can be observed here that the communication pattern in

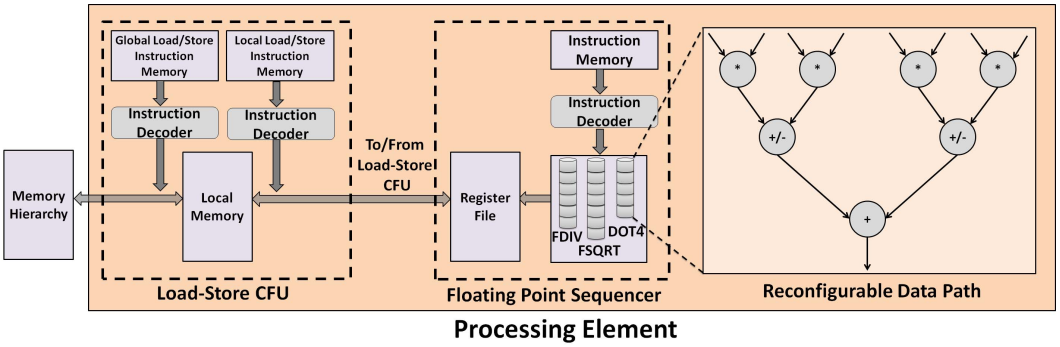


Fig. 5. Processing Element Design

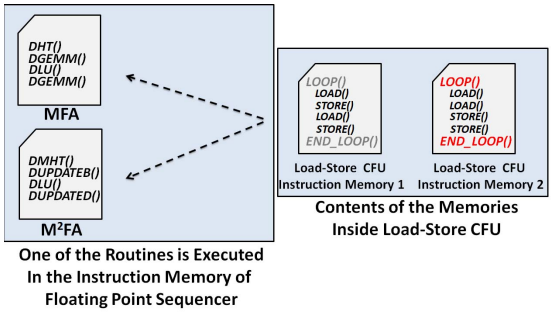
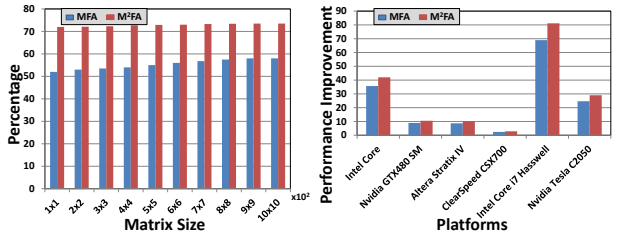


Fig. 6. Contents of Different Memories in the PE while Implementing MFA and M^{2}FA

MFA and M^{2}FA is observed to be identical and hence the implementation can be carried out by replacing the code in the FPS without affecting memory contents of the Load-Store CFU.



(a) Performance of MFA and M^{2}FA (b) Improvement in MFA and M^{2}FA In Terms of Theoretical Peak of the over Other Platforms PE

Fig. 7. Performance of MFA and M^{2}FA

It can be observed in the figure 7(a) that the performance attained by M^{2}FA is 1.27x higher than that of the performance attained by MFA in the PE. M^{2}FA is capable of achieving close to 74% of the theoretical peak of the PE that is also performance attained by dgemm. This is counter-intuitive as in classical DLA software packages, the attained performance of the routines pertaining to QR factorization are limited by the performance of the corresponding gemm routines used to implement the QR factorization routines. In this case, since we are using operations that are beyond traditional dgemm updates, the performance is not limited by dgemm and we are able to exploit higher parallelism in the routines than classical QR factorization routines.

Performance improvement in MFA and M^{2}FA in PE over other platforms is shown in figure 7(b). The performance improvement reported in the figure 7(b) is in terms of Gflops/Watt. It can be observed that the performance improvement in M^{2}FA is around 10% higher than that of the performance improvement in MFA. This is mainly due to the reduction in run time in M^{2}FA. For realization of MFA and M^{2}FA on multicore and GPGPUs, we have used open source packages Parallel Linear Algebra Software for Multicore Architectures (PLASMA) and Matrix Algebra on GPU and Multicore Architectures (MAGMA) respectively [24] [25].

A. Parallel Realization of KF in REDEFINE CGRA

We realize MFA and M^{2}FA and then KF that is based on MFA and M^{2}FA in REDEFINE CGRA. For efficient realization the PE is attached as a CFU in REDEFINE CGRA as shown in figure 8. The mapping strategies for computing $D + CA^{-1}B$ are depicted in figure 9.

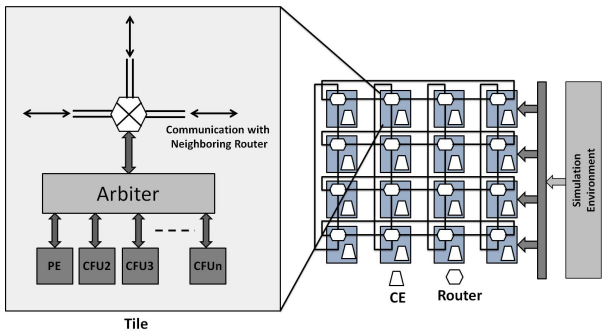


Fig. 8. REDEFINE CGRA with PE as a CFU

We use different configurations named configuration 1 (2×2 Tile array), configuration 2 (3×3 Tile array), and configuration 3 (4×4 Tile array) as depicted in the figure 9. The mapping for 2×2 Tile array is described in the following steps:

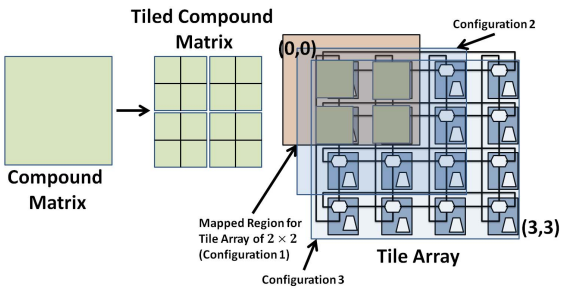


Fig. 9. Mapping Strategies for MFA and M^{2}FA in REDEFINE CGRA

- Partition matrices A, B, $-C$ and D into sub-matrices of size 2×2
- Load sub-matrix of matrix A into Tile (0,0) and perform the QR factorization using HT/MHT
- Update the trailing matrix of matrix A and matrix B. For updating matrix B, use Tile (0,1) by copying Q matrix that is produced in QR factorization
- In parallel to updating matrix B in Tile (0,1), annihilate matrix C in Tile (0,1) using diagonal elements of upper triangularized matrix A
- Update matrix D using the matrix generated while annihilating matrix C

The above mentioned method can be applied to any Tile array size. Generalizing above steps for $P \times P$ Tile array, it would require to partition the matrices A, B, $-C$ and D into sub-matrices of size $P \times P$. Performance comparison of

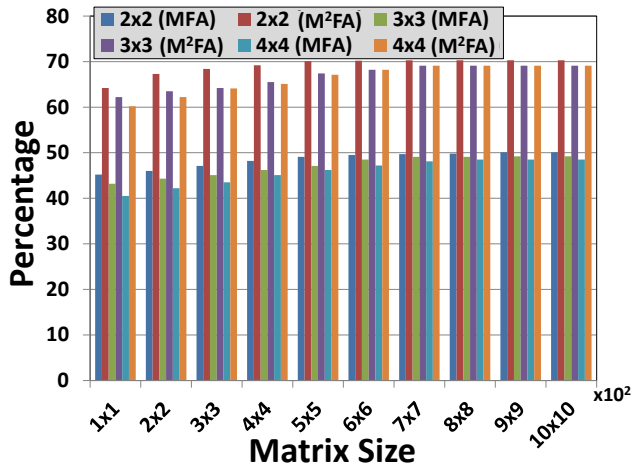


Fig. 10. Performance of MFA and M^{2}FA in REDEFINE with Different Configurations of REDEFINE

MFA and M^{2}FA in different configurations of REDEFINE is depicted in figure 10. It can be observed in the figure 10 that the performance attained in M^{2}FA is close to 70% of the theoretical peak of REDEFINE and 50-60% higher than that of performance attained by the MFA. It can also be observed that the performance attained by M^{2}FA is 94.5% of the theoretical peak attained by general matrix-multiply.

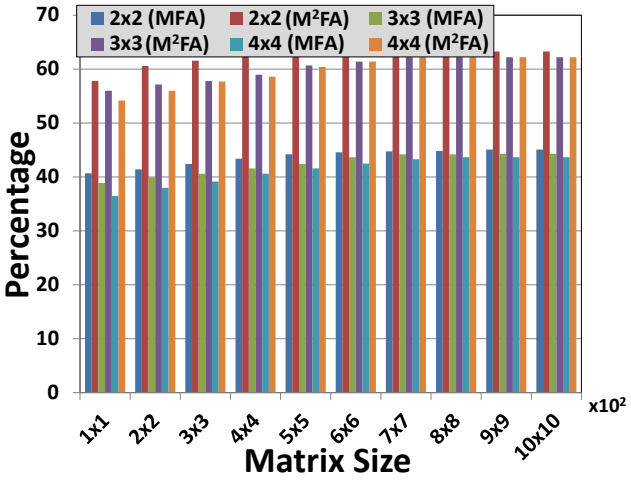


Fig. 11. Performance of KF with MFA and with M^{2}FA in REDEFINE with Different Configurations of REDEFINE

Performance of KF with MFA and M^{2}FA is shown in figure 11 in different configurations of REDEFINE. It can be observed in the figure 11 that the performance in KF in REDEFINE is 95% of the performance attained by MFA or M^{2}FA. It can also be observed from the figures 10 and 11 that the realizations in REDEFINE are scalable.

V. Conclusion

A detailed realization of Kalman filter is presented and compared with the most recent realization where Kalman filter was implemented using modified Faddeeva algorithm. It was proposed to apply modified Householder transform to accelerate the modified Faddeeva algorithm. Realization of modified Faddeeva algorithm with classical Householder transform and modified Householder transform was presented in a processing element where it was shown that the performance attained is 74% of the theoretical peak of the processing element. Performance improvement in the processing element over state-of-the-art realizations is observed to be 2-70x in case of modified Faddeeva algorithm with classical Householder transform while the performance improvement is observed to be 4-80x in case of modified Faddeeva algorithm with modified Householder transform. The processing element was attached to REDEFINE coarse-grained reconfigurable architecture as a custom function unit and it was shown that the performance in modified Faddeeva algorithm with classical Householder transform is dropped to 45-50% of the theoretical peak of the REDEFINE in different configurations while performance in modified Faddeeva algorithm with modified Householder transform is 65-70% in different configurations of REDEFINE. Kalman filter is realized in REDEFINE and similar trend is observed in the performance as observed in the modified Faddeeva algorithm with classical Householder transform and with the modified Householder transform.

References

[1] F. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Achieving efficient realization of kalman filter on cgra

through algorithm-architecture co-design," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Eds. Cham: Springer International Publishing, 2018, pp. 119–131.

[2] R. Faragher, "Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, Sept 2012.

[3] C. Wang, B. Ran, H. Yang, J. Zhang, and X. Qu, "A novel approach to estimate freeway traffic state: Parallel computing and improved kalman filter," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 180–193, Summer 2018.

[4] R. S. Tomar and M. S. P. Sharma, *One Dimensional Vehicle Tracking Analysis in Vehicular Ad hoc Networks*. Cham: Springer International Publishing, 2018, pp. 255–270.

[5] J. G. Nash and S. Hansen, "Modified faddeeva algorithm for concurrent execution of linear algebraic operations," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 129–137, Feb 1988.

[6] R. W. Cottle, "Manifestations of the schur complement," *Linear Algebra and its Applications*, vol. 8, no. 3, pp. 189 – 211, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0024379574900664

[7] A. George and J. W. Liu, "Householder reflections versus givens rotations in sparse orthogonal decomposition," *Linear Algebra and its Applications*, vol. 88-89, pp. 223 – 238, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002437958790111X

[8] F. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Achieving efficient qr factorization by algorithm-architecture co-design of householder transformation," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, Jan 2016, pp. 98–103.

[9] F. Merchant, A. Chattopadhyay, G. Garga, S. K. Nandy, R. Narayan, and N. Gopalan, "Efficient qr decomposition using low complexity column-wise givens rotation (cgr)," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Jan 2014, pp. 258–263.

[10] S. Das, K. Madhu, M. Krishna, N. Sivanandan, F. Merchant, S. Natarajan, I. Biswas, A. Pulli, S. Nandy, and R. Narayan, "A framework for post-silicon realization of arbitrary instruction extensions on reconfigurable data-paths," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 592 – 614, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762114000836

[11] Z. E. Rkossy, F. Merchant, A. Acosta-Aponte, S. K. Nandy, and A. Chattopadhyay, "Efficient and scalable cgra-based implementation of column-wise givens rotation," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, June 2014, pp. 188–189.

[12] ——, "Scalable and energy-efficient reconfigurable accelerator for column-wise givens rotation," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2014, pp. 1–6.

[13] F. A. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Efficient realization of householder transform through algorithm-architecture co-design for acceleration of qr factorization," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[14] M. Alle, K. Varadarajan, A. Fell, R. R. C., N. Joseph, S. Das, P. Biswas, J. Chetia, A. Rao, S. K. Nandy, and R. Narayan, "Redefine: Runtime reconfigurable polymorphic asic," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 2, pp. 11:1–11:48, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1596543.1596545

[15] F. A. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, R. N. S. K. Nandy, and R. Leupers, "A systematic approach for acceleration of matrix-vector operations in cgra through algorithm-architecture co-design," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, (in press) VLSI Design 2019.

[16] F. Merchant, N. Choudhary, S. K. Nandy, and R. Narayan, "Efficient realization of table look-up based double precision floating point arithmetic," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, Jan 2016, pp. 415–420.

[17] F. Merchant, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Accelerating blas and lapack via efficient floating point architecture design," *Parallel Processing Letters*, vol. 27, no. 03n04, p. 1750006, 2017. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0129626417500062

[18] R. Chaurasiya, G. John, R. Shrestha, N. Jonathan, N. Sangeeth, N. Kaustav, F. Merchant, and R. Leupers, "Parameterized posit arithmetic hardware generator," in *2018 IEEE International Conference on Computer Design (ICCD)*, (in press) ICCD 2018.

[19] M. Mahadurkar, F. Merchant, A. Maity, K. Vatwani, I. Munje, N. Gopalan, S. K. Nandy, and R. Narayan, "Co-exploration of nla kernels and specification of compute elements in distributed memory cgras," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, July 2014, pp. 225–232.

[20] F. Merchant, A. Maity, M. Mahadurkar, K. Vatwani, I. Munje, M. Krishna, S. Nalesh, N. Gopalan, S. Raha, S. K. Nandy, and R. Narayan, "Micro-architectural enhancements in distributed memory cgras for lu and qr factorizations," in *2015 28th International Conference on VLSI Design*, Jan 2015, pp. 153–158.

[21] T. Y. Sung and Y. H. Hu, "Parallel vlsi implementation of the kalrman filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-23, no. 2, pp. 215–224, March 1987.

[22] A. V. R. Teja, V. Verma, and C. Chakraborty, "A new formulation of reactive-power-based model reference adaptive system for sensorless induction motor drive," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 11, pp. 6797–6808, Nov 2015.

[23] F. Sandhu, H. Selamat, S. E. Alavi, and V. B. S. Mahalleh, "Fpga-based implementation of kalman filter for real-time estimation of tire velocity and acceleration," *IEEE Sensors Journal*, vol. 17, no. 17, pp. 5749–5758, Sept 2017.

[24] J. Kurzak, P. Luszczek, A. YarKhan, M. Faverge, J. Langou, H. Bouwmeester, and J. Dongarra, "Multithreading in the plasma library," *Multi and Many-Core Processing: Architecture, Programming, Algorithms, & Applications*, 2013.

[25] B. J. Smith, "R package magma: Matrix algebra on gpu and multicore architectures, version 0.2.2," September 3, 2010, [On-line] http://cran.r-project.org/package=magma.